HTTP-PALVELUIDEN SUORITUSKYKYTESTAUSOHJELMISTO

LAHDEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
26.4.2007
Saku Kekkonen

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

KEKKONEN, SAKU:

HTTP-palveluiden suorituskykytestausohjelmisto

Ohjelmistotekniikan opinnäytetyö, 52 sivua, 9 liitesivua

Kevät 2007

TIIVISTELMÄ

Opinnäytetyössä suunnitellaan ja toteutetaan testausohjelmisto HTTP-palveluiden suorituskyvyn testaukseen. Ohjelmisto tukee Windows- sekä Linux-käyttöjärjestelmää käyttäen avoimen lähdekoodin ohjelmistokirjastoja. Ohjelmiston pääasiallisina vaatimuksia ovat HTTP- ja HTTPS-protokollien tuki, useiden yhtäaikaisten käyttäjien simulointi sekä mittaustulosten keruu ja esitys.

Työssä tutkitaan selaimen ja palvelimen välistä toimintaa peruskäyttötapauksissa. Tutkittavana ovat sivujen lataukseen sekä tietojen välitykseen liittyvät HTML-elementit. HTML-tasolta syvennytään tutkimaan selaimen ja palvelimen välistä HTTP-protokollaa. HTTP-protokollaa tutkitaan, jotta suunniteltava ohjelmisto osaisi tulkita sekä muodostaa HTTP-protokollan mukaisia viestejä.

Toteutusosuudessa esitetään ohjelmistolle asetetut vaatimukset. Vaatimusten pohjalta esitetään eri vaihtoehtoja ohjelmistokirjastoista, joita voitaisiin käyttää ohjelmiston toteuttamiseksi sekä esitetään valitut ohjelmistokirjastot: Boost C++, libcurl sekä wxWidgets. Toteutusosuudessa esitetään myös ohjelmiston rakenne, joka tukee muuttujia sekä ulkoisia tietolähteitä, joiden avulla virtuaalikäyttäjille voidaan asettaa käyttäjäkohtaisia tietoja, kuten kirjautumistunnukset.

Toteutusosuuden lopussa esitetään käyttötapaus, josta voidaan nähdä, kuinka tehtyjen suorituskykymittausten pohjalta voidaan osoittaa testattavan kohdepalvelimen sekä palvelun yhtäaikaisten käyttäjien lukumäärän raja sekä syy.

Työssä selviää ohjelmiston graafisen käyttöliittymän esittämien kuvaajien merkitys suoritettaessa testejä. Vaikka lopulliset kuvaajat olisi tarkoitus esittää ulkoisella ohjelmalla, kuten Microsoft Excelillä, nopeuttavat ohjelmiston esittämät kuvaajat testien suoritusta, koska tulokset nähdään välittömästi ilman tietojen viemiseen liittyviä välivaiheita.

Työssä selviää myös palvelimella suoritettavien mittausten merkitys mahdollisten ongelmakohtien selvittämisessä. Testausohjelmistolla suoritettavilla mittauksilla voidaan selvittää ainoastaan palveluiden suorituskykyrajat, mutta yhdistämällä näihin tietoihin palvelimella suoritettujen mittausten tulokset on mahdollista selvittää myös palvelun suorituskykyä rajoittavat tekijät.

Avainsanat: suorituskykytestaus, HTTP, moniympäristöisyys, avoin lähdekoodi

Lahti University of Applied Sciences
Faculty of Technology

KEKKONEN, SAKU:

Performance testing software
for HTTP services

Batchelor's Thesis in Software Engineering, 52 pages, 9 appendices

Spring 2007

ABSTRACT

In this thesis, testing software is designed and implemented for performance testing HTTP services. The software is designed to support both Windows and Linux operating systems by using open source libraries. Main requirements for the software are support for HTTP and HTTPS protocols, simulating concurrent users, and presenting the results of testing.

In the theory part, interoperation between browser and server in basic use scenarios is discussed. HTML elements for loading web pages and sending information are investigated. From HTML, the focus is shifted to HTTP protocol used between browser and server. HTTP protocol is investigated, because it is needed for parsing and generating HTTP requests.

Requirements for software are represented in the empirical part of this thesis. Based on requirements, software libraries for implementing the software are studied, and selected ones: Boost++, libcurl and wxWidgets, are represented. The empirical part also presents the structure of the software for supporting variables and external data sources to enable per virtual user information, such as login.

In the end of the empirical part, a case example is described. It shows how performance testing of the target server and service can determine the maximum number of concurrent users and the reason for that limit.

In this thesis, the usefulness of internal charts of testing software is shown. Even when the final charts are meant to be presented with 3rd party application, such as Microsoft Excel, the internal charts of testing software can be used to execute tests in shorter time, because results can be seen immediately without intermediate steps needed for exporting results for 3rd party application.

The thesis also shows how statistics recorded at server side can help to pinpoint problem causers. It is possible to find out performance limits of a target system by investigating performance measurements recorded with testing software, but when measurements recorded at server side are combined with this information, it is also possible to find out the reasons behind the performance limits.

Key words: performance testing, HTTP, cross-platform, open source

SISÄLLYS

1 JOHDANTO 1

2 HTTP-PALVELUN TOIMINTA 3
2.1 Yleistä 3
2.2 WWW-sivun lataaminen 3
2.3 HTML 6
2.3.1 Yleistä 6
2.3.2 A-elementti 6
2.3.3 FORM-elementti 7
2.3.4 INPUT-elementti 7
2.4 Sivustolle kirjautuminen 8
2.5 Istunto ja evästeet 9
2.6 HTTP 12
2.6.1 HTTP-protokolla 12
2.6.2 HTTP-pyyntö 12
2.6.3 HTTP-vastaus 13
2.6.4 Viestiosa 14
2.6.5 Otsakekentät 15

3 TESTAUSOHJELMISTON TOTEUTUS 19
3.1 Tuotevaatimukset 19
3.2 Tekniset vaatimukset 20
3.2.1 Ympäristö 20
3.2.2 Ohjelmointikieli 20
3.2.3 HTTP-pyyntöjen käsittely 21
3.2.4 Boost C++ 21
3.2.5 Käyttöliittymä 22
3.2.6 Kääntöympäristö 22
3.2.7 Testisekvenssin tallennus 23
3.2.8 Testisekvenssin tulkitseminen 25
3.2.9 Testisekvenssin sisäinen rakenne 28
3.2.10 Testisekvenssin syötteiden automatisointi 30
3.2.11 Testisekvenssin ajo 36
3.2.12 Testitulokset 38

3.2.13 Graafinen käyttöliittymä 39
3.3 Käytännön testitapaus 45

4 YHTEENVETO 49

5 LÄHTEET 51

6 LIITTEET 53

1 JOHDANTO

Suunniteltaessa internetin välityksellä käytettäviä sovelluksia on otettava huomioon se, että sovelluksella voi olla satoja tai jopa tuhansia yhtäaikaisia käyttäjiä. Tämä yhtäaikaisten käyttäjien aikaansaama kuormitus voi rasittaa järjestelmää aiheuttaen odottamattomia taukoja, jotka johtavat huonoon käyttäjäkokemukseen. Mikäli palvelu on riittävän hyvin testattu ja säädetty optimaaliseen suorituskykyyn rasitustilanteissa, voidaan olla varmoja siitä, että palvelu toimii riittävällä tasolla. (Microsoft ACE Team 2002.)

Työn tilaajana toimii Flander Oy. Flander Oy on mobiili- ja telecom -toimialalla toimiva ohjelmistotalo, joka keskittyy langattomien järjestelmien ohjelmistokehitykseen sekä testaukseen. Flanderilla työskentelee yli 200 alan ammattilaista. (Flander 2006.)

Tässä opinnäytetyössä suunnitellaan ja toteutetaan ohjelmisto, jolla voidaan suorittaa suorituskykytestausta HTTP-palvelimille. Ohjelmiston pääasiallisena käyttötarkoituksena on alustavan suorituskykytestauksen ajaminen testattavina oleville palveluille, jotta saadaan kuva siitä, riittääkö palvelun suorituskyky suunnitellulle käyttäjämäärälle.

Suunniteltavan suorituskykytestausohjelmiston asiakasvaatimuksina ovat HTTP- ja HTTPS-protokollien tuki, selaimen käyttöä emuloivien virtuaalikäyttäjien tuki, virtuaalikäyttäjäkohtaisten tietojen, kuten kirjautumistunnusten tuki sekä mittaustulosten tuottaminen. Mittauksia ei tarvitse suorittaa palvelimelta. Mittaustuloksissa riittävät HTTP-viestien vasteajat sekä palvelimelta tulevien HTTP-vastausten tilakoodit.

Tutkimusosassa tutustutaan HTTP-protokollaan, HTML-kielen elementteihin, jotka liittyvät asiakkaan ja palvelimen väliseen toimintaan sekä tietoliikenteeseen asiakkaan ja HTTP-palvelimen välillä käytettäessä HTTP-palveluita selaimella. Kappaleessa 3 johdetaan teknisten vaatimusten ja tutkimusosan tietojen pohjalta sovelluksen tuotevaatimukset sekä esitellään ohjelmiston suunnittelussa tehtyjä valintoja ja kohdattuja ongelmia. Kappaleessa 4 pohditaan tavoitteiden toteutumista sekä mahdollisia jatkokehitysideoita.

2 HTTP-PALVELUN TOIMINTA

2.1 Yleistä

Seuraavissa kappaleissa tutkitaan mitä käytännössä tapahtuu, kun WWW-selaimella käytetään HTTP-palvelimen palveluita. Käyttötapausten ja niihin liittyvien protokollien tunteminen on olennaista, koska sovelluksen tarkoituksena on suorittaa automaattisesti HTTP-pyyntöjä, joilla simuloidaan useampien yhtäaikaisten WWW-selainten aiheuttamaa kuormaa palvelimelle. Kappaleen sisältö auttaa myös sovelluksen loppukäyttäjiä testauksen suunnittelussa.

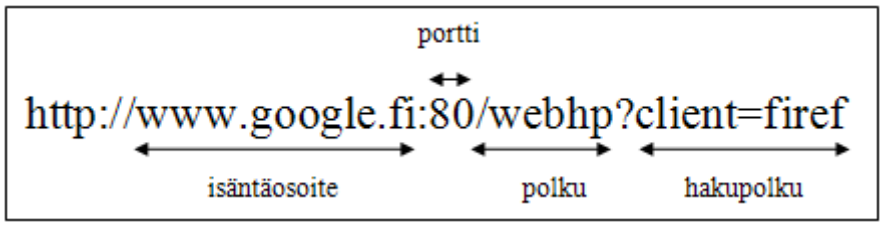
2.2 WWW-sivun lataaminen

Kun WWW-selaimen osoitekenttään (KUVIO 1) syötetään osoite ja pyydetään selainta hakemaan osoitekenttään syötetty sivu, lähettää selain palvelimelle pyynnön. Ennen pyynnön lähettämistä selaimen täytyy selvittää annetusta osoitteesta palvelun tyyppi sekä osoite.

KUVIO 1. Selaimen osoitekenttä

Selaimen osoitekenttään syötettävää osoitetta kutsutaan URL:ksi. URL on lyhenne, joka tulee sanoista Universal Resource Locator (Berners-Lee, Masinter & McCahil 1994).

URL jakaantuu kaksoispisteellä eroteltuna kahteen osaan: skeema, sekä skeema-riippuvainen osa. KUVIO 1:n esimerkkiosoitteessa skeema on HTTP ja skeema-riippuvainen osa //www.google.fi. Yleisimmät selaimella käytettävät skeemat ovat HTTP ja HTTPS, joiden skeema on KUVIO 2:ssa esitettyä muotoa.



KUVIO 2. HTTP-skeema

Mikäli annetun osoitteen URL:ssa ei ole määritelty porttia, käytetään HTTP-skeemalla vakioarvoa 80 ja HTTPS-skeemalla arvoa 443. Myös polku ja hakupolku ovat valinnaisia. Mikäli näitä ei anneta, voidaan myös polkua edeltävä kauttaviiva jättää pois. (Berners-Lee ym. 1994, 1, 8.)

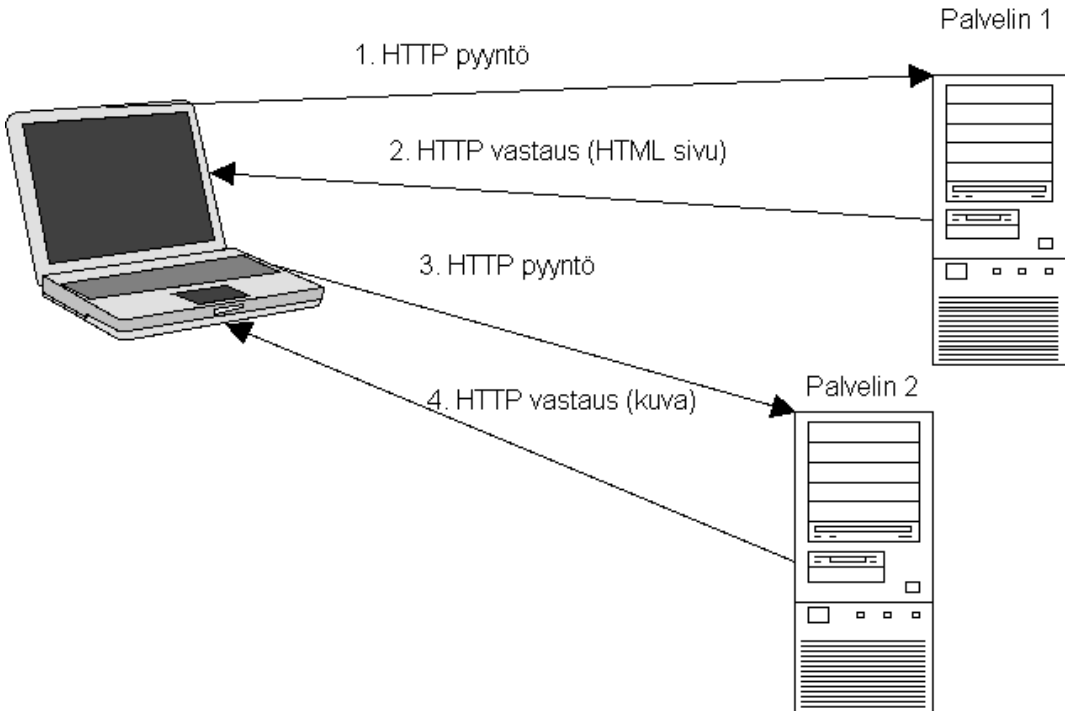
Isäntäosoite voi olla joko IPv4, IPv6 tai rekisteröidyn nimen muodossa. IPv4-muotoinen osoite muodostuu neljästä pisteellä erotetusta kokonaisluvusta välillä 0-255, esimerkiksi 127.0.0.1. IPv6-osoite muodostuu hakasulkeiden sisällä olevista heksadesimaalinumeroista, esim. [a0:01:19:ab:ba:00:5a:7a]. Mikäli annettu osoite ole kummassakaan IP-muodossa, tulkitaan sen olevan rekisteröity nimi. (Berners-Lee ym. 1994, 17–20.)

Mikäli annettu osoite on rekisteröity nimi, täytyy selaimen selvittää rekisteröidyn nimen IP-osoite. Selain antaa nimen selvitettäväksi käyttöjärjestelmälle, joka voi palauttaa IP-osoitteen välimuististaan, tai mikäli nimeä ei löydy, lähettää kyselyn yhdelle tai useammalle nimipalvelimelle (Domain name system – Wikipedia 2007). IP-osoitteen selvittämiseen kuluva aika vaihtelee sen mukaan, mitä kautta osoite saadaan selvitettyä.

Kun käytettävä skeema, IP-osoite sekä portti ovat tiedossa, lähettää WWW-selain palvelimelle pyynnön. HTTP-skeemaa käytettäessä lähetetään HTTP-pyyntö.

HTTP-palvelin vastaa HTTP-pyyntöön HTTP-vastauksella. Mikäli HTTP-pyynnössä pyydetty resurssi löytyi, palvelimen lähettämässä vastauksessa on osana kyseisen resurssin sisältö. WWW-sivun tapauksessa sisältönä on HTML-sivu.

WWW-selain parsii vastaanottamansa HTML-sivun ja näyttää sisällön selaimen ruudulla. Mikäli sivu sisältää kuvia tai muuta sivuun kuuluvaa sisältöä, jotka eivät ole osana HTML sivua, lähettää WWW-selain erilliset pyynnöt kyseisistä resurs-seista. KUVIO 3 kuvaa tilannetta, jossa palvelimelta ladataan sivu, johon kuuluu kuva. Kuva sijaitsee eri palvelimella kuin itse sivu.



KUVIO 3. HTML sivun lataus

2.3 HTML

2.3.1 Yleistä

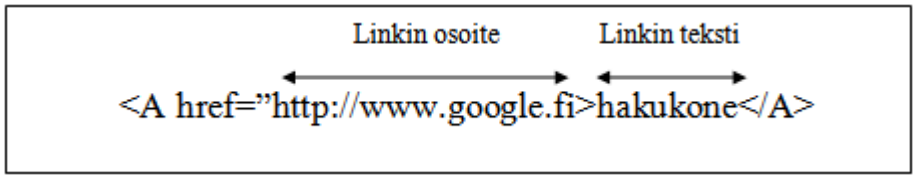
Seuraavassa kappaleissa käsitellään HTML elementtejä, jotka liittyvät sisällön lataukseen ja sitä kautta suorituskykytestaukseen.

2.3.2 A-elementti

A-elementtiä käytetään HTML-sivuilla linkkien muodostamiseen. A-elementin href-attribuutilla voidaan määritellä Web-resurssin URI. Tällä määritellään linkki elementin (lähde ankkuri) ja kohdeankkurin välille. (Raggett, Hors & Jacobs 1998, 138-139.)

URI, eli Univeral Resource Identifier, on resurssien määrittelyyn käytettävä tyyppi, jonka yhtenä alatyyppinä on URL. URL määrittelee resurssin lisäksi yhteystavan. (Bernets-Lee, Fielding & Masinter 2005, 6.)

KUVIO 4 esittää A-elementtiä, jonka sisältönä, eli selaimessa näkyvänä tekstinä, on teksti ”hakukone”. Kun sivulta valitaan kyseinen linkki, lähettää selain GET tyyppisen HTTP-pyynnön href-kentän URI:n määrittelemälle www.google.fi palvelimelle.



KUVIO 4. A-elementti

2.3.3 FORM-elementti

FORM-elementtiä käytetään kaavakkeiden luomiseen. Action-attribuutilla määritellään URI, johon kaavakkeen tiedot lähetetään. Method-attribuutilla määritellään HTTP-pyynnön metodi, joka on joko GET tai POST. Enctype-attribuutilla määritellään lähetettävän sisällön tyyppi POST-metodia käytettäessä. (Raggett ym. 1998, 210.) Kun FORM-elementin tiedot lähetetään, ne välittyvät valinnasta riippuen joko GET- tai POST-metodilla.

GET-metodissa pyyntö ei sisällä viestin sisältöosaa. Kun palvelimelle välitetään tietoa, se välitetään URI:n hakupolkuosassa. Välitettävän datan sisällössä sallitaan tällöin ainoastaan ASCII-merkkejä. (Raggett ym. 1998, 232.)

POST-metodilla tiedot välittyvät asiakkaalta (selain) palvelimelle HTTP-viestin sisällössä. POST-metodin etuna GET-metodiin verrattuna on se, että lähetettävä data voi sisältää muitakin arvoja kuin mitä ASCII-merkeillä voidaan esittää sekä se, ettei lähetettävää dataa näytetä selaimen osoitepalkissa.

2.3.4 INPUT-elementti

INPUT-elementtejä käytetään kaavakkeissa tietojen syöttämiseen. Input-elementin Name-attribuutilla kerrotaan kentän nimi. Value-attribuutilla kentälle annetaan oletusarvo. Oletusarvo on vaihtoehtoinen, mikäli kyseessä ei ole radio- tai checkbox tyyppinen kenttä (Raggett ym. 1998, 212). Type-attribuuttia käytetään input-kentän tyypin määrittelyssä. TAULUKKO 1 listaa HTTP-pyynnön sisältöön ja lähetykseen liittyvät input-kentän tyypit.

TAULUKKO 1. Input-kenttien tyypit

Kentän tyyppi	**Selite**
text	Yksirivinen tekstikenttä.
password	Yksirivinen tekstikenttä, jonka sisältöä ei näytetä.
hidden	Kenttä, jonka sisältö välitetään HTTP-pyynnössä, mutta jonka sisältöä ei näytetä selaimen ruudulla.
file	Tiedoston valinta.
submit	Kaavakkeen lähetyspainike.

2.4 Sivustolle kirjautuminen

Jotta sivuston käyttöä voitaisiin simuloida, täytyy automaattisesti suoritettavien HTTP-pyyntöjen toimia kuten normaalissa käytössäkin. Palveluissa, joissa kirjaudutaan, kirjautuminen suoritetaan kerran, ja tämän jälkeen palveluun lähetetään muita pyyntöjä. Myös simuloidussa käytössä täytyy palveluun kirjautua, joten sivustolle kirjautuminen pitää pystyä tunnistamaan ja muuttamaan kirjautumista koskevat tiedot automaattisiksi syötteiksi.

Sivustoille kirjautuminen suoritetaan yleisimmin käyttäen KUVIO 5:n tyyppistä kaavaketta. Kaavakkeessa on kaksi kenttää: nimi sekä salasana. Lisäksi kaavakkeessa on painike, jolla kirjautuminen suoritetaan.

KUVIO 5. Kirjautumiskaavake

Kaavaketta vastaava HTML-koodi on esitetty TAULUKKO 2:ssa. Koodissa on määritelty kolme input-kenttää: username, password sekä submit (rivit 5-7), joiden arvot lähetetään post-metodilla login.php sivulle (rivi 4), kun käyttäjä painaa Login-painiketta (rivi 7). Kun kirjautumistiedot on lähetetty, palvelin tarkistaa kyseiset tiedot, ja mikäli nämä täsmäävät, luo se käyttäjälle istunnon.

TAULUKKO 2. Kirjautusmiskaavakkeen HTML-koodi

Rivi	**HTML sisältö**
1	<HTML>
2	<HEAD><TITLE>login</TITLE></HEAD>
3	<BODY>
4	<FORM action="http://muuli.kekkonen.org" method="post">
5	name: <INPUT type="text" name="username">
6	password: <INPUT type="password" name="password">
7	<INPUT type="submit" name="submit" value="Login">
8	</FORM>
9	</BODY>
10	</HTML>

2.5 Istunto ja evästeet

HTTP-palvelimet vastaavat jokaiseen pyyntöön ilman, että se liittyisi mitenkään edellä tehtyihin pyyntöihin. HTTP-pyynnöt ja vastaukset voidaan liittää evästeitä käyttäen osaksi suurempaa kokonaisuutta, jota kutsutaan istunnoksi (Kristol & Montulli 1997, 2).

Evästeitä hallitaan kahdella HTTP-otsakekentällä: Set-Cookie, sekä Cookie. Palvelin voi aloittaa istunnon lähettämällä HTTP-vastauksessa Set-Cookie otsakekentän. Selain lähettää seuraavassa HTTP-kyselyssä kyseisen evästeen takaisin Cookie-otsakekentässä. Istuntoa pidetään yllä jatkamalla evästeiden välitystä Set-Cookie- ja Cookie-otsakekentillä. (Kristol & Montulli 1997, 2-3.)

HTTP-palveluissa, joissa käyttäjä kirjautuu sivustolle, palvelin luo yleensä käyttäjälle istunnon ajaksi tunnisteen, joka lähetetään Set-Cookie-otsakekentässä käyttäjän selaimelle. Kun selain tekee uuden pyynnön samalla sivustolle, lähetetään HTTP-pyynnön mukana tämä sama tunniste Cookie-otsakekentässä takaisin palvelimelle. Palvelin tarkistaa tunnisteen perusteella, mitä käyttäjää saapunut HTTP-pyyntö koskee.

TAULUKKO 3 sisältää esimerkki HTTP-pyynnön (POST /login.php), jossa lähetetään kirjautumistiedot (username=exampleuser, password=topsecretpassword) palvelimelle. Pyyntöä seuraa HTTP-vastaus (HTTP/1.x 302 Found), jossa palvelin ilmoittaa istunnon tunnisteen (ampache=cf4c59d9a4c304fe5ae8f27fa51803cd). Selain lähettää seuraavassa pyynnössä (GET /index.php) tunnisteen palvelimelle (Cookie: ampache=cf4c59d9a4c304fe5ae8f27fa51803cd), jolloin palvelin tunnistaa pyynnön tulleen samalta käyttäjältä.

TAULUKKO 3. Istunnonluonti esimerkki

POST /login.php HTTP/1.1 Host: example.domain.org User-Agent: Mozilla/5.0 Accept: text/xmlAccept-Language: fi,en;q=0.5 Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 Referer: http://example.domain.org/login.php Content-Type: multipart/form-data; boundary=---------------------------- 265001916915724 Content-Length: 417 ----------------------------265001916915724 Content-Disposition: form-data; name="username" exampleuser ----------------------------265001916915724 Content-Disposition: form-data; name="password" topsecretpassword ----------------------------265001916915724--
HTTP/1.x 302 Found Date: Mon, 12 Feb 2007 19:29:03 GMT Server: Apache X-Powered-By: PHP/5.1.6-pl6-gentoo Set-Cookie: ampache=cf4c59d9a4c304fe5ae8f27fa51803cd; path=/ampache Expires: Thu, 19 Nov 1981 08:52:00 GMT Location: http://example.domain.org/ampache/index.php Content-Length: 0 Content-Type: text/html; charset=utf-8
GET /index.php HTTP/1.1 Host: example.domain.org User-Agent: Mozilla/5.0 Accept: text/xml Accept-Language: fi,en;q=0.5 Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 Referer: http://example.org/login.php Cookie: ampache=cf4c59d9a4c304fe5ae8f27fa51803cd

2.6 HTTP

2.6.1 HTTP-protokolla

HTTP, eli Hypertext Transfer Protocol, on ohjelmistotason tilaton, pyyntö-vastaus tyyppinen protokolla. Pyyntö-vastaus protokollassa asiakas lähettää palvelimelle pyynnön, johon palvelin vastaa. Koska HTTP on tilaton, palvelin ei osaa yhdistää asiakkailta tulevia erillisiä pyyntöjä toisiinsa. HTTP-palveluissa on kuitenkin usein tarve pitää asiakkaan ja palvelimen välistä tilaa yllä, jotta esimerkiksi HTML-sivuja käytettäessä voitaisiin kirjautua. Tätä mekanismia kutsutaan istunnoksi.

HTTP-protokolla ei ota kantaa käytettävään siirtotiehen, mutta yleisimmin HTTP-protokollaa käytetään TCP-protokollan päällä. HTML-selaimet käyttävät TCP-protokollaa ja ottavat yhteyden palvelimen TCP-porttiin 80, mikäli porttia ei erikseen määritellä.

2.6.2 HTTP-pyyntö

HTTP-transaktio alkaa asiakkaan lähettämällä pyynnöllä. Lähetettävä pyyntö (TAULUKKO 4) sisältää pyynnön metodin, URI:n, protokollaversion, sekä MIME-tyyppisen viestin, jossa on otsakekenttiä sekä mahdollinen sisältö (eng. body content). (Fielding, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999, 10.)

TAULUKKO 4. HTTP-pyyntö

Pyynnön rivi	**Selite**
GET /tl HTTP/1.1	Pyynnön metodi, URI & protokollaversio.
Host: www.lamk.fi	1. otsakekenttä
Referer: http://www.lamk.fi	2. otsakekenttä

2.6.3 HTTP-vastaus

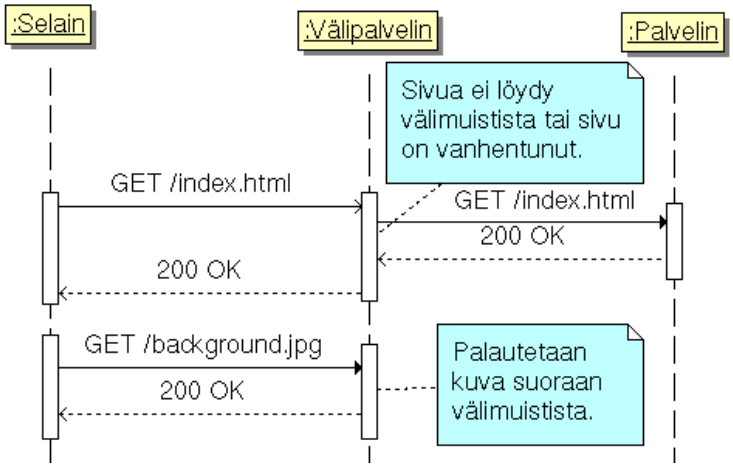
Palvelin vastaa pyyntöön viestillä (TAULUKKO 5), joka sisältää tilarivin jossa olevasta koodista ilmenee pyynnön onnistuminen tai epäonnistuminen. Tilariviä seuraa MIME-tyyppinen viesti, joka sisältää tietoa palvelimesta, sekä mahdollisen viestin sisällön. (Fielding ym. 1999, 10.)

TAULUKKO 5. HTTP-vastaus

Vastauksen rivi	Selite
HTTP/1.x 200 OK	Tilarivi
Date: Thu, 14 Dec 2006 18:18:29 GMT	1. otsakekenttä
Content-Type: text/html	2. otsakekenttä
Content-Encoding: gzip	3. otsakekenttä
Content-Length: 2171	4. otsakekenttä
	Tyhjä rivi. Otsakekenttien ja viestin erotin.
<HTML>	Sisällön alku

Asiakkaan ja palvelimen välissä saattaa olla myös välipalvelin, jonka kautta asiakkaan ja palvelimen välinen viestiliikenne kulkee. Välipalvelin voi tallentaa välittämiään tietoja välimuistiin ja palauttaa asiakkaan pyytämän resurssin suoraan välimuistista ilman, että sitä kysytään kohde palvelimelta (Fielding ym. 1999, 11–12).

KUVIO 6:n sekvenssikaavio esittää tilanteen, jossa index.html sivu haetaan kohdepalvelimelta, mutta jälkimmäisen pyynnön kuvatiedosto palautetaan suoraan välipalvelimelta.



KUVIO 6. Välipalvelin

2.6.4 Viestiosa

Viestiosa koostuu otsakekentistä (header fields) sekä mahdollisesta sisällöstä (body). Viesti erotetaan otsakekentistä näiden välissä olevalla tyhjällä rivillä, eli rivillä, jolla on ainoastaan rivinvaihtomerkit CRLF (carriage return & linefeed). Jokainen otsakekenttä on omalla rivillään. Mikäli uusi tekstirivi alkaa välilyönnillä tai vaakatabulaattorilla, lasketaan tämän rivin kuuluvan edellisen rivin otsakekentän tietoihin (Crocker 1982, 10, 15).

2.6.5 Otsakekentät

Rakenne

Otsakekenttä jakautuu kentän nimeen sekä kentän sisältöön. Nämä osat erotetaan toisistaan kaksoispisteellä. Otsakekenttä päättyy CRLF-rivinvaihtoon. TAULUKKO 6 esittää otsakekentän jaettuna osiin. Kentän nimi voi sisältää ainoastaan US-ASCII-merkkejä, joiden arvo on väliltä 33 - 126, pois lukien kaksoispiste. Otsakekentän nimissä ei erotella isoja ja pieniä kirjaimia, joten esimerkiksi seuraavat kentännimet ovat yhteneviä: Content-Type, content-type, CONTENT-typE. Kenttä voi sisältää mitä tahansa US-ASCII-merkkejä paitsi CR ja LF. (Crocker 1982, 5, 14.)

TAULUKKO 6. Otsakekentän rakenne

nimiosa	**Osien erotin**	**sisältöosa**	**Otsakekenttien erotin**
Content-Type	:	text/html	CRLF

Content-Length

Content-Length-kenttä kertoo viestin sisällön tarkan pituuden oktetteina, eli 8-bittisinä datasekvensseinä. HTTP-pyynnöissä content-length-kentän olemassa olo ilmoittaa palvelimelle, että pyyntö sisältää viestin sisältöosan. HTTP-vastauksissa content-length-kentän olemassaolo riippuu pyynnön tyypistä sekä vastauksen tilakoodista. (Fielding ym. 1999, 15, 32.)

Content-Type

Content-Type-kenttä kertoo viestin sisällön tyypin. Jokaisen HTTP/1.1 pyynnön, joka sisältää viestin sisältöosan, tulisi sisältää Content-Type-kenttä. Mikäli content-type-kenttää ei ole, voidaan sisällön tyyppi arvioida sisällön perusteella. Mikäli viestin tyyppiä ei tiedetä, se tulkitaan sisällöltään application/octet-stream tyyppiseksi. (Fielding ym. 1999, 42.)

Application/x-www-form-urlencoded

x-www-form-urlencoded on sisältötyyppi viestille, jolla lähetetään kaavakkeen kenttien tietoja POST-metodilla palvelimelle. TAULUKKO 7 sisältää esimerkin x-www-form-urlencoded sisältötyyppisen HTTP-viestistä. Esimerkkiviestissä ilmenee sisältötyypin lisäksi sisällön pituus, 32 oktettia, sekä itse sisältö URL-koodattuna. URL-koodauksessa merkit, joiden ASCII-heksadesimaaliarvo on väliltä 00-1F, tai 7F-FF, esitetään prosenttimerkillä, jota seuraa merkin ASCII-arvo heksadesimaalimuodossa (Berners-Lee ym. 1994, 1-3). Esimerkin sisältöosa olisi URL-koodauksen poistamisen jälkeen ”username=testi&password=särö”.

TAULUKKO 7. url-encoded esimerkki

Viestin sisältö	**Selite**
POST /login.php HTTP/1.1	Pyynnön tyyppi.
Content-Type: application/x-www-form-urlencoded	Sisältötyyppi.
Content-Length: 32	Sisällön pituus.
	Tyhjä rivi otsakekenttien ja sisällön erottimena.
username=testi&password=s%84r%94	Kaavakkeen kentät.

Multipart

Multipart on sisältötyyppi, joka koostuu yhdestä tai useammasta osasta. Eri osat erotellaan toisistaan Content-Type-otsakkeen boundary-parametrin osoittamalla erottimella. Varsinainen erotin muodostetaan lisäämällä kahden väliviivamerkin perään boundary-parametrin osoittama merkkijono. Viimeistä osaa seuraa lopetuserotin. Lopetuserotin muodostetaan lisäämällä boundary-parametrin osoittaman erottimen alkuun ja loppuun kaksi väliviivaa. (Freed & Borenstein 1996, 17.)

TAULUKKO 8. Erottimen muodostus

Otsake	Content-Type: multipart/mixed; boundary=a3b1cde3Es1M
Erotin	--a3b1cde3Es1M
Lopetuserotin	--a3b1cde3Es1M--

Erottimien täytyy alkaa suoraan rivin alusta eli CRLF-merkkiparin jälkeen. CRLF-merkkiparin lasketaan kuuluvan osaksi erotinta, ei tätä edeltävää osaa. Erotinosaa seuraa valinnainen määrä valkomerkkejä (whitespaces), joiden perään tulee toinen CRLF-merkkipari. Erottimia seuraa Otsakekentät sekä sisältöosa eroteltuina toisistaan tyhjällä rivillä. Sisältöosa on samantapainen kuin kappaleessa 3.3 esitelty HTTP-viesti, sillä erotuksella, että otsakekentät ovat valinnaisia. Mikäli otsakekenttiä ei ole, seuraa viestin sisältö erotinosaa tyhjällä rivillä erotettuna. (Freed & Borenstein 1996, 17, 19.)

TAULUKKO 9 esittää multipart-viestin rakennetta esimerkki sisällöllä. Viestistä esitetään käytettävä erotin (abc123), sekä kaksi osaa eroteltuna kyseisellä erottimella. KUVIO 7 esittää oikean HTTP-pyynnön, jossa käytetään huomattavasti pidempää erotinta (----------265001916915724), jotta todennäköisyys sille, että itse sisällöstä löytyisi sama merkkijono, olisi mahdollisimman pieni. Viestissä esitettävään sisällön pituuteen (475) on laskettu mukaan kaikki merkit Content-Length-rivin jälkeen, sisältäen myös erottimet.

TAULUKKO 9. Multipart-viesti

Viestin sisältö	**Selite**
Content-Type: multipart/mixed; boundary=abc123	Otsakekenttä
	Otsake-sisältö erotin
--abc123	Erotin
Content-Type: text/html	Otsakekenttä
	Otsake-sisältö erotin
<HTML></HTML>	Sisältö
--abc123	Erotin
	Otsake-sisältö erotin
<PRE></PRE>	Sisältö
--abc123--	Lopetuserotin

```
POST /ampache/search.php HTTP/1.1
Host: subdomain.somedomain.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fi; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: fi,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://somedomain.org/ampache/index.php
Cookie: ampache=b81bd36058560dbebd69f5186471d177
Content-Type: multipart/form-data; boundary=---------------------------265001916915724
Content-Length: 475
-----------------------------265001916915724
Content-Disposition: form-data; name="search_string"

mokoma
-----------------------------265001916915724
Content-Disposition: form-data; name="action"

quick_search
-----------------------------265001916915724
Content-Disposition: form-data; name="method"

fuzzy
-----------------------------265001916915724
Content-Disposition: form-data; name="object_type"

song
-----------------------------265001916915724--
```

KUVIO 7. Esimerkki multipart-viesti

3 TESTAUSOHJELMISTON TOTEUTUS

3.1 Tuotevaatimukset

Ohjelmiston asiakasvaatimuksina olivat HTTP- ja HTTPS-protokollien tuki, selaimen käyttöä emuloivien virtuaalikäyttäjien tuki, virtuaalikäyttäjäkohtaisten tietojen, kuten kirjautumistunnusten tuki, sekä mittaustulosten tuottaminen. Näiden vaatimusten lisäksi ohjelmiston piti kyetä samoihin toimintoihin kuin referenssiapplikaationa käytetty Webserver Stress Tool 7, jonka kuvaruutukaappauksia löytyy liitteestä 1.

Edellä mainittujen asiakasvaatimusten pohjalta luotiin seuraavat toiminnalliset vaatimukset:

TAULUKKO 10. Toiminnalliset vaatimukset

Vaatimus	**Selite**
HTTP tuki	Ohjelman on tuettava HTTP-protokollan mukaisia viestejä.
HTTPS tuki	Ohjelman on tuettava HTTPS-protokollan mukaisia viestejä.
Evästeet	Ohjelman on kyettävä lähettämään ja vastaanottamaan evästetietoja.
User-agent	Ohjelmassa pitää voida määrittää palvelimelle lähetettävä user-agent tieto.
Vapaavalintaiset otsakekentät	Ohjelman pitää kyetä lähettämään HTTP-pyynnöissä haluttuja otsakekenttiä.
Timeout asetus	Ohjelman pitää tukea timeout-arvojen asettamista HTTP-pyynnöille.
Ethernet-rajapinnan asetus	Ohjelman pitää tukea käytettävän ethernet-rajapinnan valintaa.
HTTP-sekvenssin tallennus	Ohjelman pitää kyetä nauhoittamaan selaimen lähettämät HTTP-pyynnöt toistoa varten.
HTTP-sekvenssin ajo	Ohjelman pitää kyetä toistamaan nauhoitettuja HTTP-pyyntöjä.

(jatkuu)

TAULUKKO 10. (jatkuu)

HTTP-pyyntöjen välinen tauko	Ohjelmassa pitää voida säätää pyyntöjen välissä pidettävä tauko.
HTTP-pyyntöjen virtuaalikäyttäjäkohtaiset tiedot	Ohjelman pitää kyetä asettamaan lähettäviin HTTP-pyyntöihin virtuaalikäyttäjäkohtaisia tietoja, kuten kirjautumistunnus ja salasana.
Yhtäaikaisten HTTP-kyselyiden tuki	Ohjelman pitää kyetä lähettämään useita HTTP-pyyntöjä samanaikaisesti, jotta voidaan simuloida yhtäaikaisia palvelimelle pyyntöjä lähettäviä selaimia.
Yhtäaikaisten virtuaalikäyttäjien määrän valinta	Ohjelmassa pitää voida valita yhtäaikaisten virtuaalikäyttäjien lukumäärä.
Vasteaikojen mittaus	Ohjelman on kyettävä mittaamaan HTTP-kyselyihin kuluneet ajat.
Mittaustulosten tallennus	Ohjelman on kyettävä tallentamaan mittaustulokset.
Mittaustulosten esitys	Ohjelman on kyettävä esittämään mittaustulokset.
Windows tuki	Ohjelman on toimittava Windows XP käyttöjärjestelmässä.

3.2 Tekniset vaatimukset

3.2.1 Ympäristö

Ohjelmiston asiakasvaatimukset eivät olleet kovin tarkkoja, mikä johtui asiakkaan vähäisestä käytännön kokemusta HTTP-palveluiden suorituskykytestauksessa. Tästä johtuen ohjelmisto suunniteltiin mahdollisia kehitysideoita silmällä pitäen siten, että ei rajoitettaisi ohjelmiston laajennettavuutta ja käyttökohteita. Vaaditun Windows XP tuen lisäksi ohjelmistossa päätettiin tukea myös Linux ympäristöä, mikäli tästä ei aiheudu suurta lisätyötä.

3.2.2 Ohjelmointikieli

Ohjelmiston toteutuksessa mietittiin vaihtoehtoisesti C/C++ kielen käyttämistä tai PHP:tä. PHP:lle löytyivät tarvittavat kirjastot HTTP-pyyntöjen suorittamiseen, mutta säikeiden tuen puutteen takia vaihtoehto hylättiin.

3.2.3 HTTP-pyyntöjen käsittely

HTTP-pyyntöjen käsittelyn toteuttamiseksi tutkittiin kolmea eri toteutustapaa: avoimen lähdekoodin kirjastot, avoimen lähdekoodin komentorivisovellukset, sekä viimeisenä vaihtoehtona oma toteutus. Omaa toteutusta haluttiin välttää suuren työmäärän takia. Valmiista Windows- ja Linux-ympäristössä toimivista komentorivisovelluksista tutkittiin curl- ja wget-sovelluksia. Kyseisistä sovelluksista löytyy myös ohjelmistokirjastot. Näiden ohjelmistokirjastojen lisäksi tutkittiin vaihtoehtona myös Mozilla Firefox-selaimen käyttämää Neckoa, mutta tämä osoittautui liian raskaaksi johtuen riippuvuuksista muihin Firefoxin osiin.

Näistä vaihtoehdoista valittiin curl-kirjasto. Curl-kirjaston etuina oli suoritettujen HTTP-pyyntöjen eri osa-alueiden suoritusaikojen mittaus, sekä se ettei Curl tarvinnut muita kirjastoja toimiakseen. Kirjaston käyttö osoittautui hieman oletettua hankalammaksi, koska kirjasto on suunniteltu niin, että se luo tarvittavat kentät HTTP-pyyntöön annettujen tietojen pohjalta. Testaussovellus sisälsi kuitenkin HTTP-pyynnöt jo kokonaisuudessaan, joten tietojen antaminen curl:lle niin, että ne myös lähetettäisiin sellaisenaan palvelimelle, vaati odotettua enemmän työtä.

3.2.4 Boost C++

Koska sovelluksen haluttiin toimivan sekä Windows- että Linux-ympäristöissä, tarvittiin mm. säikeiden käsittelyyn molemmissa käyttöjärjestelmissä toimiva kirjasto. Vaihtoehtona olisi ollut myös oma toteutus, mutta koska tähän käyttöön varten löytyi sopiva kirjasto, jota voidaan käyttää säikeiden käsittelyn lisäksi monissa muissakin sovelluksen osissa, päädyttiin käyttämään Boost C++-kirjastoja. Boost on lukuisilla eri alustoilla toimiva avoimen lähdekoodin kirjasto, jota voidaan käyttää myös suljetun lähdekoodin kaupallisissa sovelluksissa. Boost sisältää mm. kirjastot säikeiden käsittelyyn, komentoriviparametrien käsittelyyn, sekä säännöllisten lausekkeiden käsittelyyn. Lisäksi Boost sisältää luokat kellonajan käsittelyyn sekunnin osien tarkkuudella.

3.2.5 Käyttöliittymä

Asiakasvaatimuksissa ei otettu kantaa käyttöliittymään. Ohjelma päätettiin toteuttaa komentorivisovelluksena, jonka tuottamat mittaustulokset voitaisiin viedä Exceliin. Ohjelmasta toteutettiin komentoriviversio, joka oli kehitysvaiheessa toiminnallisuuden testaamisessa nopeampikäyttöinen kuin mitä graafisella käyttöliittymällä tehty olisi ollut. Mittaustulosten vieminen taulukkolaskentaohjelmaan oli kuitenkin käytännössä hidas vaihe, jonka koettiin hankaloittavan käyttöä, joten ohjelmasta päätettiin toteuttaa myös versio graafisella käyttöliittymällä. Komentoriviversion lähdekoodi sisältää vain parametrin lukuun ja tulostukseen liittyvät toiminnot, joten jatkossa komentoriviversion ylläpito graafisen version ohessa ei lisää suuresti työmäärää. Lisäksi komentoriviversiota voidaan ajaa helposti etänä palvelimella ja mahdollisesti jatkokehittää palvelinsovellukseksi.

Windowsilla sekä Linuxilla toimivan graafisen käyttöliittymän kehittämiseen löytyy lukuisia kirjastoja, mm: QT, GTK+, FLTK sekä wxWidgets. QT vaihtoehto hylättiin, koska sen lisenssi ei salli kaupallisen sovelluksen tekemistä ilmaisella lisenssillä. Lopullinen valinta kohdistui wxWidgetsiin, koska sillä tehdyt sovellukset käyttävät käyttöjärjestelmän omia komponentteja, joten esimerkiksi Windowsille käännetty versio näyttää ja käyttäytyi kuin muutkin Windows-ohjelmat. Linux versiossa vaihtoehtoina ovat GTK ja X Window System.

3.2.6 Kääntöympäristö

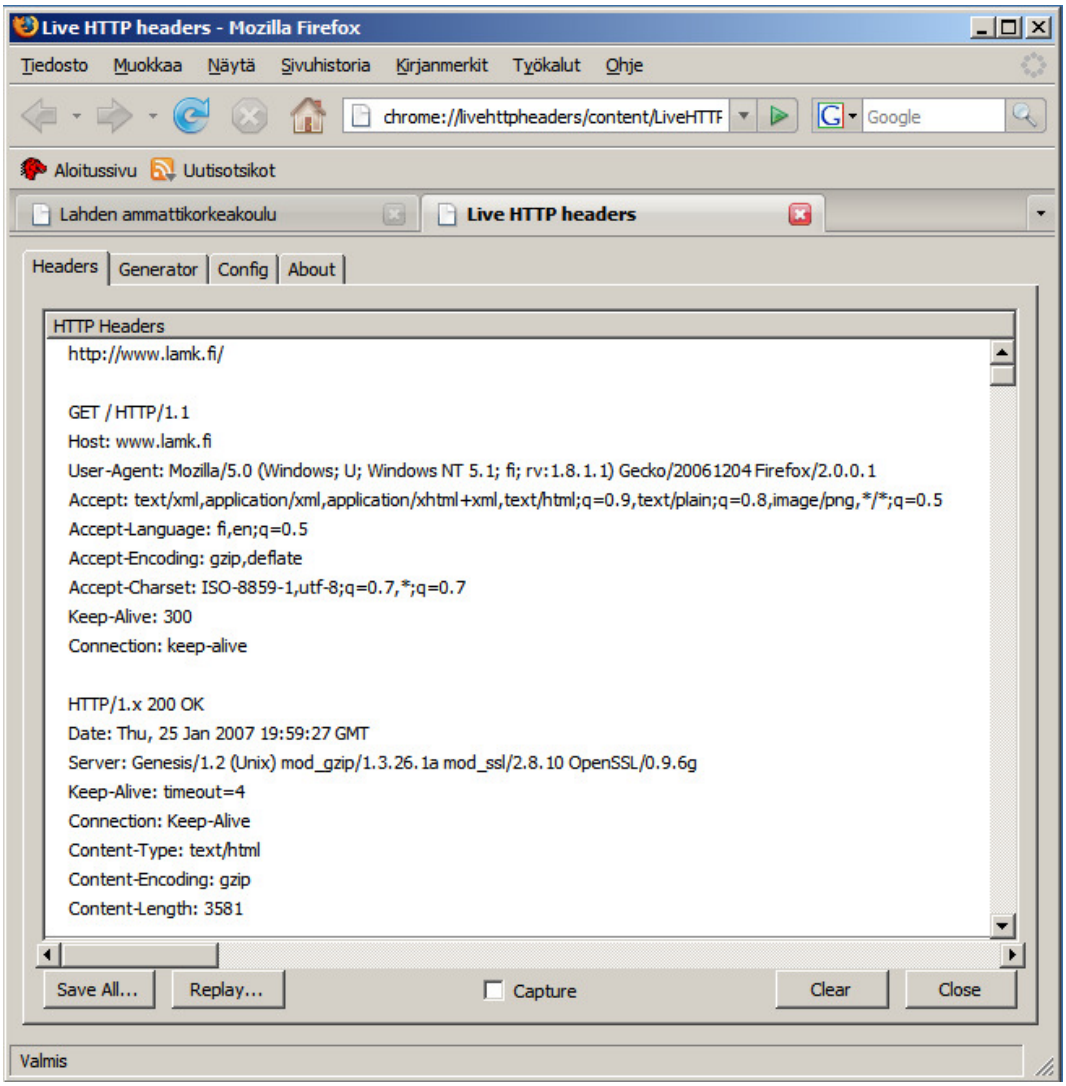
Kääntöympäristöksi valittiin CMake. CMake osaa tehdä ohjaustiedostojen pohjalta kohdeympäristölle sopivat projektitiedostot, joten samoista lähdekoodeista saadaan kääntymään sovelluksesta omat versiot eri käyttöjärjestelmille. Linux-ympäristössä CMake luo Makefile tiedostot, jotka käyttävät GCC-kääntäjää. Windows-ympäristössä kääntäjäksi valittiin ilmainen Visual C++ express edition. CMake osaa tehdä ohjaustiedostojen pohjalta Visual C++:lle sopivat projektitiedostot, joten lähdekoodit saa käännettyä käyttäen Visual C++:n käyttöliittymää.

3.2.7 Testisekvenssin tallennus

Referenssisovelluksena olleessa Webserver Stress Tool 7:ssa on toiminto, jolla saadaan nauhoitettua palvelimelle lähetetyt pyynnöt kun käyttäjä lataa palvelimella olevia sivuja samoin kuin normaalia selainta käyttäessä. Samalta valmistajalta löytyy myös vastaava erillinen sovellus nimeltä Paessler URL Recorder. URL recorder toimii kuitenkin ainoastaan Windows-ympäristössä, joten ohjelmalle etsittiin parempaa vaihtoehtoa. Vaihtoehdoksi löytyi Mozilla Firefoxin avoimen lähdekoodin laajennus nimeltä Live HTTP headers.

Live HTTP headers tallentaa palvelimelle lähetetyt pyynnöt sellaisenaan samalla kun käyttäjä selaa palvelimella olevia sivustoja. Lisäksi palvelimelta tulleiden vastausten otsakekentät tallentuvat muistiin. Live HTTP headers:n tallentamat tiedot saa tallennettua tiedostoon. Tallennetut tiedot ovat otsakekenttien osalta ASCII-dataa, joten tallennettua tiedostoa voi muokata halutessa normaalilla tekstieditorilla. Mikäli tallennetussa HTTP-pyynnössä on lähetetty binaaridataa, tallentuu myös se tiedostoon, joten tämä pitää ottaa huomioon tiedoston käsittelyssä.

Koska Live HTTP headers on avoimen lähdekoodin sovellus, on sen toiminnallisuutta mahdollista kehittää, mikäli tarvetta esiintyy. Live HTTP headers on kirjoitettu javascript:llä.



KUVIO 8. Live HTTP headers

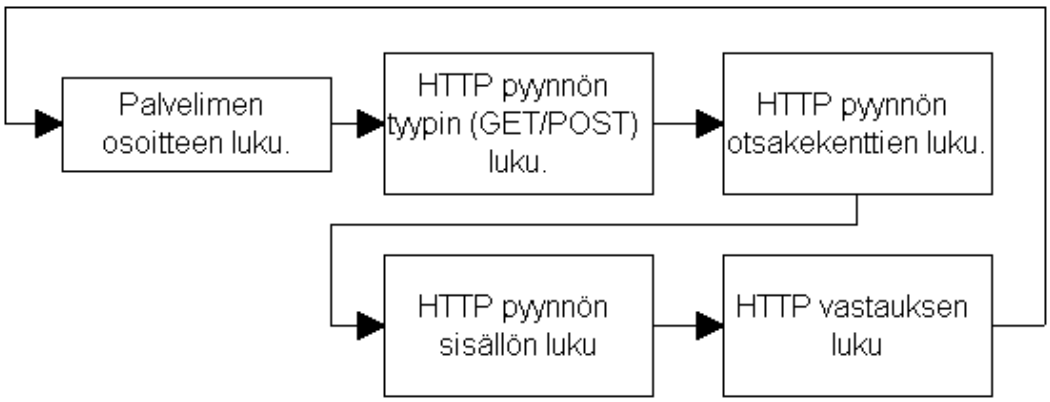
KUVIO 8:n kuvaruutukaappauksessa näkyy Live HTTP headers:n tallentamana ensimmäinen palvelimelle lähtenyt pyyntö sekä palvelimelta saapunut vastaus.

Kuvassa ensimmäinen rivi on kohdepalvelimen osoite, joka ei kuulu HTTP-protokollan dataan. Seuraava osa on palvelimelle lähetty HTTP-pyyntö. Seuraava, tyhjällä rivillä erotettu osa on palvelimelta tullut vastaus.

3.2.8 Testisekvenssin tulkitseminen

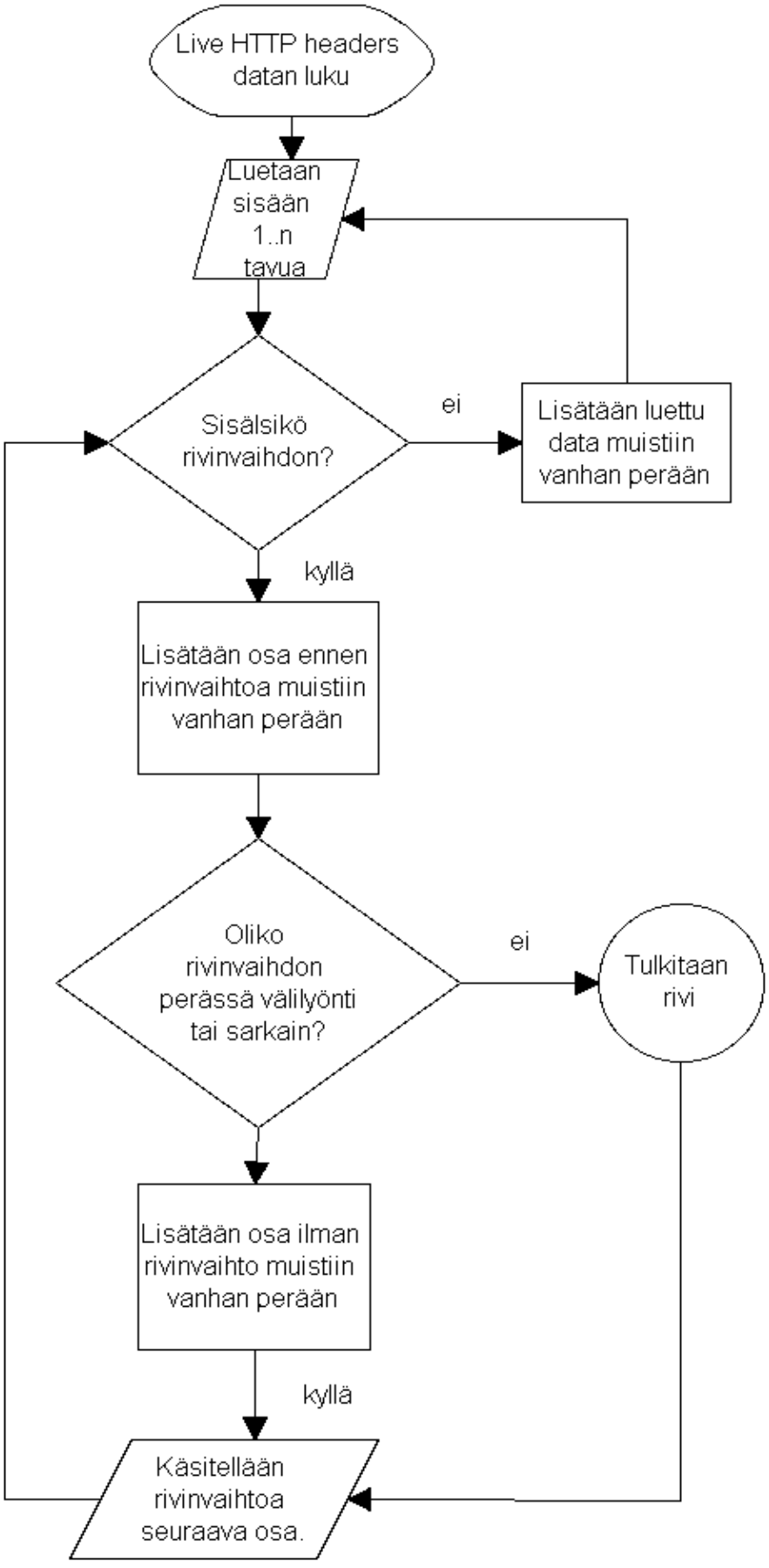
Jotta tallennettu testisekvenssi voitaisiin toistaa, täytyy ohjelmiston osata tulkita sitä. Tallennetusta sekvenssistä pitää erotella eri pyynnöt toisistaan ja ohittaa testisovelluksen kannalta turhat palvelinvastukset. Pyyntöjen erottelussa ei riitä pelkkä rivien lukeminen ja tietyn tiedon etsiminen, koska HTTP-protokollassa rivit voivat jatkua toiselta fyysiseltä riviltä, mikäli rivinvaihtoa seuraa valkomerkki. Lisäksi pyynnöt saattavat sisältää erottimien sisässä mitä tahansa dataa, joten myös erottimet pitää etsiä datasta. Erottimet vastaavasti kerrotaan otsakekentissä, joten myös otsakekenttien rakenne täytyy tuntea.

KUVIO 9 esittää Tekstisekvenssin HTTP-transaktiota (HTTP-pyyntö ja -vastaus). Tekstisekvenssi voi sisältää yhden tai useamman palvelimen osoitetiedolla alkavan HTTP-transaktion.



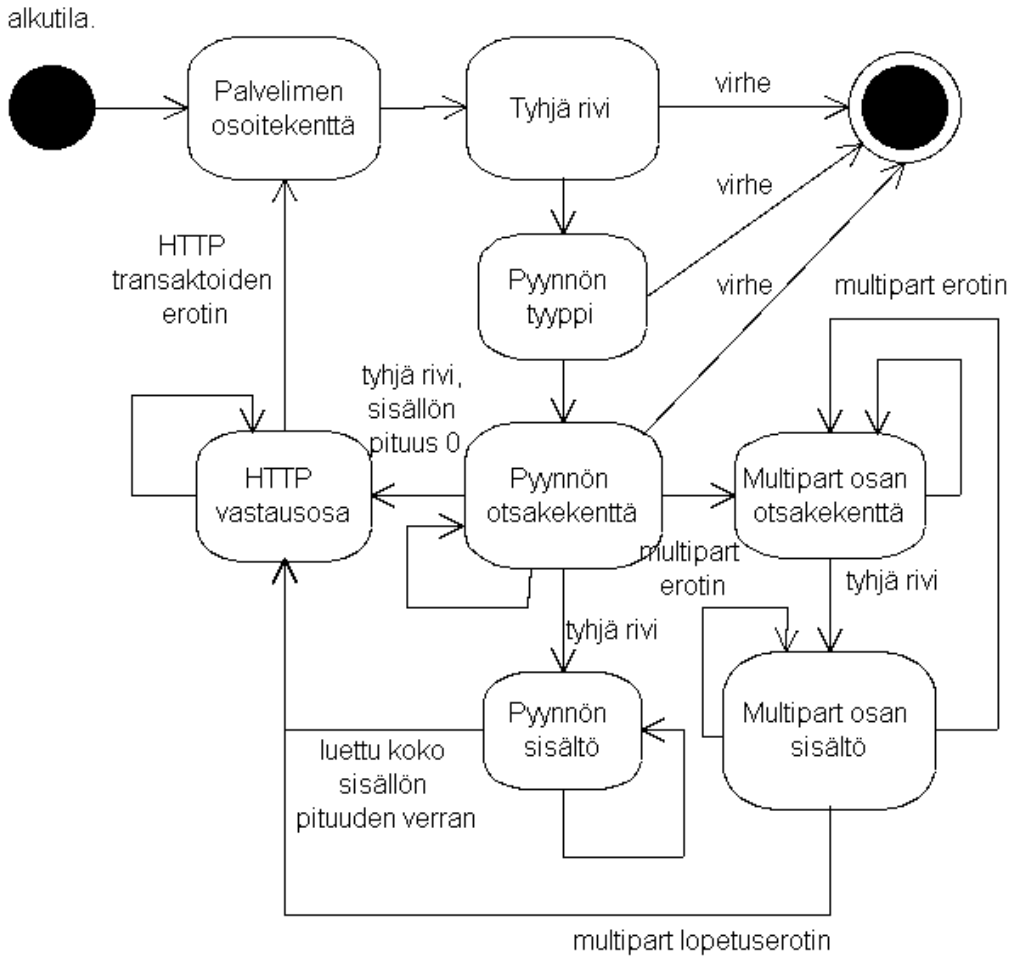
KUVIO 9. Testisekvenssin HTTP-transaktio

Palvelimen osoite on omalla rivillään, jota seuraa aina yksi tyhjä rivi. Seuraava osa alkaa aina rivillä joka sisältää mm. HTTP-pyynnön tyypin. HTTP-pyynnön tyyppiä seuraavat HTTP-pyynnön otsakekentät, joiden lukumäärä ei ennakkoon tiedetä. Otsakekentät saattavat olla jakaantuneena useammalle riville, kuten on aikaisemmin todettu. KUVIO 10 esittää yksinkertaistetun vuokaavion testisekvenssin luvusta osissa sekä useammalle riville jakaantuneiden otsakekenttien yhdistämisestä.


Live HTTP headers datan luku
Luetaan sisään 1..n tavua
Sisälsikö rivinvaihdon?
ei
Lisätään luettu data muistiin vanhan perään
kyllä
Lisätään osa ennen rivinvaihtoa muistiin vanhan perään
Oliko rivinvaihdon perässä välilyönti tai sarkain?
ei
Tulkitaan rivi
Lisätään osa ilman rivinvaihto muistiin vanhan perään
kyllä
Käsitellään rivinvaihtoa seuraava osa.

KUVIO 10. Testisekvenssin rivin luku

Luettu rivi annetaan eteenpäin tulkittavaksi tilakoneelle, joka pitää yllä tietoa siitä missä kohtaa testisekvenssiä tiedon luku on. Tilakoneen eri tilat ja siirtymät on kuvattuna KUVIO 11:ssä.



KUVIO 11. Testisekvenssin tulkitsemisen tilakone

Tilakone odottaa alkutilanteessa saavansa rivin, jossa on palvelimen osoite. Seuraavaksi tulevan rivin pitäisi olla tyhjä, ja mikäli se ei ole, tästä annetaan virhe. Seuraavaksi alkaa HTTP-pyyntö, jonka ensimmäisenä rivinä tulkitaan pyynnön tyyppi. Mikäli tässä kohtaa tulee tyhjä rivi, annetaan virhe ja lopetetaan testisekvenssin tulkinta. Pyynnön tyyppiä seuraavat otsakekentät.

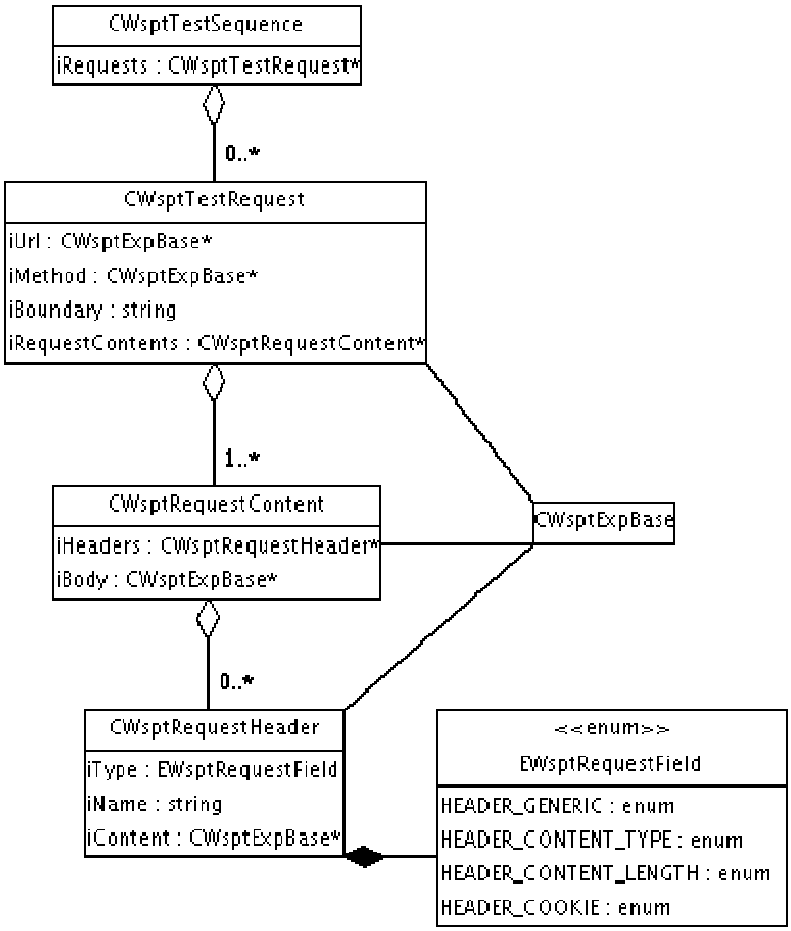
Otsakekenttien sisältö tutkitaan mahdollisen multipart-erottimen tunnistamiseksi, sekä mahdollisen pyynnön sisällön pituuden tallentamista varten. Otsakekenttien rakenne tarkistetaan ja virheellisestä rakenteesta annetaan virhe ja lopetetaan testisekvenssin tulkinta. Normaalien HTTP-protokollan mukaisten otsakekenttien lisäksi ohjelma tulkitsee wspt-request ja wspt-response nimiset kentät, joita käytetään syötteiden automatisointiin.

Otsakekenttiä seuraa tyhjällä rivillä erotettuna pyynnön sisältö mikäli otsakekentissä oli content-length-kenttä. Otsakekenttiä saattaa myös seurata ns. multipart-osa. Näitä seuraa HTTP-vastausosa, jonka sisältö jätetään käsittelemättä. Tilakone siirtyy lukemaan uutta HTTP-transaktiota, kun luettu rivi sisältää Live HTTP headers ohjelman käyttämän HTTP-transaktioiden erottimen.

3.2.9 Testisekvenssin sisäinen rakenne

Kun Live HTTP headers ohjelman luomaa tiedostorakennetta tulkitaan, se luetaan ohjelman sisäiseen tietorakenteeseen, josta testisekvenssiä on nopea lukea sekvenssiä toistettaessa. Lisäksi rakenne tukee syötteiden automatisointia sekä mahdollistaa sen, että ohjelmaan voidaan myöhemmin lisätä tuki muillekin testisekvenssityypeille, kuin mitä Live http headers tuottaa. Testisekvenssin toisto käyttää sisäistä rakennetta, joten mikäli uudelle testisekvenssityypille lisätään tuki, tarvitaan ainoastaan tähän tulkintaan uusi toteutus, muihin osiin ei tarvita muutoksia.

KUVIO 12 esittää luokkakaavion testisekvenssin rakenteesta. Testisekvenssi (CWsptTestSequence) sisältää listan sekvenssiin kuuluvista HTTP-pyynnöistä (CWsptTestRequest). HTTP-pyynnöllä on pyyntöä koskeva URL, pyynnön tyyppi, sekä mahdollisen multipart-tiedon erottimen (iBoundary). HTTP-pyynnön otsakekentät ja sisältö ovat omassa luokassa (CWsptRequestContent). Nämä tiedot on eriytetty omaan luokkaan, koska myös mahdollinen multipart-tieto käyttää samoja tietotyyppejä. HTTP-pyyntöön kuuluu aina vähintään yksi CWsptRequestContent-luokan instanssi, joka sisältää kyseisen pyynnön tiedot. Mikäli pyyntö on multipart-tyyppiä, sisältyy listaan oma CWsptRequestContent-luokan instanssi jokaista multipart-viestin osaa kohden.



KUVIO 12. Testisekvenssin luokkakaavio

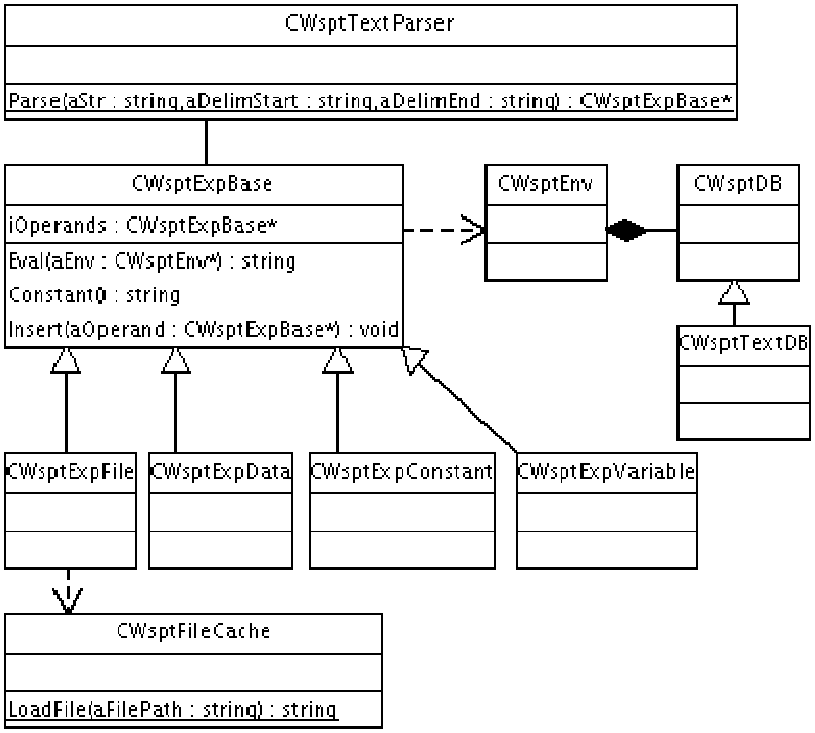
CWsptRequestContent-luokka sisältää HTTP-pyyntöön, tai multipart osaan liittyvät otsakekentät sekä sisältöosan. Otsakekentät ovat omassa luokassaan (CWsptRequestHeader), joka sisältää otsakekentän nimen, tyypin, sekä sisällön. Otsakekentän tyyppi on asetettu testisekvenssin tulkintavaiheessa otsakekentän nimen perusteella. Tätä tietoa käytetään ajovaiheessa, jotta tiettyjen otsakekenttien vaatimat erityiskäsittelyt voidaan suorittaa.

Osa tiedoista on tallennettuna CWsptExpBase-luokan ilmentymiin. Näiden tietojen sisällöt luodaan ajonaikaisesti. Tämä mahdollistaa syötteiden automatisoinnin.

3.2.10 Testisekvenssin syötteiden automatisointi

Luokkarakenne

Yhtenä sovelluksen asiakasvaatimuksena oli virtuaalikäyttäjäkohtaisten tietojen käsittely. Esimerkiksi eri käyttäjien pitää pystyä kirjautumaan sivustolle eri käyttäjätunnuksilla valmiiksi luotujen tietojen pohjalta. Vaatimus täytettiin luomalla rakenne, jossa osa testisekvenssi tiedoista voidaan korvata ajonaikaisesti luotavilla tiedoilla. Ohjelmiston testisekvenssissä nämä tiedot ovat CWsptExpBase-luokasta perittyjen luokkien ilmentymissä. KUVIO 13 esittää CWsptExpBase- luokasta perityt luokat, sekä muut automaattisten syötteiden käsittelyyn liittyvät luokat.



KUVIO 13. CWsptExpBase luokkakaavio

CWsptExpBase

CWsptExpBase-luokka sekä tästä perityt luokat sisältävät Eval-metodin, jota käytetään palauttamaan testisekvenssin ajon aikana luokan ilmentymän sisältämän datan. CWsptExpBase-luokka luo palautettavan datan käyttämällä parametreinaan iOperands-listassa olevia muita CWsptExpBase-luokan ilmentymiä. Eval-metodi käy iOperands-listan läpi, kutsuu kunkin listassa olevan luokan Eval-metodia, yhdistää Eval-kutsujen paluuarvot yhteen ja lopuksi palauttaa kootut paluuarvot.

CWsptExpFile

CWsptExpFile-luokka palauttaa Eval-kutsussa tiedoston sisällön. Käytettävän tiedoston nimi muodostetaan iOperands-listan perusteella. Luokka käyttää CWsptFileCache-luokan instanssia tiedostojen lataamiseen.

CWsptFileCache

CWsptFileCache-luokka lataa pyydetyn tiedoston sisällön muistiin ja palauttaa viittauksen tiedoston sisältöön kutsujalle. Kun CWsptFileCache-luokan LoadFile-metodia kutsutaan uudelleen samalla tiedoston nimellä, palautetaan viittaus edellisellä kerralla ladatun tiedoston sisältöön. CWsptFileCache-luokkaa käytetään säästämään muistia, koska on oletettavaa että testisekvenssin ajossa useampi virtuaalikäyttäjä käyttää samaa tiedostoa ja virtuaalikäyttäjiä voi olla tuhansia.

CWsptExpData

CWsptExpData-luokkaa käytetään lataamaan tietoja yksinkertaisesta tietokannasta. Eval-kutsussa kutsutaan iOperands-listassa oleville luokille Eval-kutsua sekä muodostetaan näistä tieto, jota käytetään viittaamaan tietokannan tiettyyn alkioon. Muodostetusta viittauksesta etsitään kaksoispiste. Kaksoispisteen vasemman puoleisesta osasta muodostetaan viitattavan kentän nimi. Oikean puoleisesta osasta muodostetaan rivi, johon tietokannassa viitataan. Esimerkiksi username:0 hakee tietokannasta ensimmäisestä rivistä (rivi numero 0) ”username” nimisen kentän arvon.

CWsptDB

CWsptDB on abstrakti kantaluokka tietokannoille, jota CWsptExpData-luokka voi käyttää. Tietokannoilla on Fetch niminen metodi, jolle annetaan haettavan kentän nimi ja rivin numero.

CWsptTextDB

CWsptTextDB on CWsptDB luokasta peritty luokka, joka lataa record-jar muotoa vastaavasta tekstitiedostosta (TAULUKKO 11) tiedot muistiin, josta Fetch-metodi palauttaa kutsujalle halutun kentän sisällön.

TAULUKKO 11 CWsptTextDB tiedostomuoto

Muoto	**Esimerkki**
Kentän nimi: kentän arvo	Username: testaaja
Kentän nimi: kentän arvo	Password: erittäinsalainen
Tietokantarivien erotin, aina %%	%%
Kentän nimi: kentän arvo	Username: admin
Kentän nimi: kentän arvo	Password: vaikea

Kun esimerkin mukaiselle kannalle suoritetaan Fetch(0, ”Password”) kutsu, saadaan paluuarvona teksti ”erittäinsalainen”. CWsptTextDB-luokkaa voidaan käyttää testisekvenssissä mm. antamaan jokaiselle virtuaalikäyttäjälle omat kirjautumistunnukset, kun haussa rivinumeroksi annetaan kunkin virtuaalikäyttäjän tunnistenumero.

CWsptExpConstant

CWsptConstant-luokka palauttaa Eval-metodia kutsuttaessa vakiotiedon. CWsptContant-luokalla ei voi olla iOperands-listassa muita luokkia.

CWsptExpVariable

CWsptExpVariable-luokkaa käytetään palauttamaan halutun muuttujan arvo tai kutsun paluuarvon. Muuttujan tai komennon nimi muodostetaan iOperands-listan luokkien Eval-kutsujen perusteella muodostusta merkkijonosta. Merkkijono pilkotaan kaksoispisteiden perusteella osiin. Ensimmäinen osa tulkitaan muuttujan tai komennon nimeksi. Mahdolliset seuraavat osat tulkitaan parametreiksi komennolle. Esimerkiksi ”add:1:3” tulkitaan add-nimiseksi komennoksi, jonka parametreina ovat ”1” sekä ”3”. CWsptExpVariable kutsuu CWsptEnv-luokan GetVariable-metodia ja palauttaa tämän paluuarvon kutsujalle.

CWsptEnv

CWsptEnv-luokkaa käytetään virtuaalikäyttäjäkohtaisena ympäristönä, jonka avulla testisekvenssin ajossa voidaan tallentaa ja lukea erilaisia muuttujia sekä ajaa erinäisiä komentoja. Yksi esimerkki CWsptEnv-luokan sisältämistä komennoista on ”ID”, jolta testisekvenssi voi kysyä virtuaalikäyttäjän tunnistenumeron.

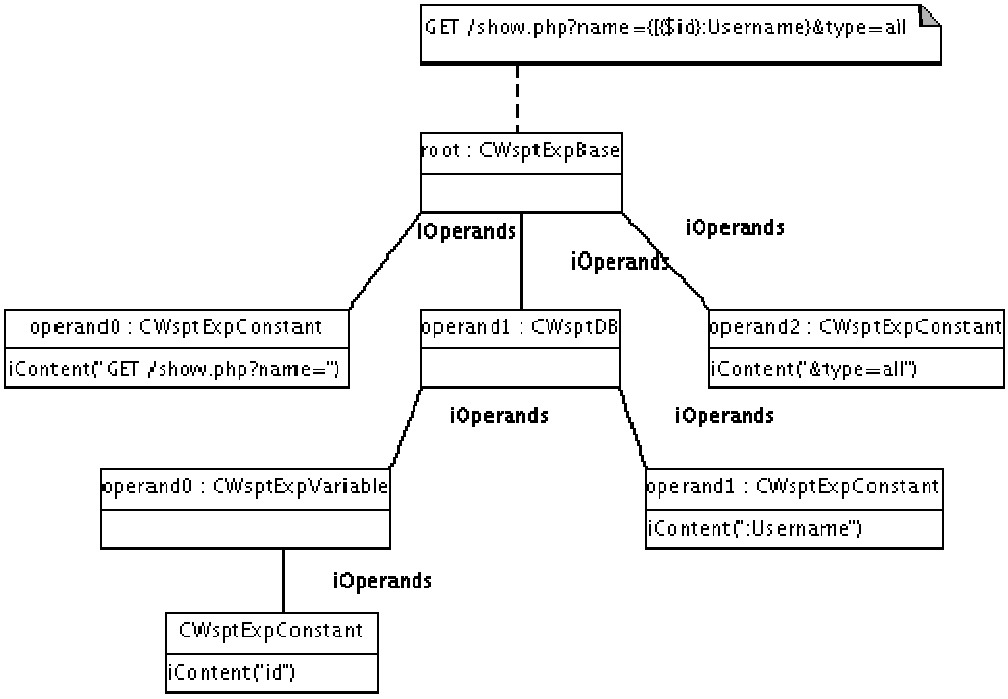
CWsptTextParser

CWsptTextParser-luokan Parse-metodia käytetään tekstimuodossa olevien automaattisten syötteiden muuntamiseksi CWsptExpBase-luokan instanssiksi. Metodille annetaan aloitus- ja lopetuserottimet, joista tekstissä olevat syötteet tunnistetaan. Syötteet alkavan syötteen tyyppiä kuvaavalla tunnisteella, jota seuraa itse syöte. CWsptTextParser tukee syötetyyppejä (TAULUKKO 12), joilla voidaan mm. lukea CWsptDB-kannasta tietoa.

TAULUKKO 12. Tuetut syötetyypit

Tunniste	**Syötetyypin luokka**	**Esimerkki**
$	CWsptExpVariable	$id
[	CWsptDB	[0:Password
@	CWsptExpFile	@../helloworld.cpp

CWsptTextParser tukee sisäkkäisiä syötteitä. KUVIO 14 esittää muistirakenteen, joka muodostuu, kun kuvan ylimmän osan esittämä teksti tulkitaan. Esimerkissä syötteiden alkuerottimena käytetään aloittavaa aaltosulkua. Vastaavasti lopetuserottimina toimii lopettava aaltosulku.



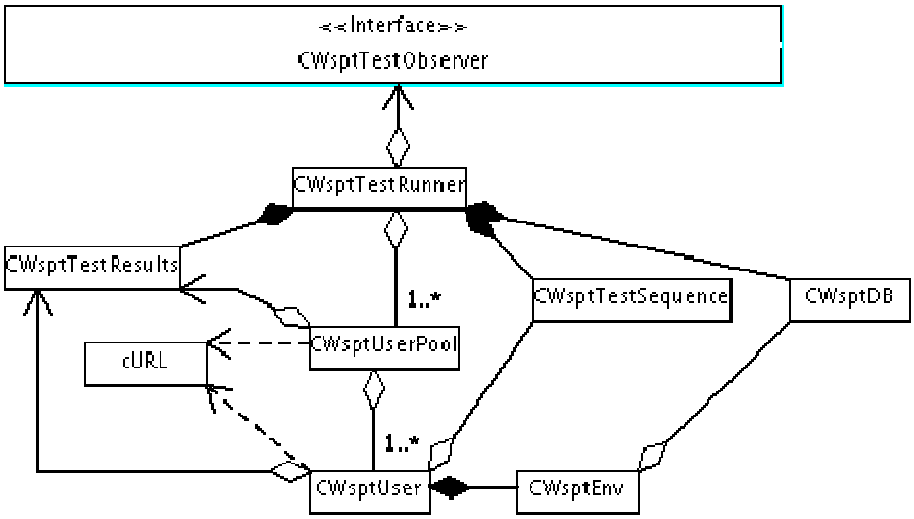
KUVIO 14. Oliokaavio syötteiden muistirakenteesta

Rakenteessa ylimpänä on CWsptExpBase-luokan olio, jolla on operandeinaan kolme CWsptExpBase:sta perityn luokan oliota: CWsptExpConstant, CWsptDB ja CWsptExpConstant. Vastaavasti CWsptDB-luokalla on operandeinaan CWsptExpVariable ja CWsptExpConstant. CWsptExpVariable:lla on ainoastaan yksi operandi, CWsptExpConstant, jonka arvo on ”id”.

Kun kuvassa ylimpänä näkyvälle CWsptExpBase-luokalle, jonka CwsptTextParser on palauttanut, kutsutaan testisekvenssin ajossa Eval-metodia, kutsuu CWsptExpBase operandeinaan oleville luokille Eval-metodia, jotka vuorostaan kutsuvat omille operandeilleen Eval-metodia jne.

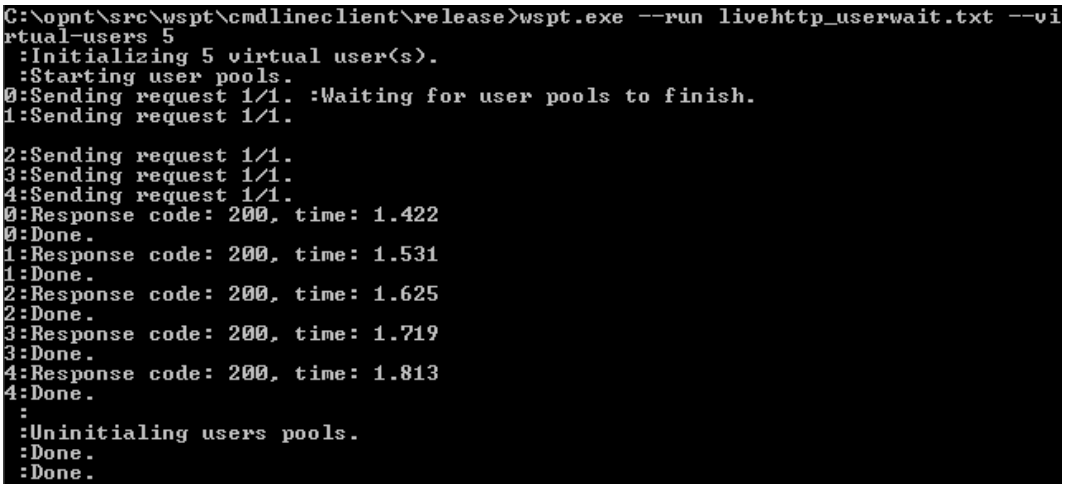
Mikäli kutsuvan virtuaalikäyttäjän tunnistenumero olisi nolla, ja käytettäisiin TAULUKKO 11:n tietokantaa, tulisi Eval-kutsun paluuarvoksi seuraava merkkijono: ”GET /show.php?name=testaaja&type=all”. Syötteiden käytöstä kerrotaan tarkemmin englanninkielisessä käyttöoppaassa (LIITE 2).

3.2.11 Testisekvenssin ajo

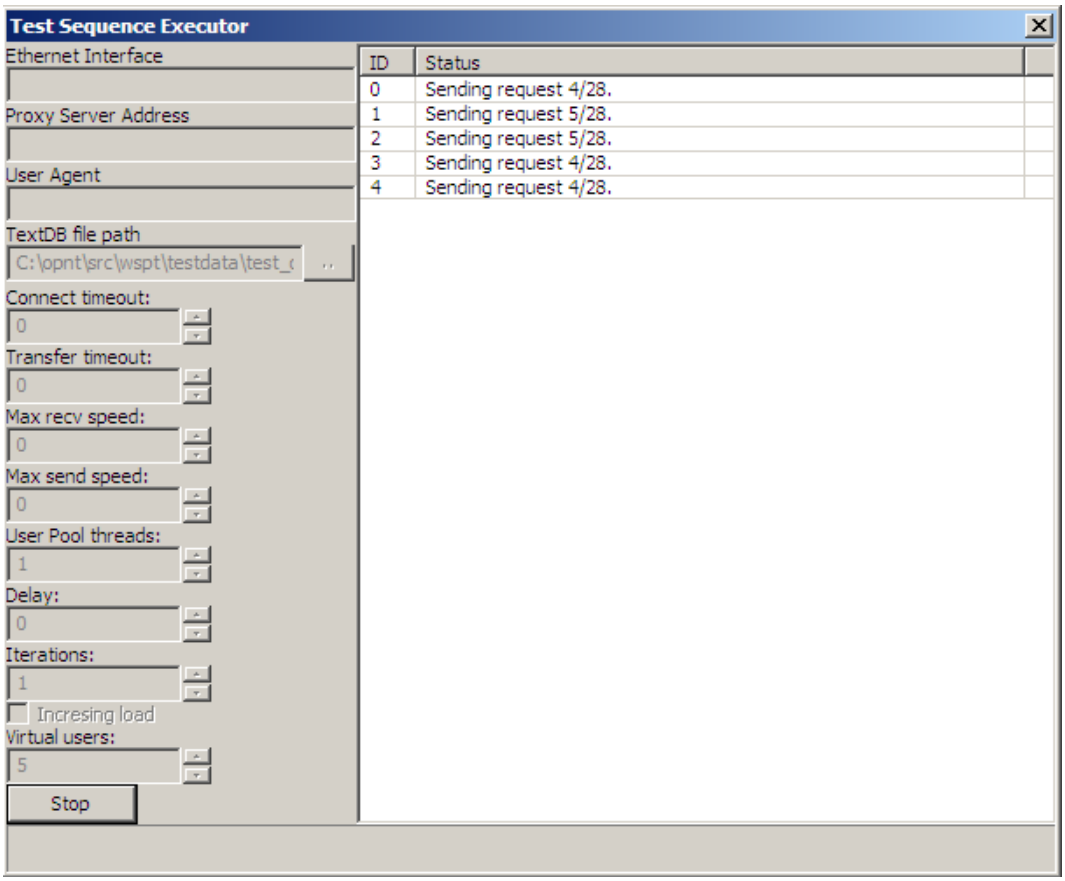


KUVIO 15. Luokkakaavio testisekvenssin ajosta

CWsptTestRunner on omassa säikeessä toimiva luokka, joka luo testien suorittamiseen tarvittavat CWsptUserPool-luokat sekä välittää CWsptUserPool-luokkien ilmentymiltä tulevat viestit eteenpäin luokalle, joka on periytetty CWsptTestObserver-luokasta. Komentoriviversiossa testin tapahtumista kertovat tapahtumat tulostetaan sellaisenaan ruudulle (KUVIO 16). Graafisella käyttöliittymällä varustettu versio (KUVIO 17) päivittää saamiensa viestien pohjalta listanäkymää, jossa kullekin virtuaalikäyttäjälle on oma tilarivi.



KUVIO 16. Kuvaruutukaappaus komentoriviversion toteutuksesta



KUVIO 17. Kuvaruutukaappaus graafisen käyttöliittymän versiosta

CWsptUserPool-luokka pitää listaa yhdestä tai useammasta CWsptUser-luokasta. CWsptUserPool toimii omassa säikeessään ja käyttää cURL-kirjaston curl_multi-rajapintaa suorittamaan useampia HTTP-pyyntöjä samanaikaisesti. Alkuperäisessä suunnitelmassa tätä luokkaa ei ollut, ja jokainen virtuaalikäyttäjä toimi omassa säikeessään. Tämä kuitenkin hukkaa käyttöjärjestelmän sekä suorittimen resursseja, eikä ohjelmalla voinut käytännössä ajaa tuhansia virtuaalikäyttäjiä, joten väliin lisättiin curl_multi-rajapintaa käyttävä luokka, joka suorittaa useampien virtuaalikäyttäjien toiminnot samassa säikeessä.

CWsptUser-luokka käsittelee yksittäisen virtuaalikäyttäjän toiminnot. Luokka luo testisekvenssin tietojen pohjalta tarvittavat HTTP-pyynnöt käyttämällä cURL-kirjaston curl_easy-rajapintaa.

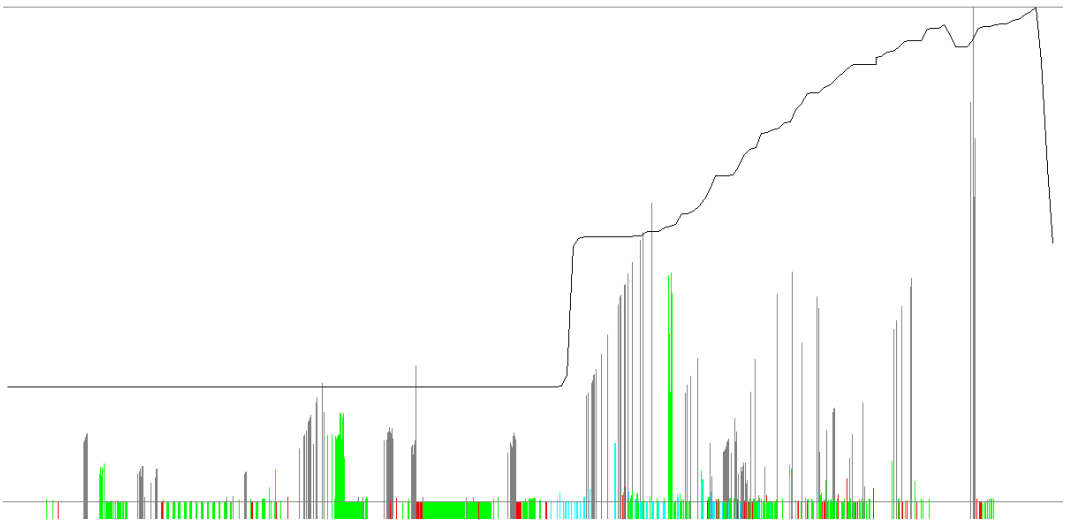
Kun CWsptUser-luokka on saanut cURL-kirjastolta ilmoituksen HTTP-pyynnön suoritukselta, kysytään cURL-kirjastolta pyyntöön liittyneet vastauskoodit sekä vasteajat. Nämä tiedot, sekä aikaleima lisätään CWsptTestResults-luokan ylläpitämään listaan testin tuloksista. Kun koko testi on suoritettu, CWsptTestObserver-rajapinnan kautta lähetetään ilmoitus ylemmälle tasolle. Testin suorituksen jälkeen testitulokset voidaan pyytää CWsptTestRunner-luokalta.

3.2.12 Testitulokset

Testisekvenssin ajossa syntyneet testitulokset tallentuvat CWsptTestResults-luokan ylläpitämään listaan. Komentoriviversio tallentaa tulokset tai tulostaa ruudulle CSV (Comma Separated Values) muodossa (KUVIO 18). Tämän muotoinen data voidaan viedä esimerkiksi Exceliin. Graafisen käyttöliittymän versio näyttää testitulokset projektin ikkunassa, josta tulokset voidaan viedä joko CSV-muodossa tai kuvana (KUVIO 19).

```
timestamp, userid, requestid, usercount, responsecode, namelookup, connect, pret
ransfer, starttransfer, total, speedup, speeddown, sizeup, sizedown
2007-01-28T16:54:14.843750,0,0,5,200,0,0,0.015,0.015,0.031,0,0,0,0
2007-01-28T16:54:14.953125,1,0,5,200,0,0.015,0.015,0.125,0.141,0,0,0,0
2007-01-28T16:54:15.046875,2,0,5,200,0,0,0.015,0.218,0.234,0,0,0,0
2007-01-28T16:54:15.140625,3,0,5,200,0.016,0.016,0.031,0.328,0.328,0,0,0,0
2007-01-28T16:54:15.234375,4,0,5,200,0.016,0.031,0.031,0.422,0.422,0,0,0,0
```

KUVIO 18. Testitulokset komentoriviversiossa

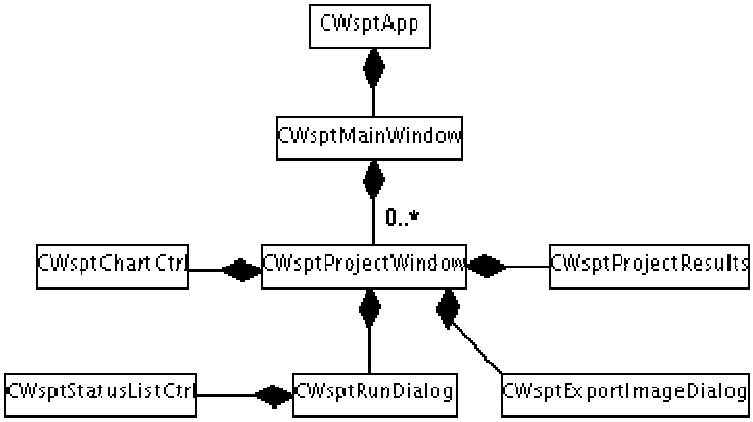


KUVIO 19. Testitulos vietynä graafisesta käyttöliittymäversiosta

3.2.13 Graafinen käyttöliittymä

Luokkarakenne

Graafinen käyttöliittymä toteutettiin käyttäen wxWidgets-kirjastoa. KUVIO 20 esittää ohjelman luokkarakenteen olennaisimpien luokkien osalta.



KUVIO 20. Graafisen käyttöliittymän luokat

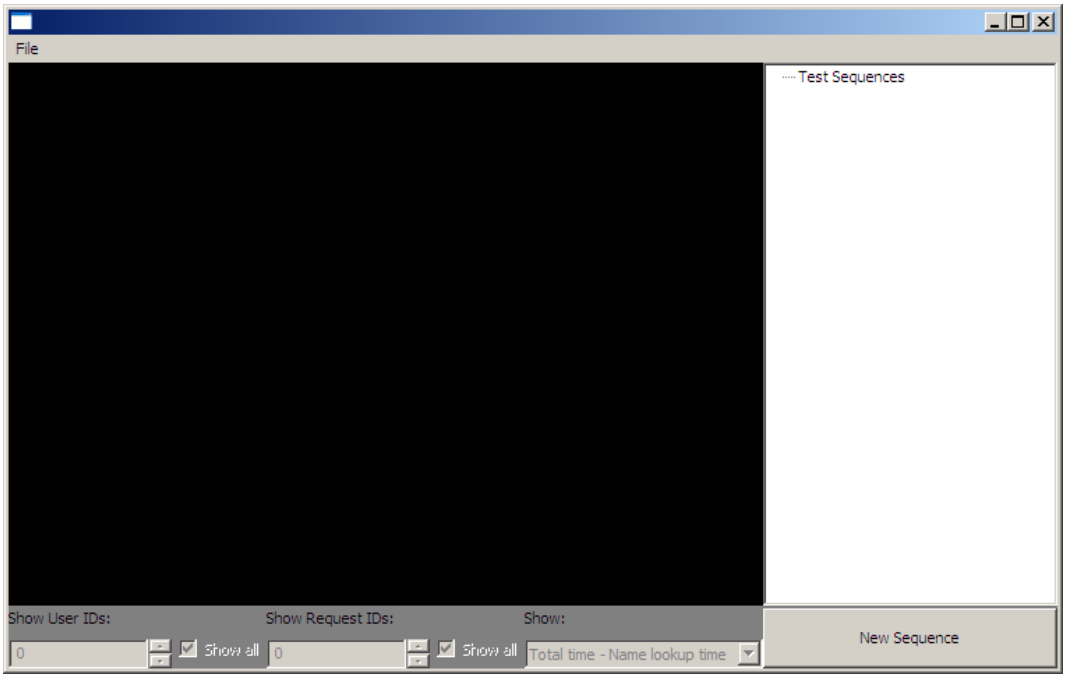
CWsptApp

CWsptApp-luokka on jokaisessa wxWidgets-kirjastoa käyttävässä ohjelmassa tarvittava luokka. Tässä sovelluksessa luokan ainoana tehtävänä on luoda CWsptMainWindow-ikkuna (KUVIO 21).

KUVIO 21. Kuvaruutukaappaus CWsptMainWindow-luokan ikkunasta

CWsptMainWindow

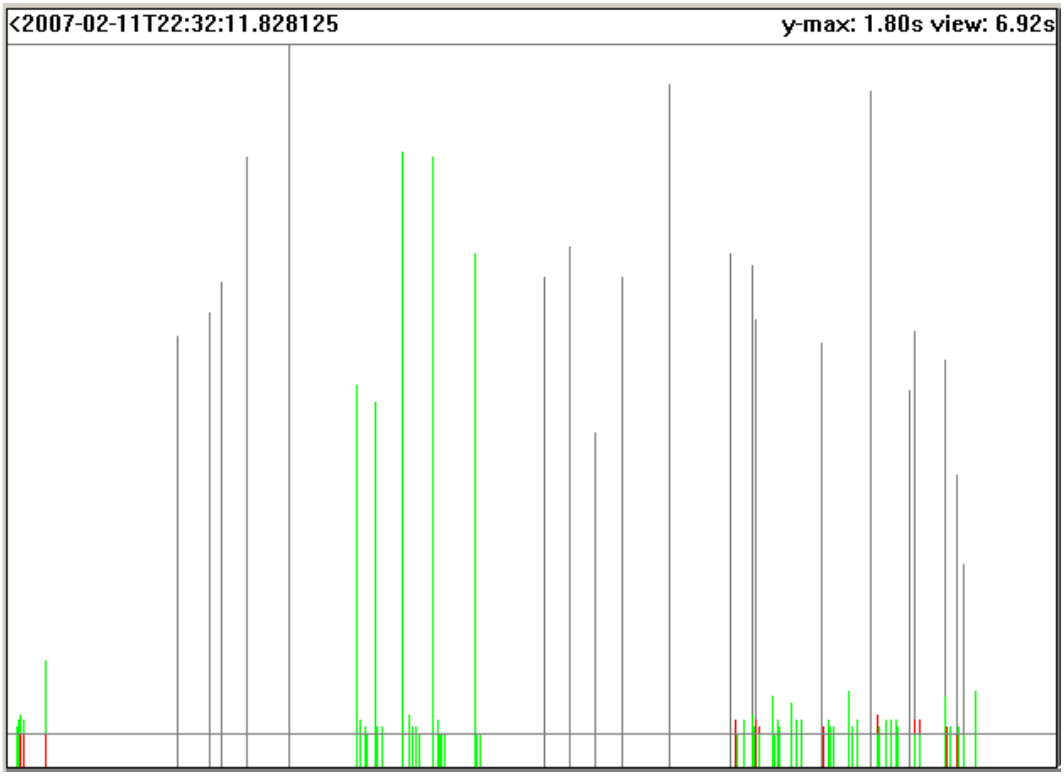
CWsptMainWindow-luokan ikkuna sisältää valikon, josta voidaan luoda uusi projekti tai avata tallennettu projekti. Uuden projektin luonti avaa CWsptProjectWindow-luokan ikkunan (KUVIO 22).



KUVIO 22. CWsptProjectWindow-luokan ikkuna

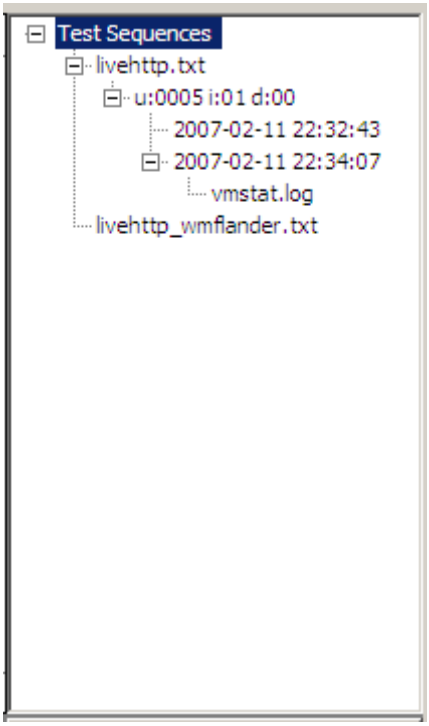
CWsptProjectWindow

CWsptProjectWindow-luokan ikkuna sisältää kaksi pääkomponenttia: puunäkymän testisekvensseistä sekä CWsptChartCtrl komponentin, joka näyttää kuvaajia testituloksista (KUVIO 23).



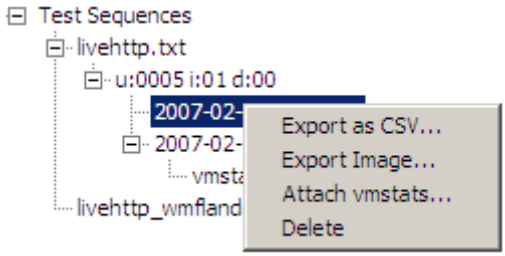
KUVIO 23. CWsptChartCtrl-komponentti

Testisekvenssin puurakenteessa (KUVIO 24) ylimpänä on testisekvenssi. Testisekvenssi ladataan Live http headers:n tallentamasta tekstitiedostosta. Testisekvenssillä on asetukset testiajolle. Testiajossa on määritelty mm. yhtäaikaisten virtuaalikäyttäjien määrä. Testiajoasetusten alla on testiajot, joiden aikaleimat näkyvät puurakenteessa. Testiajoilla voi olla liitettynä palvelimella suoritetun vmstat-ohjelman tulostama loki, jossa selviää mm. muistin käyttö eri kellonaikoina.



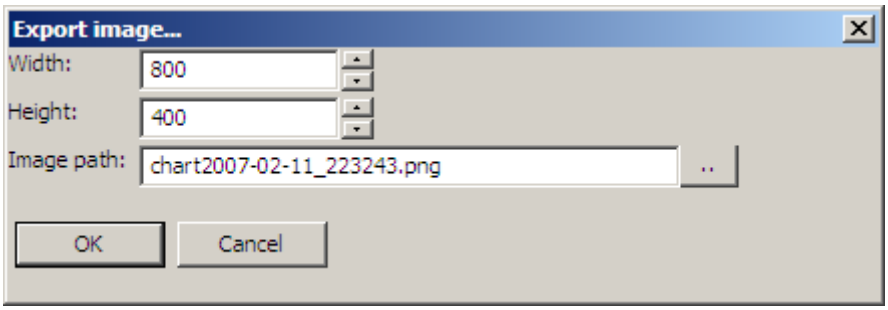
KUVIO 24. Testisekvenssien puurakenne

Puurakenteesta voidaan avata valikko (KUVIO 25), jonka kautta voidaan tallentaa tulokset CSV-muodossa tai viedä kuvaaja bittikarttakuvana.



KUVIO 25. Testisekvenssin puurakenteen valikko

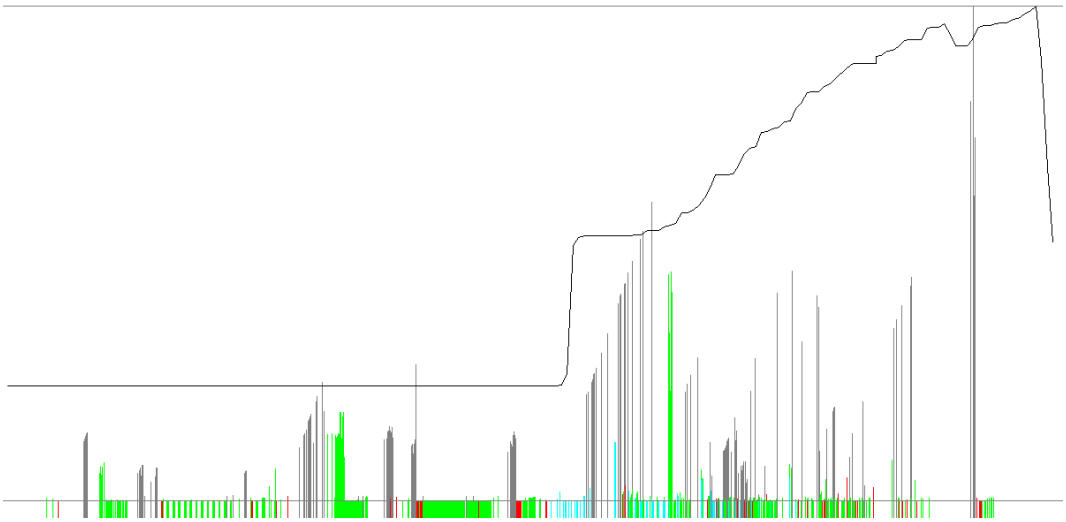
Kuvan vienti vaihtoehto avaa uuden ikkunan (KUVIO 26), josta voidaan valita luotavan kuvan koko, nimi sekä tallennusmuoto tiedoston päätteen perusteella. Tallennettavan kuvan kuvaaja näkyy samoilla valinnoilla, kuin kuvaaja on sovelluksen ruudulla tallennushetkellä.



KUVIO 26. Kuvaajan vienti kuvatiedostoon

CWsptChartCtrl

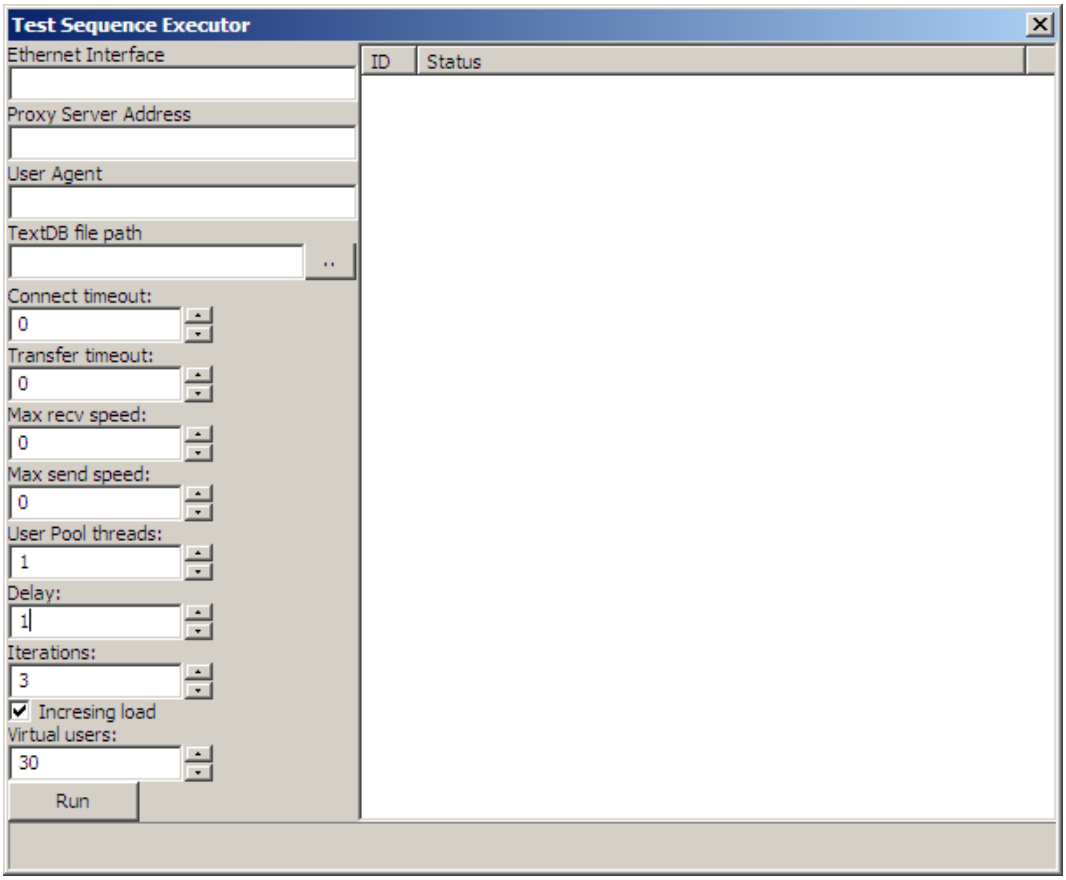
CWsptChartCtrl piirtää kuvaajia testisekvenssin ajon tuloksista. Kuvaajan näkyvän osan aikajanaa voidaan säätää hiiren rullalla sekä liikuttaa vetämällä hiirellä pitäen hiiren vasenta nappia samalla pohjassa. Kuvaajassa (KUVIO 27) näkyy pystyviivoja, joiden korkeus kuvaa HTTP-pyynnön suoritukseen kulunutta aikaa. Pystyviivan väri kertaa HTTP-vastauksen tilakoodin, esim. punainen kuvaa virhettä ja vihreä onnistunutta suoritusta. Kuvaajaan voidaan myös valita sovelluksen ulkopuolelta tuotuja tietoja. KUVIO 27:n esittämässä kuvaajassa näkyy, kuinka palvelimen virtuaalimuistin käyttö alkaa lisääntyä testin edetessä.



KUVIO 27. CWsptChartCtrl-komponentin piirtämä kuvaaja

CWsptRunDialog

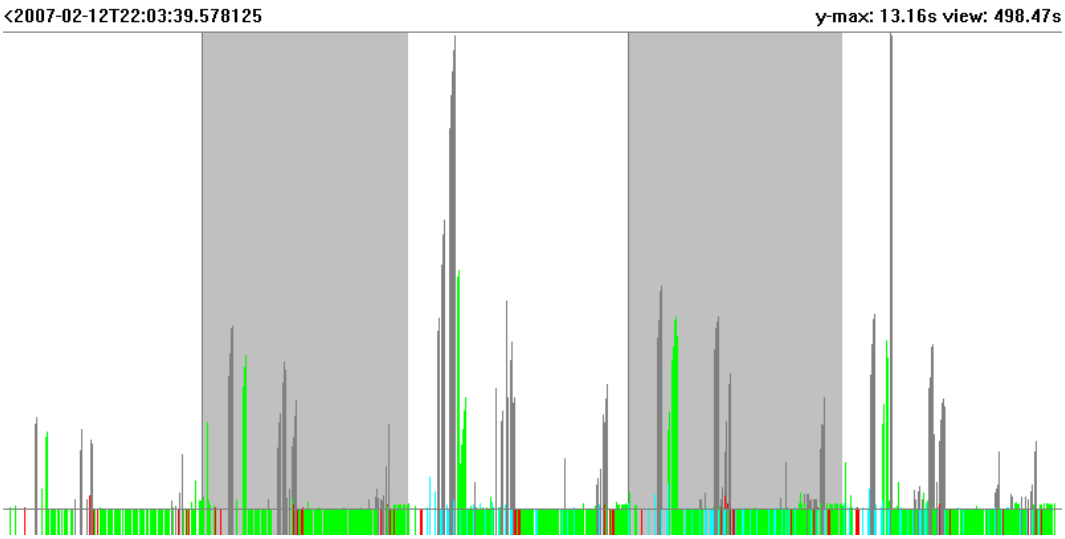
CWsptRunDialog-ikkunassa (KUVIO 28) valitaan ajettavan sekvenssin asetukset. Ikkunan oikeanpuoleinen listanäkymä näyttää testin edistymisen.



KUVIO 28. CWsptRunDialog-luokan ikkuna. Testiasetusten valinta

3.3 Käytännön testitapaus

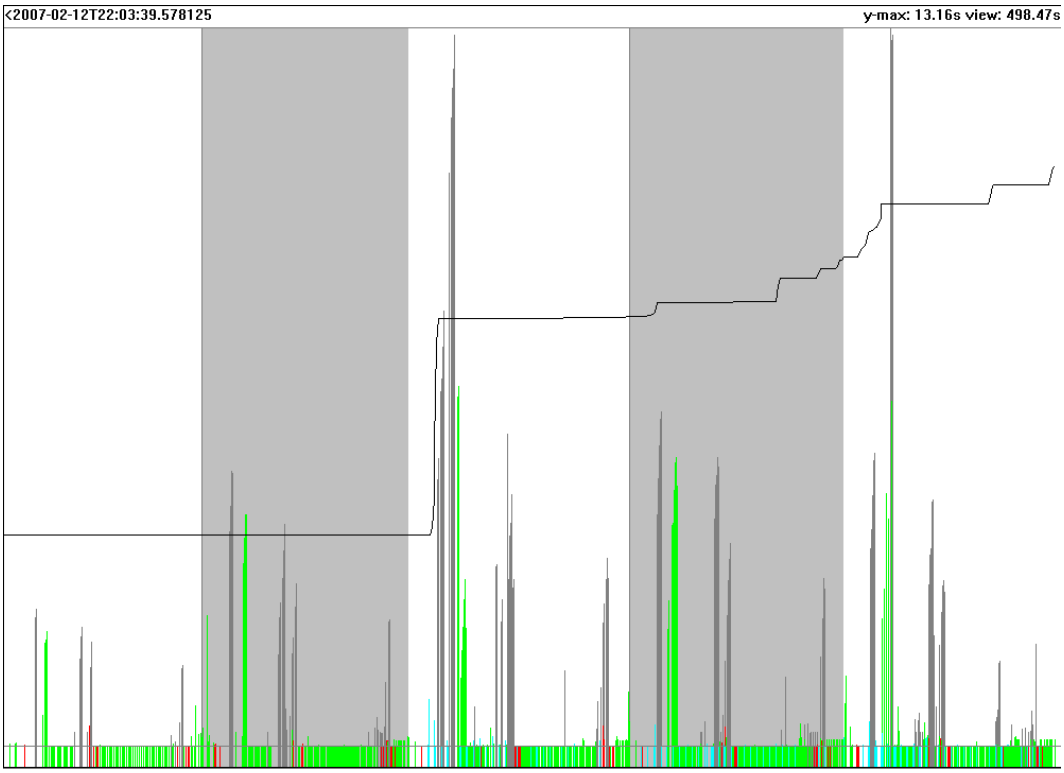
Sovelluksella ajettiin palvelimelle testisekvenssiä, jossa kirjaudutaan sivustolle ja suoritetaan tietokantakysely. Virtuaalikäyttäjien määrä asetettiin nousemaan vaiheittain 50 yhtäaikaiseen käyttäjään. 20 yhtäaikaisella käyttäjällä maksimivasteaika oli noin neljä sekuntia (KUVIO 29).



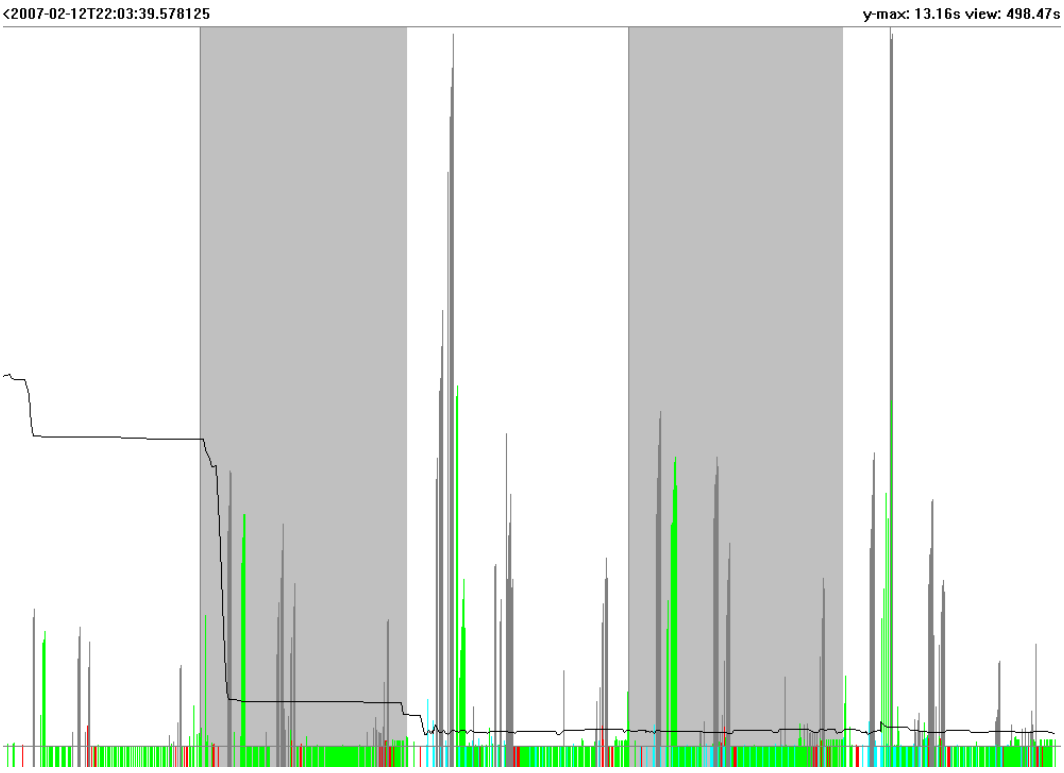
KUVIO 29. HTTP-pyyntöjen vasteaikojen kuvaaja

Kun käyttäjämäärä kasvoi 30:een, kohosivat vasteajat yli 13 sekunnin. Vasteaikapiikit eivät olleet tietokantahakujen kohdalla, joten pullonkaula ei näyttänyt olevan tietokannassa. Ilman palvelimella tehtyjä mittauksia voitiin vain todeta, että palvelin ei pystynyt palvelemaan yli 20 yhtäaikaista käyttäjää.

Palvelimella suoritetut mittaukset paljastivat välittömästi hidastumisen aiheuttajan. Palvelimen virtuaalimuistinkäyttö nousee jyrkästi samalla hetkellä, kun vasteajoissa näkyy piikki (KUVIO 30). Vapaan muistin määrä laskee jyrkästi jo aikaisemmassa vaiheessa (KUVIO 31).

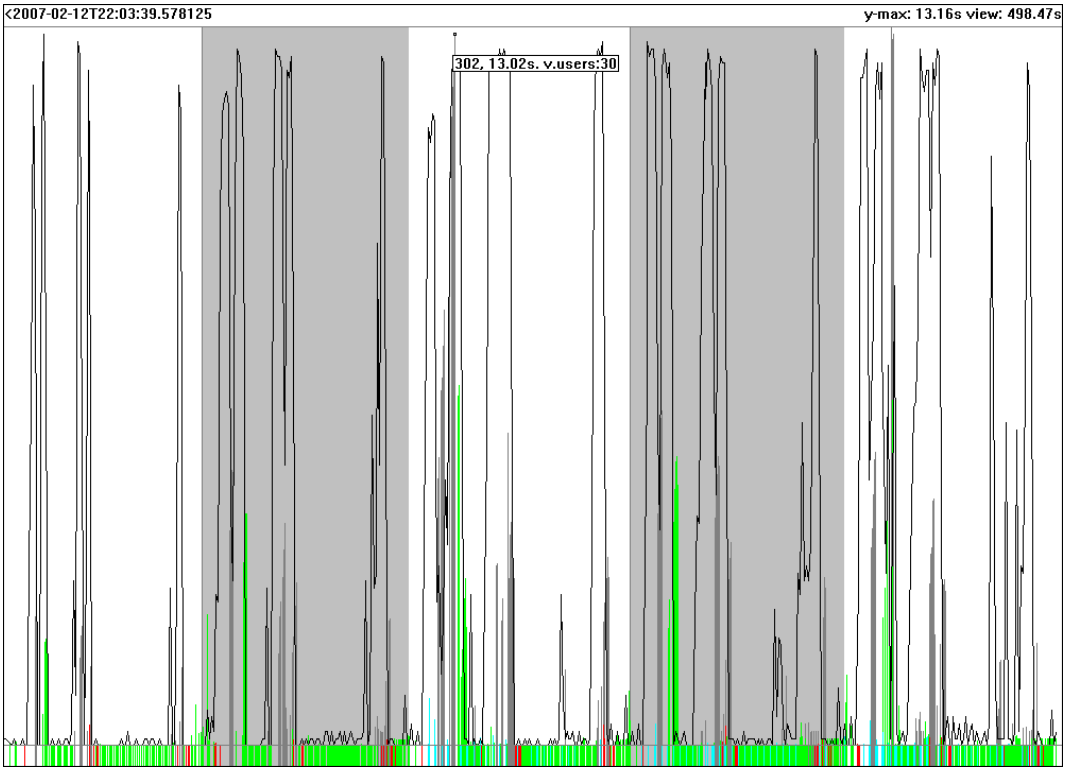


KUVIO 30. HTTP-pyyntöjen vasteajat, sekä virtuaalimuistin käyttö



KUVIO 31. HTTP-pyyntöjen vasteajat, sekä vapaan muistin määrä

Palvelimella suoritetut mittaukset osoittautuivat kyseisessä esimerkkitapauksessa hyödyllisiksi. Kaikista palvelimella suoritetuista mittauksista ei voitu kuitenkaan tehdä yhtä hyvin päätelmiä, kuten palvelimen suorittimen käyttöasteen kuvaaja (KUVIO 32) osoittaa.



KUVIO 32. HTTP-pyyntöjen vasteajat, sekä suorittimen käyttöaste

4 YHTEENVETO

Työn tavoitteena oli suorituskykytestaustyökalun suunnittelu ja toteutus. Kehitysvaiheessa alkuperäisten vaatimusten mukaisen ohjelmiston testaaminen osoittautui melko työlääksi, koska mittaustulokset piti viedä taulukkolaskentaohjelmaan, ennen kuin mittaustuloksia pystyi käytännössä tulkitsemaan. Tästä johtuen ja koska projekti oli edennyt odotettua nopeammin, päätettiin ohjelmistoon lisätä graafinen käyttöliittymä. Graafisen käyttöliittymäversion piirtämästä kuvaajasta nähtiin nopeasti mittaustulokset.

Ohjelmistoa testattaessa testauksen kohdepalvelimena toimineen tietokoneen HTTP-palvelinsovellus lakkasi vastaamasta pyyntöihin, eikä koneen SSH-palvelinsovelluskaan enää vastannut yhteyspyyntöihin. Palvelimen lokitietoja tutkimalla ongelmaksi selvisi muistin loppuminen. Testausohjelmistoon päätettiin lisätä vielä tuki vmstat-ohjelman statistiikkatiedoille, joista selviää mm. vapaan muistin määrä sekä heittovaihtotiedoston käyttö.

Yhdistämällä palvelimella suoritettujen mittausten tulokset asiakassovelluksessa suoritettuihin mittauksiin saatettiin kuvaajista hyvin havaita, kuinka palvelimen vapaan muistin määrä laski yhtäaikaisten käyttäjien lisääntyessä ja laskien pisteeseen, jossa palvelimen käyttöjärjestelmä joutui ottamaan heittovaihtotiedoston käyttöön.

Pelkillä asiakassovelluksen mittauksilla saatiin selvitettyä kuormituspiste, jossa palvelin ei enää toiminut riittävällä tasolla. Liittämällä mukaan palvelimen statistiikkaa selvitettiin myös järjestelmän pullonkaula.

Kun tämän ohjelmiston käytöstä, sekä suorituskykytestauksesta yleensä, tulee enemmän käytännön kokemusta, voidaan ohjelmistoon lisätä tarvittavia ominaisuuksia. Suorituskykytestausohjelmistolle ei nähty tässä vaiheessa tarvetta tukea hajautettua järjestelmää, jossa kohdepalvelua voitaisiin kuormittaa usealta tietokoneelta samanaikaisesti. Suorituskykytestauksessa suunnitteluvaihe sekä tulosten analysointi vievät mittausten suoritukseen nähden niin paljon enemmän aikaa, että testaussovellusta ollaan valmiita ajamaan useammalta koneelta manuaalisesti, mikäli yhden koneen suorituskyky ei riitä.

Mikäli testausohjelmistolle haluttaisiin joskus tuki hajautetulle järjestelmälle, voitaisiin se toteuttaa lisäämällä sovelluksen komentoriviversioon HTTP-yhteys keskuspalvelimelle, joka kontrolloisi testausten suoritusta sekä keräisi ja esittäisi mittaustulokset.

5 LÄHTEET

Berners-Lee, T., Fielding, R. & Masinter, L. Uniform Resource Identifier (URI): Generic Syntax [online]. Network Working Group, tammikuu 2005 [viitattu 16.12.2006]. Saatavissa: http://www.ietf.org/rfc/rfc3986.txt. STD 66

Berners-Lee, T., Masinter, R. & McCahil, L. Uniform Resource Locators (URL) [online]. Network Working Group, joulukuu 1994 [viitattu 16.12.2006]. Saatavissa: http://www.ietf.org/rfc/rfc1738.txt

Crocker, D. STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES [online]. Newark: University of Delaware, elokuu 1982 [viitattu 16.12.2006]. Saatavissa ftp://ftp.rfc-editor.org/in-notes/rfc822.txt. STD 11

Domain name system – Wikipedia [online]. A WIKIMEDIA project, tammikuu 2007 [viitattu 22.1.2007]. Saatavissa: http://en.wikipedia.org/wiki/Domain_name_system

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. Hypertext Transfer Protocol - HTTP/1.1 [online]. Network Working Group, kesäkuu 1999 [viitattu 16.12.2006]. Saatavissa: http://www.ietf.org/rfc/rfc2616.txt

Flander [online]. Flander Oy, 2006 [viitattu 9.2.2007]. Saatavissa: http://www.flander.com/fi_index.php

Freed, N. & Borenstein, N. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types [online]. Network Working Group, marraskuu 1996 [viitattu 16.12.2006]. Saatavissa: ftp://ftp.rfc-editor.org/in-notes/rfc2046.txt

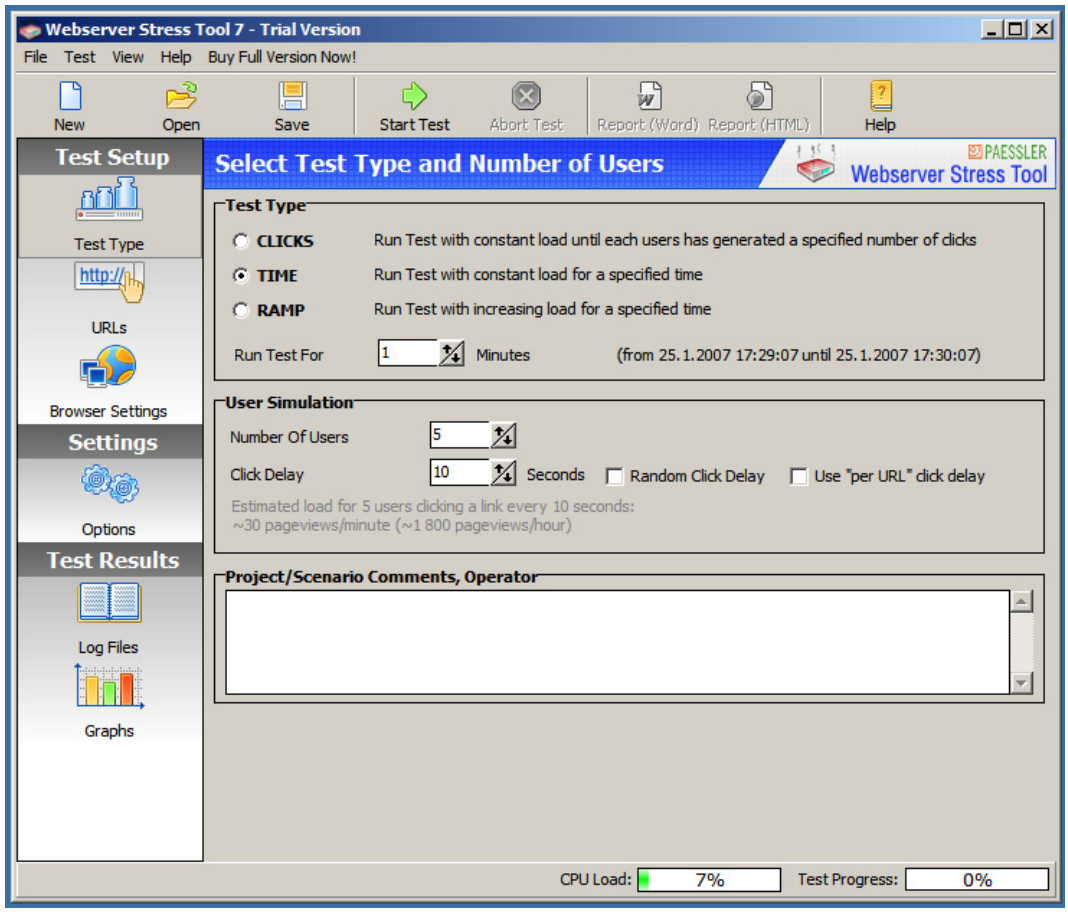
Kristol, D. & Montulli, L. HTTP State Management Mechanism [online]. Network Working Group, helmikuu 1997 [viitattu 12.2.2007]. Saatavissa: http://www.ietf.org/rfc/rfc2109.txt

Microsoft ACE Team. 2002. PERFORMANCE TESTING MICROSOFT .NET WEB APPLICATIONS. Washington: Microsoft Press.

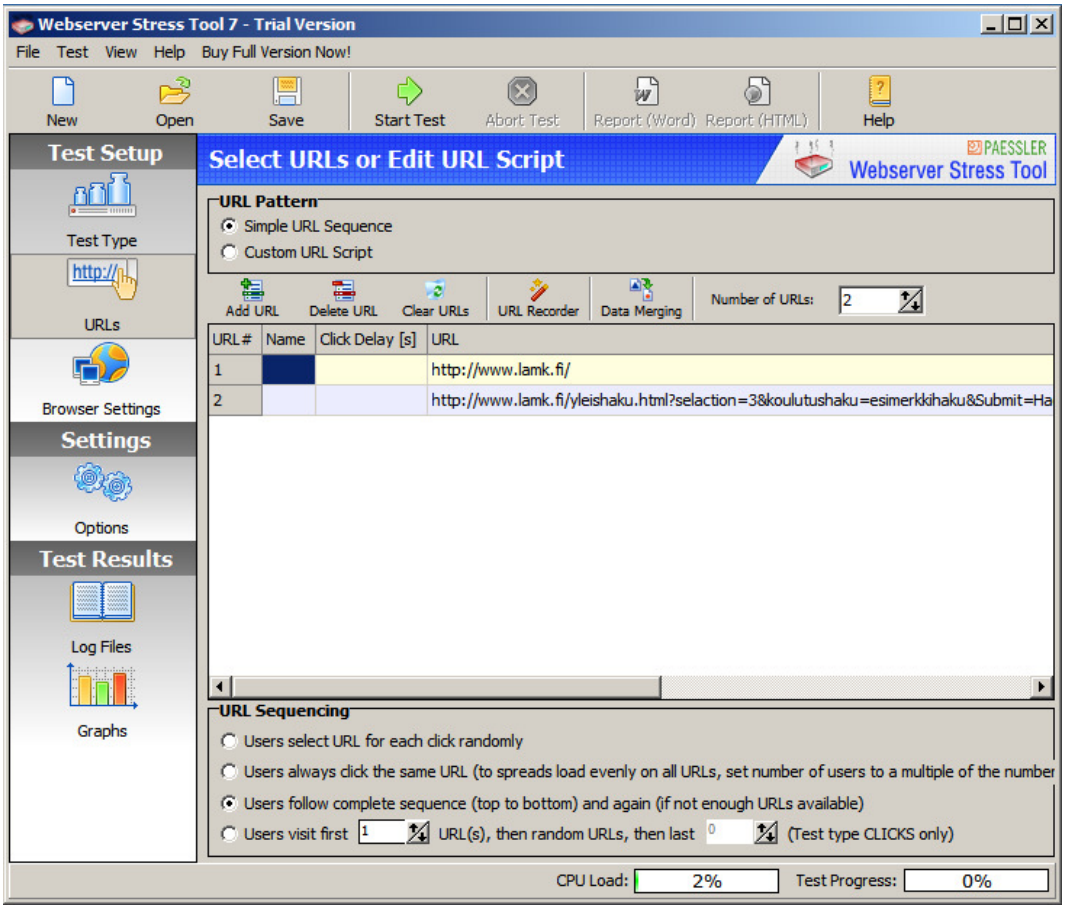
Raggett, D., Hors, A. & Jacobs, I. HTML 4.0 Specification [online]. World Wide Web Consortium, huhtikuu 1998 [viitattu 17.12.2006]. Saatavissa http://www.w3.org/TR/1998/REC-html40-19980424/html40.pdf

6 LIITTEET

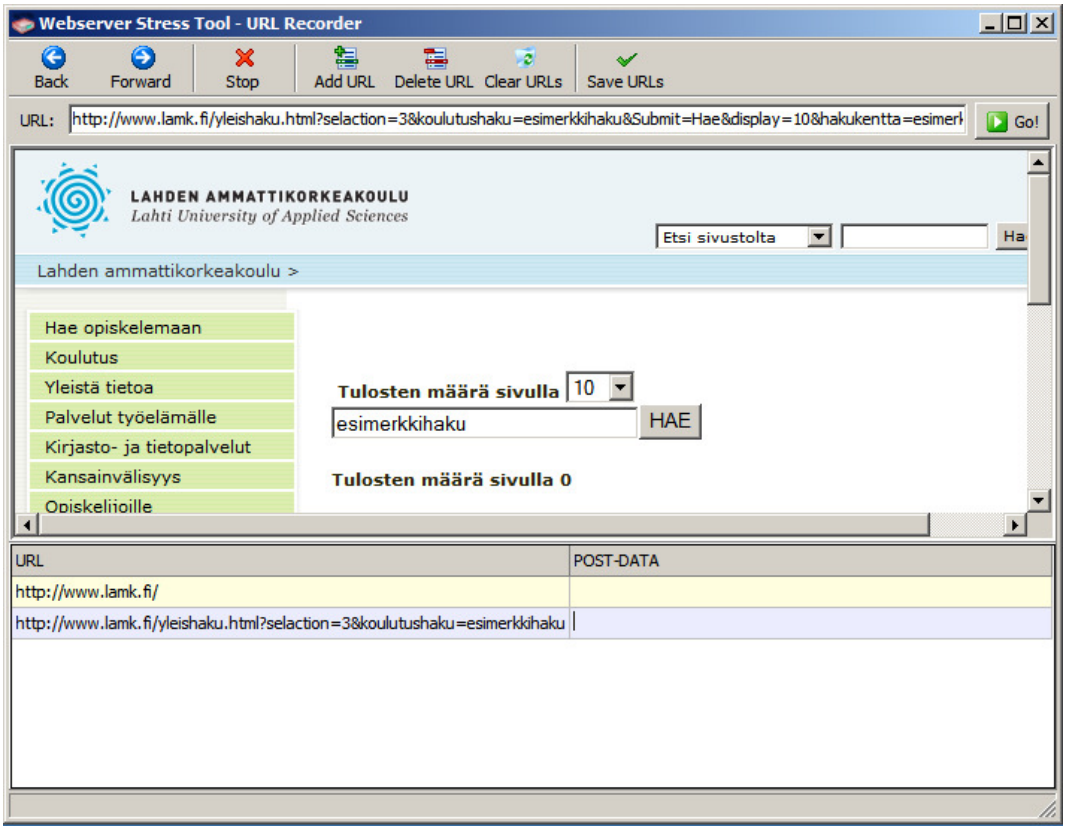
LIITE 1/1. Kuvaruutukaappauksia Webserver Stress Tool ohjelmasta



LIITE 1/2. Kuvaruutukaappauksia Webserver Stress Tool ohjelmasta



LIITE 1/3. Kuvaruutukaappauksia Webserver Stress Tool ohjelmasta



LIITE 2/1. Testisekvenssien automatisointi

7. Automating test sequences (wspt expressions)

7.1 Expressions and delimiters

Test sequences start with two delimiter definition lines beginning with ':'. First line defines starting delimiter for wspt expressions, seconds line defines ending delimiter. For example:

```
:{{{
:}}}
```

Would define {{{ as starting delimiter and }}} as ending delimiter.

In following examples (and) are used as delimiters for simplicity. In real use single character delimiters are not sufficient as they might exist in actual requests.

Wspt expressions are simple, nestable operations that can be used to automate information, such as login account for virtual users. The expressions are defined in following format:

[starting-delimiter][operation][operands][ending-delimiter]

Where operation is single ASCII character:

Operation	Operation type	Description
$	Variable / Command	Executes wspt command or returns variable value
[	TextDB content	Returns field content from specific row.
@	File content	Returns file content.

For example ($id) would return ID number (starting from 0) of virtual user. When expression returns value, it simply replaces the expression with return value. For example following line:

```
GET /userinfo.php?user=($id)&type=all
```

Would be become following for virtual user with ID value of 4:

```
GET /userinfo.php?user=4&type=all
```

Expressions can be nested, so that result of inner expressions becomes operand for outer one. Expressions are parsed starting from the most inner one:

```
(@image_($id).jpg)
```

Would be parsed (assuming virtual user id is 4) in 1st stage to:

```
(@image_4.jpg)
```

In 2nd stage the parser would load content of image_4.jpg file and replace expression with the content of that file.

LIITE 2/2. Testisekvenssien automatisointi

7.2 File content expression

File content expression is defined with @ character.

File content expression takes file path as input parameter. When wspt parser encounters file content expression, it will load file from the given path and replace the expression with the content of loaded file.

For example, If we have file named c:\example.txt that contains text 'ample', and following expression:

```
(@c:\example.txt)
```

Wspt parser would load content of c:\example.txt and replace the expression with content of that file. In this example the line would become:

```
ample
```

Wspt expressions can be nested:

```
(@c:\ex(@c:\example.txt).txt)
```

Would also be parsed to text ample. In 1st stage the parser loads content for the inner expression:

```
(@c:\ex(@c:\example.txt).txt)
```

Resulting to following:

```
(@c:\example.txt)
```

That would result to:

```
ample
```

Wspt has file cache for file contents. When multiple virtual users load the same file, it is loaded to memory only once.

LIITE 2/3. Testisekvenssien automatisointi

7.3 TextDB expression

TextDB expressions is defined with [character. TextDB expression takes operand that contains field name row number separated with ':' character, for example:

```
([Username:2)
```

Where 'Username' is name of the field, and 2 is row number (3th row, starting from zero index).

When wspt parser encounters TextDB expression, it will replace the expression with content from TextDB. TextDB is text file. TextDB file is defined when running test sequences (see chapter: Using the GUI version).

TextDB file uses following format:

[field name]: [field content]
[field name]: [field content]
[row separator]
[field name]: [field content]
[field name]: [field content]
[row separator]
[field name]: [field content]

Where row sepator is %%. Example TextDB content:

```
Username: James Bond
Password: 007
%%
Username: Austin Powers
Password: Mojo
%%
Username: Maxwell Smart
Password: CONTROL
```

With TextDB shown above:

```
([Username:2)
```

Would be replaced with:

```
Maxwell Smart
```

As with all wspt expressions, other expressions can be used as operands for TextDB expression:

```
([Password:($id))
```

Would be replaced with

```
James Bond
```

Assuming that the virtual user id is 0.

LIITE 2/4. Testisekvenssien automatisointi

7.4 Variable / Command expression

Variable / Command expression is defined with $ charater. Commands take zero or more arguments separated with ':' character.

id

Id returns ID of current virtual user.

virtualusers

Virtualusers command returns number of concurrent virtual users in current test phase.

Responsecode

Responsecode variable holds response code of last executed HTTP request.

requestnumber

Requestnumber variable holds request number of current HTTP request. This number is index of request in test sequence (First HTTP request in test sequence has index 0).

null

Null is special variable that returns always emptry string. Null variable can be used as destination variable for expressions such as set. Null variable can be used in situations where variable value is read conditionally (getting value of null variable instead of the real one). Wspt parser throws error if unknown variable is accessed to prevent errors caused by mistyped variable names.

Following example returns empty string:

```
$(set:null:999)($null)
```

zero

Zero is special variable that returns always 0. See null for the reasoning for this variable.

set

Set command sets variable content. Variable name can be empty, in which case command is ignored.
Parameters :

```
variable:value
```

Following example sets value www.example .com for variable named url:

```
($set:url:www.example.com)
```

setonce

Setonce command sets variable content if it is not already set. Parameters are same as with set command.

if

uf compares two string values and returns the first return option if the strings are equal, otherwise the seconds return option is returned.
Parameters:

```
string1:string2:return1:return2
```

Following example returns string "example":

LIITE 2/5. Testisekvenssien automatisointi


```
($if:james:bond:not this:example)
```

In real application, atleast one of the strings compared would be taken from other expression.

cmp

cmp compares two integer values and returns the first return option if they are equal, otherwise the second return option is returned.
Parameters:

```
value1:value2:return1:return2
```

Following example returns value 9:

```
($cmp:1:2:0:9)
```

In real application, atleast one of the values compared would be taken from other expression. Following example returns value 1 if id of current virtual user is 0, otherwise -1 is returned.

```
($cmp:$(id):0:1:-1)
```

add

If command adds integer value to variable. Variable name can be empty, in which case command is ignored. Variable must exist (use set command) before it can be used with add command.
Parameters:

```
variable:value
```

Following example adds value 5 to variable named counter:

```
($add:counter:5)
```

goto

Alters current test sequence request number. This command can be used for conditional execution of test requests when compined with other expressions (if, cmp).
Parameters:

```
position
```

If position contains + or – sign, then it is handled as relative offset. Following example would move test sequence to previous request:

```
($goto:-1)
```

Goto command should be only used in special locations used for hanlding test sequence logic. These locations are used in place where HTTP requests fields are defined in test sequence.

Commands executed in wspt-request field are executed before request is sent to server.

Commands executed in wspt-response field are executed after receiving response from server. When wspt-response fields are handled, requestnumber variable is already updated to point to next request. Using ($goto:-1) expression in this field would move requestnumber back to the same request causing infinite loop.

Following example test sequence loops 1+999 times if servers keeps returning status code 200.

LIITE 2/6. Testisekvenssien automatisointi


```
http://www.example.com

GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; fi; rv:1.8.1)
Accept: text/xml
Accept-Language: fi,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
wspt-response: ($setonce:counter1:($if:($responsecode):200:999:0))
wspt-response: ($goto:($if:($counter1):0:+0:-1))
wspt-response: ($add:counter1:-1)

------------------------------------------------------------
```

Wspt parses expressions starting from the most inner one, so this is how the sequence would be parsed if server keeps returning status code 200:

```
wspt-response: ($setonce:counter1:($if:($responsecode):200:999:0))
wspt-response: ($setonce:counter1:($if:200:200:999:0))
wspt-response: ($setonce:counter1:999)
```

At this point counter1 holds value 999.

```
wspt-response: ($goto:($if:($counter1):0:+0:-1))
wspt-response: ($goto:($if:999:0:+0:-1))
wspt-response: ($goto:-1)
```

requestnumber is set to back to 0. (It is increased by 1 before wspt-reponse fields are parsed).

```
wspt-response: ($add:counter1:-1)
```

counter1 = counter1 + -1, so counter1 is now 998. After sequence is parsed, wspt executes request pointed by requestnumber, in this example the same one is executed again. After request is sent and response received wspt parses wspt-response fields again:

```
wspt-response: ($setonce:counter1:($if:($responsecode):200:999:0))
wspt-response: ($setonce:counter1:($if:200:200:999:0))
wspt-response: ($setonce:counter1:999)
```

Next wspt-response field decreases the counter..

```
wspt-response: ($goto:($if:($counter1):0:+0:-1))
wspt-response: ($goto:($if:998:0:+0:-1))
wspt-response: ($goto:-1)
```

After many iterations, when counter 1 reaches zero:

```
wspt-response: ($goto:($if:($counter1):0:+0:-1))
wspt-response: ($goto:($if:0:0:+0:-1))
wspt-response: ($goto:+0)
```

goto method adds 0 to requestnumber variable, so test sequence continues to next request. NOTE ($goto:0) would not do the same as it would SET the requestnumber to 0, not alter it by +0.