RIA-TEKNOLOGIAT JA WEB-KEHITYS

Timo Paananen

Opinnäytetyö
Marraskuu 2009

Tietojenkäsittely
Luonnontieteiden ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
KUVAILULEHTI

Tekijä(t)	Julkaisun laji	Päivämäärä
PAANANEN, Timo	Opinnäytetyö	03.12.2009
	Sivumäärä 83	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
RIA-TEKNOLOGIAT JA WEB-KEHITYS

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Työn ohjaaja(t)
TUIKKA, Tommi

Toimeksiantaja(t)
Codecenter Oy

Tiivistelmä
Opinnäytetyön tarkoituksena oli tutkia Internet sivujen ulkoasuun ja käytettävyyteen vaikuttavien teknologioiden kehittymistä 2000-luvulla sekä perehtyä neljään eri RIA-teknologiaan tarkemmin. Näistä neljästä oli tavoitteena löytää paras mahdollinen teknologia Codecenter Oy:n kehittämän Webtrix-julkaisujärjestelmän käyttöliittymän kehittämistä varten. RIA-teknologioiksi valittiin Google Web Toolkit, Echo Framework, Java2Script ja Adobe Flex.

Tutkimus aloitettiin perehtymällä CSS:n, JavaScriptin, Ajaxin ja RIA-teknologioiden historiaan 2000-luvun alusta tähän päivään asti. Tästä jatkettiin RIA-teknologioiden tuomien hyötyjen ja haittojen kartoittamiseen sekä mahdollisten ongelmakohtien selvitykseen. Lopuksi valituilla teknologioilla toteutettiin kuvagallerian käyttöliittymä valmiina olevan bisneslogiikkaprojektin päälle. Toteutuksella pyrittiin selvittämään tarkemmin mahdollisia ongelmakohtia otettaessa teknologioita käyttöön valmiiseen projektiin. Toteutetuille käyttöliittymille suoritettiin suorituskykytestaus Jmeter-sovelluksella, jotta saataisiin selville eri teknologioiden kuorman kesto sekä viitteitä tehtyjen palvelinpyyntöjen määrästä.

Työn tulokset pyrittiin esittämään siten, että niitä pystyisivät myös muut tahot hyödyntämään RIA-teknologiavalintaa tehdessä sekä teknologiaa käyttöön ottaessaan.

Avainsanat (asiasanat)
Adobe Flex, Echo Framework, GWT, Java, Java2Script, JSF, RIA

Muut tiedot

22 liitesivua

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of Publication	Päivämäärä
PAANANEN, Timo	Bachelor´s Thesis	03.12.2009
	Pages 83	Language Finnish
	Confidential ( ) Until	Permission for web publication (X)

Title
RIA-TECHNOLOGIES AND WEB-DEVELOPMENT

Degree Programme
Degree Programme in Business Information Systems

Tutor(s)
TUIKKA, Tommi

Assigned by
Codecenter Co

Abstract
The purpose of this Bachelor's thesis was to examine development of technologies that effect the outfit and usability of webpages in 21st century and take a deeper look at four RIAtechnologies. The objective was to find the best technology of these four for Codecenter Co's Webtrix content management system's user interface development. The selected technologies were Google Web Toolkit, Echo Framework, Java2Sscript and Adobe Flex.

The research was started with the history of CSS, JavaScript, Ajax and RIAtechnologies from the beginning of the 21st century to the present day. From there the research moved to examine pros, cons and potential problem points of RIAtechnologies. The last phase in research was the implementation of image gallery with each technology. The point of the implementation was to find more specific problems in integration with an already existing project. After the implementation of user interfaces performance tests were executed for all implemented image galleries with Jmeterapplication. This revealed how different technologies survived on heavy load and how many server calls they made.

The results of this bachelor's thesis are presented in such a way that they can be used by third parties when they are choosing RIAtechnology or initializing a project with some of the selected technologies.

Keywords
Adobe Flex, Echo Framework, GWT, Java, Java2Script, JSF, RIA

Miscellaneous
22 appendixes

SISÄLTÖ

KUVIOT

TERMISTÖ

Adobe Flex	Adoben kehittämä sovelluskehys RIA-ohjelmien kehittämiseen
AJAX	Asynchronous JavaScript and XML, teknologiaryhmä jossa yhdistyy Javascripti ja XML. AJAX:n avulla voidaan suorittaa palvelinpyyntöjä ja käyttöliittymän päivitystä ilman koko sivun uudelleen latausta.
Echo Framework	NextApp:n kehittämä sovelluskehys, mikä on tarkoitettu RIA -ohjelmien tekemiseen.
GWT	Google Web Toolkit on käyttöliittymien toteutukseen suunniteltu sovelluskehys, jonka mukana toimitetaan oma kääntäjä, jolla voidaan kääntää Java-koodia javascript-koodiksi sekä työkalut, joilla voidaan luoda projektille runko.
Java2Script	Java2Javascript Pacemaker, Eclipsen laajennus, jolla pystyy kääntämään RCP (Rich Client Platform) ohjelmia RIA -ohjelmiksi.
Java	Olio-ohjelmointikieli
Java EE	Java Enterprise Edition eli webkehitykseen tarkoitettu Java teknologia.
JSF	JavaServer Faces, web-sovelluskehys, mikä on tarkoitettu käyttöliittymien toteutukseen
SWING	Sun Microsystems:n tarjoama käyttöliittymien kehitykseen tarkoitettu kirjasto, mikä tulee Javan mukana.
SWT	The Standard Widget Toolkit, Java-pohjainen käyttöliittymien kehitykseen tarkoitettu kirjasto.

1 INTERNET 2000-LUVULLA

Internet juontaa juurensa jo 50-luvulta, jolloin aloitettiin kehittämään teknologioita, jotka lopulta johtivat nyky-internetin kehittymiseen. Vaikka internetin juuret ovat syvällä historiassa, sallittiin voittoa tavoittelemattomille tahoille siihen pääsy vasta vuonna 1994, jolloin siinä käytetyt perusteknologiat olivat kehittyneet pohjimmiltaan nykyiselle tasolle. Nykyisin internet on tuttu suurimmalle osalle ihmisistä. Maapallon väestöstä 1.1 miljardilla ihmisellä on mahdollista päästä käyttämään internetin tarjoamia palveluita säännöllisesti ja jokainen näistä käyttäjistä pystyy nykypäivänä tuottamaan sisältöä internetiin. 1990-luvun loppupuolella sekä 2000-luvulla kehitys on keskittynyt enemmän sisällöntuottamiseen sekä sisällön näyttämiseen liittyviin teknologioihin.

2000-luvulla päätään ovat nostaneet RIA-teknologiat (Rich Internet Application), joiden avulla voidaan toteuttaa dynaamisempia ja käyttäjäystävällisempiä käyttöliittymiä web-palveluille. RIA-teknologiat perustuvat selaimessa suoritettavaan koodiin, jota voidaan suorittaa muulloinkin kuin sivun uudelleen päivityksen yhteydessä. Tiedon tallennukseen ja noutamiseen palvelimelta RIA-teknologiat käyttävät AJAX:ia (Asynchrous Javascript And Xml). Joka mahdollistaa että taustalla voidaan suorittaa palvelinpyyntöjä ja päivittää tarvittavat sivun osat saadun vastauksen perusteella.

Tässä opinnäytetyössä keskitytään selvittämään 2000-luvun internetkäyttöliittymien kehitystrendejä sekä vertailemaan erilaisia RIA-teknologioita keskenään sekä vertaamaan niitä perinteiseen Java EE -käyttöliittymäteknologiaan JSF (JavaServer Faces).

Paremman käsityksen saamiseksi RIA-teknologioiden eroista opinnäytetyössä esitellään koodi-esimerkkejä eri teknologioilla toteutetuista käyttöliittymistä, jotka pyritetään tekemään mahdollisimman saman kaltaisiksi toiminnallisuuksiltaan. Myös teknologioiden kuorman kestävyyttä selventämään laaditaan suorituskykytestejä, jotka suoritetaan erikseen jokaiselle teknologialle.

2 TUTKIMUSASETELMA

Tässä luvussa esitellään opinnäytetyön toimeksiantaja. Jäljempänä esitellään tutkimuksen tavoitteet sekä käsitellään aihealuetta rajoittavia tekijöitä. Luvun lopussa määritellään tutkimuskysymykset, jotka ohjaavat tutkimusta sekä esitellään käytetyt tutkimusmenetelmät.

2.1 Toimeksiantaja

Opinnäytetyön toimeksiantajana toimii vuonna 2003 toimintansa aloittanut Jyväskyläläinen Codecenter Oy. Codecenter Oy on nykyaikaisiin verkkoteknologioihin erikoistunut asiantuntijayritys, minkä erityisosaamisalueena on Java EE -teknologiat, kuten Spring Framework, ja niihin liittyvät projektityöt, koulutukset sekä konsultointi. Codecenterillä on yli kymmenen vuoden vankka kokemus Java-teknologioista niin koulutuksessa kuin projektitöissä sekä webteknologioista. Asiakaslähtöisten projektitöiden ja koulutuksien lisäksi Codecenter Oy tekee omaa tuotekehitystä, jonka seurauksena on syntynyt esimerkiksi Codecenter Webtrix -julkaisujärjestelmä sekä tuntikirjapitoon erikoistunut Codecenter Momento järjestelmä.

2.2 Tutkimusasetelma

2.2.1 Tutkimuksen tavoitteet

Codecenter Webtrix -julkaisujärjestelmä on kypsynyt jo neljänteen versioonsa ja on ominaisuuksien puolesta erittäin kattava. Se soveltuukin sekä websivujen kehittäjien työkaluksi että sivuston sisältöä ylläpitävien henkilöiden työkaluksi. Tämä tutkimuksen tavoite on selvittää mikä teknologia sopisi parhaiten sen käyttöliittymän kehitykseen, jotta siitä saataisiin dynaamisempi ja entistä käyttäjäystävällisempi. Tutkimuksessa vertaillaan eri RIA-teknologioita keskenään sekä RIA-teknologioita perinteisesti käytettyihin käyttöliittymän rakentamiseen tarkoitettuihin teknologioihin.

RIA-teknologioiksi tutkimuksessa on valittu teknologioita, joilla voidaan toteuttaa toiminnallisuuksia ilman sivun uudelleen latausta. Valitut teknologiat ovat Googlen kehittämä Google Web Toolkit (GWT), NextApp:n kehittämä Echo Framework, The Standart Widget Toolkit:iin (SWT) perustuva Java2Script Pacemaker (J2S) sekä Adobe Macromedian kehittämä Adobe Flex. Vertailun pohjalta on tarkoitus valita parhaiten Codecenter Webtrixin käyttöliittymän kehitykseen sopiva teknologia.

Tutkimuksesta joudutaan rajaamaan pois teknologiat, joilla tehdään perinteisiä käyttöliittymiä kuten SWING ja SWT, koska Codecenter Webtrix-julkaisujärjestelmä toimii täysin webselain pohjaisesti. Järjestelmän toimiessa HTTP-protokollan päällä asettaa se myös muita rajoituksia käyttöliittymälle, kuten sen ettei tiedon tallennus välttämättä ole aina mahdollista.

2.3 Tutkimuskysymykset

Tutkimuskysymykset ovat pyritty asettamaan siten, että niiden avulla pystyttäisiin selvittämään mahdollisimman hyvin paras vaihtoehto Webtrix-julkaisujärjestelmän käyttöliittymän kehittämiseen. Asetetut tukimuskysymykset ovat seuraavat:

1. Mitä ovat 2000-luvulla vaikuttaneet internetkäyttöliittymien kehitystrendit.

2. Mitä etuja tai haittoja RIA-teknologioissa ja käyttöliittymissä on perinteisiin käyttöliittymiin nähden

3. Miten valitut RIA-teknologiat eroavat toisistaan ja miten erot vaikuttavat käyttöliittymien kehitykseen.

Internetkäyttöliittymien kehitystrendejä tutkittaessa perehdytään siihen millä tasolla Internetkäyttöliittymäteknologiat olivat vuosituhannen alussa ja mihin ne ovat nykypäivään mennessä kehittyneet.

Tutkittaessa RIA-teknologioiden eroja ja niiden vaikutuksia käyttöliittymien kehitykseen kiinnitetään huomiota siihen miten eri teknologioilla toteutetut ratkaisut kestävät

rasitusta. Tutkimuksessa huomioidaan myös miten toteutetut käyttöliittymät voidaan integroida taustalla toimivaan valmiiseen järjestelmään ja mitä mahdollisia rajoituksia teknologia asettaa käyttöliittymää kehitettäessä.

2.4 RIA-teknologiat

Rich Internet Application teknologioilla tarkoitetaan teknologioita, joilla pystytään toteuttamaan internet sivustoja joissa käyttäjä pystyy vaikuttamaan sisällön esitystapaan dynaamisesti ilman sivun uudelleen latausta. RIA-teknologioiden tulee siis erotella käyttöliittymä ja siihen liittyvät toiminnot asiakaspuolelle. Se jättää dataoperaatiot ja datamuokkauksen palvelinpuolen hoidettavaksi. Adobe Macromedia on määritellyt, että RIA-teknologioiden tulee täyttää seuraavat vaatimukset.

- ***Tarjota tehokas sekä korkea suorituskykyinen ajonaikainen ympäristö koodin ajamiselle, sisällölle sekä kommunikoinnille.*** *Käyttäjän ei tulisi kärsiä HTML-pohjaisten web sovellusten tehokkuusongelmista, jotka johtuvat sivupyyntö/vastaus-renderöinti mallista; tarpeesta dynaamisesti tuottaa suuri määrä tekstiä vähäisen datan lähetystä varten; asiakasohjelman puoleisen datan säilönnän puutteesta; puutteesta pystyä helposti käyttämään bisneslogiikkaa ja jopa HTML:n grafiikka modelista. Teknologian tulisi parantaa näitä kaikkia.*

- ***Integroida sisältö, kommunikointi sekä järjestelmän rajapinnat samaan ympäristöön****. Käyttäjän kokemus nykypäivän internetissä on sirpaloitunut HTML selaimeen tekstisisällölle ja perusohjelmistojen rajapinnoille; monia viestintäohjelmia kommunikaatiotoiminnoille; sekä useita mediasoittimia audio, video ja muiden mediaformaattien toistolle. Rich clientien täytyy tarjota hyvä integraatio kaikille näiden tyyppisille toiminnoille.*

- ***Tarjota tehokas ja laajennettavissa oleva oliomalli interaktiivisuudelle****. Internet selaimet ovat kehittäneet tukeaan interaktiivisuudelleen DOM:n (Document Object Model), Javascriptin ja DHTML kautta niissä ei silti ole sitä, mitä tarvitaan kunnollisten sovelluksien kehitykseen. Rich clientien täytyy tarjota tehokas, oliopohjainen malli sovelluksille ja tapahtumille. Tämän yleisen oliomallin tulee integroida käyttöliittymä, kommunikointi ja järjestelmätason palvelut.*

- ***Mahdollistaa nopea järjestelmän kehitettävyys komponenttien ja koodin uudelleen käytettävyyden kautta****. Rich clientien tulisi tukea tehokasta komponenttisuuntaista kehitystä, mahdollistaen kolmansien osapuolien ja yrityksien kehittäjille helppo uudelleen käytettävyys visuaalisille kompo-*

nenteille ja antaa nuoremmille kehittäjille pääsy monimutkaisiin toimintoihin. Näiden komponenttien tulisi integroitua saumattomasti suunnitteluaikaiseen ympäristöön kehitystä helpottamaan.

- ***Mahdollistaa sovelluspalvelimien tarjoaminen web- ja datapalveluiden käyttö**. Rich clientien lupauksen sisältävät mahdollisuuden siististi eroteltuihin näyttökerrokseen ja käyttöliittymään logiikan sijaitessa internetissä. Rikkaitten asiakasohjelmien tulisi tarjota malli jolla voidaan helposti käyttää palvelinkomponenttien tarjoamia etäpalveluita olivatpa ne sovelluspalvelimella tai XML- web- palveluita.*

- ***Mahdollistaa asiakasohjelman toimiminen online ja offline tiloissa**. Vaikka moni käyttäjä on tottunut olemaan yhdistettynä internetiin ja käyttämään internet-selainta töissä tosiasia on, että moni sovellus hyötyisi siitä että ne pystyisivät toimimaan offline tilassa laitteilla, jotka saavat internet yhteyden vain toisinaan, kuten PDA (Personal Digital Assistent) laitteilla ja kannettavilla tietokoneilla. Kuten monet sovellukset vaativat tuen jatkuvalle yhteydelle kaksisuuntaisella, ilmoituspohjaisille kommunikaatiolle. Rikkaat asiakasohjelmien täytyy mahdollistaa kummankin tyyppisten sovelluksien kehittäminen ja julkaisu.*

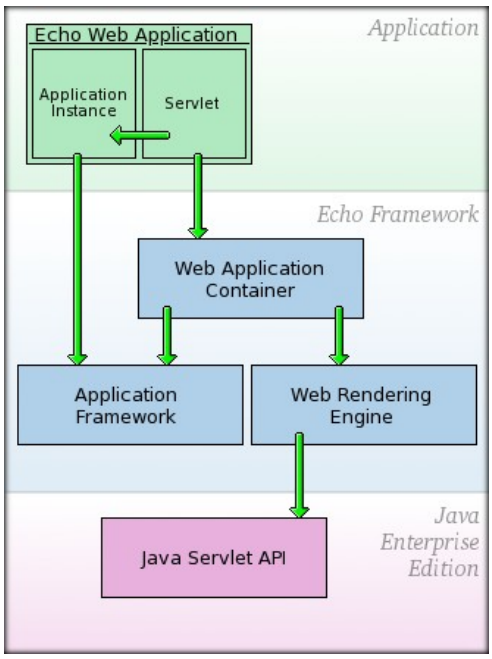
- ***Mahdollistaa järjestelmän helppo julkaiseminen eri alustoille sekä laitteille**. Internet ohjelmissa on kyse tavoitettavuudesta. Internetin lupaus on yksi sisältö, ohjelmat missä tahansa, alustasta ja laitteesta riippumatta. Rikkaitten asiakassovelluksien täytyy tukea kaikki suosittuja työpöytä käyttöjärjestelmiä sekä suurinta osaa sulautetuista järjestelmistä kuten älypuhelimia, PDA:a, Set-top-yksiköitä, pelikonsoleita ja internet-laitteita.*

2.4.1 Echo Framework

Echo Framework on NextApp-nimisen yrityksen kehittämä sovelluskehys, mikä on julkaistu avoimen lähdekoodin Mozilla public -lisenssin alaisuudessa. Echo on tarkoitettu niiden RIA-sovellusten toteuttamiseen, jotka muistuttavat työpöytäsovelluksia. Kehittäjän näkökulmasta katsottuna Echo on kuin mikä tahansa muu käyttöliittymäkirjasto kuten SWING tai SWT. (Echo Framework 2007.) Echo on kehittynyt tätä opinnäytetyötä kirjoittaessa 3.0 beta 6 versioon asti, joka on samalla tässä opinnäytetyössä käytettävä versio.

Echo on jaettu kolmeen moduuliin: sovelluskehykseen, renderöintimoottoriin sekä web-sovellussäiliöön. Toimiakseen Echo vaatii Java Server säiliön, joka toteuttaa 2.3 Servlet määrityksen sekä vähintään version 1.4 Java virtuaalikoneesta. Echo ei vaadi käyttöliittymäkoodin kääntämistä JavaScriptiksi, kuten Googlen tarjoama GWT-tek-

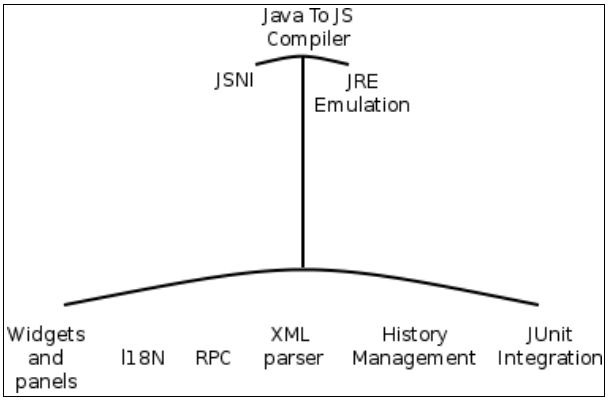
nologiassa vaatii, vaan Echo:lla toteutettuja ohjelmia ajetaan sen oman web-sovellussäiliön päällä. Sovellus käyttää AJAX:ia palvelimen ja asiakasohjelman väliseen tiedonsiirtoon, kuvio 1 selkeyttää Echon rakennetta.



KUVIO 1. Echo sovelluksen rakenne (Echo Sovelluskehyksen toimintaperiaate 2007.)

2.4.2 Google Web ToolKit

GWT (Google Web Toolkit) on Googlen kehittämä RIA-teknologia, mikä kääntää Java-koodin selaimessa suoritettavaksi Javascript-koodiksi. Google julkaisi GWT:n Toukokuussa 2006 ja on lisensoinut sen Apache 2.0 -linsenssin alaisuuteen. GWT:n mukana tulee muun muassa kääntäjä, kevyt internetselain, debug-konsoli, lokalisointityökalu, integraatio-työkalu Eclipse kehitysympäristölle sekä työkalu, jolla voidaan luoda helposti uudelle projektille runko. Kuvio 2, esittelee kokonaisuudessa mitä kaikkia työkaluja GWT:n mukana saadaan käyttöön.



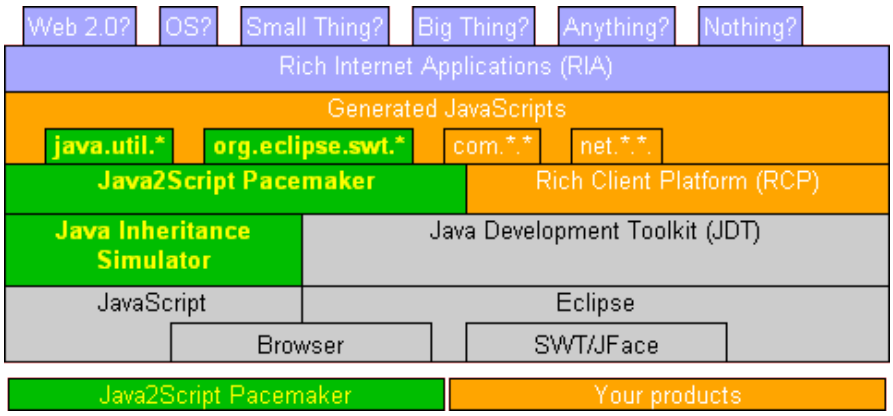
KUVIO 2. GWT:n rakenne (Hanson & Tacy 2007).

GWT vaatii toimiakseen Java SDK:n (Software Development Kit). Googlen tarjoama kääntäjä käyttää Googlen itse kehittämäänsä JRE (Java Runtime Environment) emulaatiokirjastoa Java-koodin kääntämiseen Javascriptiksi, jonka luvataan olevan yhteensopiva Internet Explorer, Firefox-, Mozilla-, Safari- sekä Opera-selainten kanssa. Emulaatiokirjaston puutteiden takia aivan kaikki Javan tarjoamat ominaisuudet eivät ole käytettävissä. Versio 1.5 GWT:stä paransi tilannetta huomattavasti tuodessaan tuen suurelle osalle Java 5 ominaisuuksia. Uusin versio, 1.6, tuo mukanaan myös paljon muutoksia GWT-projekteihin sillä se korvaa GWTShell- ja GWTCompiler-työkalut uusilla sekä muuttaa koko projektirungon rakennetta entistä vapaammaksi sekä laajentaa emulaatiokirjastoa.

2.4.3 Java2Script Pacemaker

Javas2Script Pacemaker on avoimen lähdekoodin yhteisön kehittämä Eclipsen plugini, joka tarjoaa Java-JavaScript kääntäjän sekä oliopohjaisen Javascript-emulaatiokirjaston, joka on periaatteeltaan samanlainen kuin GWT:n JRE-emulaatiokirjasto. Sen ensimmäinen virallinen versio julkaistiin Lokakuussa 2007 ja sen viimeisin versio 2.0 julkaistiin noin vuotta myöhemmin Marraskuussa 2008. Versiota 2.0 on tullaan käyttämään tässä opinnäytetyössä. Vaikka J2S muistuttaa paljon GWT:tä ja ne jakavat paljon samanlaisia ideoita, niillä ei kuitenkaan ole mitään yhteistä projekteina vaan molemmat projektit ovat täysin itsenäisiä ja toisistaan riippumattomia.

J2S hyödyntää SWT-käyttöliittymäkomponenttikirjastoa, jonka se kääntää selaimessa ajettavaksi JavaScript-koodiksi. Kuvio 3 selkeyttää J2S:n rakennetta. J2S-sovelluksia kehitettäessä voidaan hyödyntää Eclipsen SWT-käyttöliittymä kehitystyökalua, jolla voidaan luoda käyttöliittymiä raahaamalla komponentteja editoriin. J2S:llä kehitettyjä sovelluksia pystytään ajamaan kehityksen aikana suoraan Eclipsen omassa internet selaimessa J2S pluginin avulla tai vastaavasti pluginin antaman osoitteen avulla sovellus voidaan suorittaa tavallisessa selaimessa.



KUVIO 3. Java2Script RPC to RIA arkkitehtuurirakenne (Java2Script – yleiskuva 2008).

J2S:n kehittäjät kuvailevat sen etuja seuraavasti heidän kotisivuillaan. Java2Script Pacemaker ei vaadi ylimääräisten API:en opettelua. Kaikki mitä sinun tarvitsee tehdä on kehittää Java sovelluksesi tavalliseen tapaan Eclipsessä. Kehittäessäsi ”Java sovellusta” voit debugata lähdekoodia rivi riviltä käyttäen Eclipsen debug-työkalua ja voi suunnitella SWT dialogisi SWT UI designerillä. (Java2Script – yleiskuva 2008.)

2.4.4 Adobe Flex

Adobe Flex on Adobe Systemsin kehittämä avoimen lähdekoodin sovelluskehys. Jolla on tarkoitus tehdä Flash tiedostoja, jotka toimivat RIA-sovelluksina. Adobe Flex julkaistiin maaliskuussa 2004 ja sen uusin versio 3.2 julkaistiin marraskuussa 2008. Tässä opinnäytetyössä käytettävä versio on 3.2.

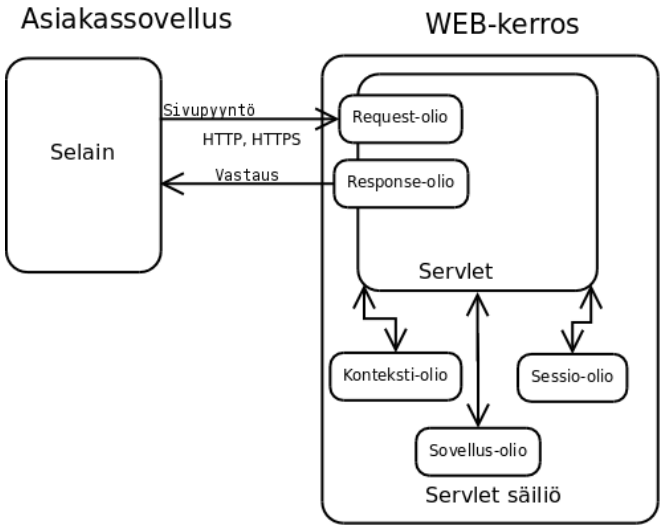
Flexillä toteutettuja RIA-sovelluksia ajetaan selaimessa Adobe Flash Playerillä kun

taas työpöydällä Adobe Airilla. Flexin mukana toimitetaan SDK, Java EE-integraatiosovellus nimeltä Flex Data Services sekä laajan UI-komponenttikirjasto, jonka komponentteja on mahdollista itse laajentaa. Tarjolla on myös maksullinen Adobe Flex Builder, joka on Eclipse pohjainen kehitysympäristö. Se on suunniteltu helpottamaan ja nopeuttamaan Flex-sovellusten kehitystä. Flex sovelluksien käyttöliittymä kehitetään MXML-kielellä, joka on declarative-XML pohjainen kieli. Asiakasohjelman logiikka taas toteutetaan oliopohjaisella ActionScript 3 kielellä.

MXML on kieli, jota ohjelmistokehittäjät käyttävät käyttöliittymän asetelmoinnin määrittelyssä, ulkoasun toteutuksessa sekä käyttäytymisen määrittelemiseen. ActionScript 3 on olio-pohjainen kieli, joka pohjautuu yritys-standardiin ECMAScript, jota käytettään määrittelemään sovelluksen logiikka. Tuotetut MXML ja ActionScript tiedostot käännetään yhdessä yhdeksi SWF-tiedostoksi, joka on itse Flex-sovellus.

2.5 Perinteiset Java-pohjaiset web-ratkaisut

Kaikki Javalla toteutetut internet-sovellukset pohjautuvat Java EE -teknologiaan. Java EE tarjoaa kaikki Java SE:n (Java Standard Edition) ominaisuudet sekä laajentaa niitä web-kehitykseen suunnitelluilla ominaisuuksilla. Java EE-sovellukset pohjautuvat servletteihin ja JSP (JavaServer Pages) teknologioihin, joista servletit hoitavat sivupyyntöjen käsittelyn ja JSP-sivut tiedon näyttämisen. JSP-sivuja ajetaan Java EE:lle tarkoitetussa säilössä. Säiliö hoitaa servlettien keskustelun asiakasohjelman kanssa ja tarjoaa pääsyyn moniin Javan palveluihin. Yleensä asiakasohjelman ja säiliön välinen keskustelu tapahtuu HTTP tai HTTPS protokollien kautta pyyntö/vastausmenetelmää käyttäen (ks. kuvio 4). Servleteillä on myös pääsy Sessio-olioon, jonka avulla servlet tietää missä tilassa asiakasohjelma kyseisellä hetkellä on. Servlet lähettää pyyntöön vastaukseksi Response-olion, joka pitää sisällään kaiken pyydetyn datan. Servlet voi ohjata sivupyynnön myös suoraan tavalliselle HTML (Hyper Text Markup Language) sivulle, jolloin menetetään mahdollisuus hyödyntää JSP-sivun dynaamista sisältöä.



KUVIO 4. Servletin ja asiakasohjelman keskustelu

JSP-sivut ovat datan näyttämiseen suunniteltu teknologia, joka koostuu neljästä erilaisesta elementistä: direktiiveistä, tekstielementeistä, tageista ja skript-elementeistä. Direktiivit ovat ohjeita JSP-prosessorille, joka kääntää JSP-tiedostot suoritettavaan muotoon. Tekstielementit ovat tavallisia HTML muotoisia sisältöjä. Tagien avulla voidaan kapseloida erilaisia toimintoja, kuten tietokantahakuja, helposti käytettävien HTML-tagien muotoon. Tagit joko tulostavat suoraan kapseloidun toiminnon tuottaman datan tai asettavat sen myöhemmin käytettävään muuttujaan, mutta tagit voivat myös tehdä yksinkertaisia tehtäviä, kuten suorittaa totta/epätotta tarkistuksia. Script-elementit voivat olla muuttujien julistuksia, scriplettejä tai ilmaisuja. Script-elementtien avulla voidaan tuottaa dynaamista sisältöä ja käyttää samoja toimintoja mitä tageilla, mutta ilman kapselointia.

2.5.1 JavaServer Faces

JavaServer Faces on Sun Microsystemsin kehittämä käyttöliittymäsovelluskehys, joka on tarkoitettu palvelinpuolen käyttöliittymien kehitykseen. JCP (Java Community Progress) aloitti sovelluskehyksen spesifikaation, JSR-127 (Java Spesification Request 127), määrittelyn vuonna 2001 ja julkaisi vuonna 2003 ensimmäisen JSF-spesifikaation. JSF-spesifikaatio 2.0:n odotetaan valmistuvan vuonna 2009. Tässä opinnäytetyössä käytetään versiota 1.2, joka julkaistiin 2006.

JSF-käyttöliittymät ovat osa Java EE -sovelluksia ja näin ollen vaativat Javan sovelluspalvelimen toimiakseen. JSF on suunniteltu joustavaksi ja se käyttää hyödykseen olemassa olevaa käyttöliittymästandardia ja webkerroksen konseptia. JSF on komponenttipohjainen sekä tapahtumavetoinen, jolloin voidaan helposti uudelleen käyttää aiemmin kehitettyjä osia. JSF-sovelluskehyksessä käytetään näkymien toteuttamiseen JSP (JavaServer Pages) sivuja, jotka pyytävät näytettävän datan JSF:n valmiiksi tarjoamilta tai itse JSF:llä kehitetyiltä komponenteilta.

Sun Microsystemsin mukaan JavaServer Faces arkkitehtuuri määrittelee selkeästi erottelun bisneslogiikan ja näyttökerroksen välillä, tehden samalla helpoksi yhdistää näyttökerros sovelluksen koodiin. Tämä suunnittelumalli mahdollistaa sen, että jokainen sovelluksen kehitysryhmään kuuluva jäsen pystyy keskittymään omaan kehitysprosessiinsa sekä tarjoaamaanyksinkertaisen ohjelmointimallin osien linkittämiseksi keskenään. (Sun Microsystems: JSF yleiskuva 2009.)

2.6 Tutkimusmenetelmät

Tutkimuksessa käytetään kahta eri tutkimusmenetelmää, jotta tutkimukselle asetettuihin tutkimuskysymyksiin saataisiin mahdollisimman hyvät ja kattavat vastaukset. Tutkimusmenetelmiksi on valittu teoreettis-käsitteellinen ja vertaileva tutkimusmenetelmä. Valitut tutkimusmenetelmät esitellään tässä luvussa.

2.6.1 Teoreettis-käsitteellinen tutkimusmenetelmä

Tutkimuksen toteuttamisessa käytetään pääasiassa teoreettis-käsitteellistä tutkimusmenetelmää. Teoreettis-käsitteellistä tutkimusmenetelmää käytetään kun selvitetään vastausta ensimmäiseen ja toiseen tutkimuskysymykseen. Teoreettis-käsitteellisessä tutkimuksessa pyritään löytämään vastaus aiemmin määriteltyyn tutkimuskysymykseen perustelemalla, esittämällä argumentteja sekä etsimällä vastaesimerkkejä. Tutkimusta voidaan vielä tehostaa käyttämällä problematisointia sekä kyseenalaistamalla itsestäänselvyyksiä. ”Teoreettis-käsitteellisen tutkimuksen lähestymistavalla ei ole selvää tutkimusmallia, koska se on itsenäistä, kriittistä ajattelua vaativaa, laaja-alaista ja syvällistä lukeneisuutta osoittavaa tutkimusta” (Teoreettis-käsitteellinen tutkimus n.d.).

2.6.2 Vertaileva tutkimusmenetelmä

Vertailevaa tutkimusmenetelmää käytetään pääasiassa selvitettäessä millaisia eroja eri RIA-käyttöliittymäteknologioilla on keskenään sekä miten ne eroavat perinteisen JSF-teknologiasta. Tutkimuksella on tarkoitus saada vertailukelpoisia tuloksia eri teknologioiden rasituksen kestosta, integraatiomahdollisuuksista sekä jokseenkin suuntaa antavia tuloksia käyttöliittymien kehitykseen kuluvasta ajasta.

Vertailevassa tutkimuksessa tarkastellaan aineiston yksilöitä tai tapauksia, jotka voidaan luokitella samaan lajiin mutta kuitenkin omaavat jonkin eroavaisuuden toisistaan. Tarkastelun jälkeen eroavaisuuksia täsmennetään ja eroavaisuuksia tarkastellaan silmällä pitäen mahdollisia johdonmukaisesti vaihtelevia eroja. ”Vertailumenetelmä on myös joustava: se sopii niin koko tutkimushankkeen rungoksi kuin myös yksityiskohtien vertailemiseen toisten menetelmien apuna. Lisäksi sillä on harvinainen etu: sen avulla voidaan tehokkaasti saada esiin (eksplikoida) hiljaista eli sanatonta tietoa tai asenteita”. (Routia 2007.)

3 INTERNET KÄYTTÖLIITTYMÄT 2000-LUVULLA

3.1 Teknologioiden kehittyminen

Uudelle vuosituhannelle saavuttaessa internet oli jo omaksunut graafisien elementtien näytön, näkymien ulkoasunmuodostamisen erillisillä tyylitiedostoilla, Javascriptin jolla voitiin lisätä käyttöliittymien toiminnallisuuksia sekä Java-appletit. Seuraavissa luvuissa onkin tarkoituksena kertoa tarkemmin näiden teknologioiden kehitymisestä sekä antaa perustiedot niiden käyttötarkoituksista sekä käyttötavoista.

3.1.1 CSS tyylitiedostojen esiin marssi

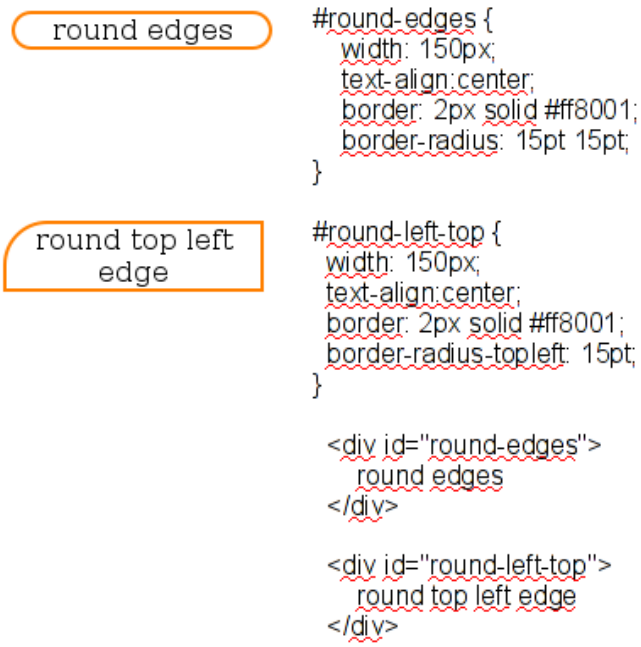
CSS:ää käytettiin jo 2000-luvun alussa laajasti eri internet-sivustoilla. Tyylitiedostojen levinneisyydestä on nykypäivänä kuitenkin tarjolla hyvin vähän statistiikkaa, jonka avulla voitaisiin selventää sen levinneisyyttä internet-sivustoilla. Statistiikan vähäisyyteen vaikuttavia tekijöitä on vaikea määritellä, mutta yksi suurin tekijä todennäköisesti on CSS-tyylien monipuoliset käyttötavat. Käyttötapojen monipuolisuus todennäköisesti vaikeuttaa luotettavan statistiikan keräämistä siinä määrin, ettei sen kerääminen ole kannattavaa. Sivustolla voi olla esimerkiksi HTML-elementtien style-attribuuttiin määriteltyjä tyylimäärityksiä tai HTML-dokumentin sisällä style-elementtien sisälle määriteltyjä tyylejä sekä ulkoisia css-tiedostoja. Näillä voidaan määritellään koko HTML-dokumentin ulkoasu tai sitten vain pieni osa siitä. Rene Saarsoon tekemä diplomityö internet sivuilla käytetyistä teknologioista antaa jonkinlaista käsitystä CSS-tyylitiedostoja käyttävien sivustojen määrästä. Tutkimus osoittaa, että 67 % prosenttia internet sivuista käytti CSS-tyylitiedostoja ainakin jossain määrin sivujen ulkoasun määrittelyyn vuonna 2006. (Saarsoo 2006.) Statistiikkaa keräävää ohjelmaa ajettiin tutkimuksessa vain yhden viikon ajan. Tässä ajassa se keräsi tietoja alle kahdelta miljoonalta sivustolta, joka on vain pieni osuus kaikista internet-sivustoista, joita oli vuoden 2005 tammikuussa Gulli A. ja Signorini A:n tekemän tutkimuksen mukaan noin 11.5 miljardia (Gulli & Signorini 2005). Näin ollen Sarsoon tutkimuksen tuottamia tietoja voidaan pitää vain suuntaa antavina.

Uuden vuosituhannen alussa W3C valmisteli CSS määrityksen toisen version päivitystä, CSS2.1, josta julkaistiin vuonna 2002 versio, jota W3C alkoi suositella käytettäväksi. CSS2.1 määrittelystä käy ilmi, että siinä on korjattuja virheitä, joita aiemmasta määritelmästä löydettiin. Esimerkiksi absoluuttisesti määriteltyjen elementtien korkeuden ja leveyden määrittelyyn liittyviä ongelmia on korjattu. Määrittelyssä esitellään myös uusia ominaisuuksia, joita on laajasti toivottu. Uusia ominaisuuksia on esimerkiksi kolme uutta arvoa ominaisuudelle, jolla voidaan määritellä listaelementtien ”etumerkki” sekä tuki määritellä uudenlainen valitsin id-attribuuteille. Uusi valitsin mahdollistaa niin sanotun jokerivalitsimen käytön id-attribuuttien yhteydessä, jolloin samaa tyylimääritystä voidaan käyttää useammalla eri sivulla eri elementeille ja näin ollen pienentää CSS määrittelyn kokoa, uuden id valitsimen käyttötapaa esitellään kuviossa 5.

CSS3 määritys jatkaa siitä mihin edeltäjänsä määrittelyn jätti ja uusimmasta määrityksen vedoksesta näkeekin jo, että luvassa on useita uusia ominaisuuksia jotka helpottavat sivujen ulkoasujen ja käyttöliittymien toteuttamista. Uusia ominaisuuksia tulee olemaan muun muassa monipuolisempi tapa muotoilla HTML-elementtejä. Uusi määritys mahdollistaa esimerkiksi elementtien kulmien pyöristyksen pelkän CSS:n avulla, joka on aiemmin täytynyt toteuttaa läpinäkyvien kuvien avulla. Uuden ominaisuuden helppous on nähtävissä kuviossa 6. Uusi muotoilumahdollisuus ei kuitenkaan rajoitu pelkästään kulmien pyöristykseen vaan se mahdollistaa myös elementtien reunojen mielivaltaisen muotoilun. Kaikki uudistukset mitä tähän mennessä on suunniteltu tulevan CSS3-määritykseen on luettavissa osoitteessa http://www.w3.org/Style/CSS/current-work, jossa on yksityiskohtaisesti esitetty eri määrittely-osa-alueiden tila. CSS3-määrityksen julkaisupäivämäärää ei kuitenkaan vielä tiedetä ja saattaa kestää jopa pari vuotta ennen kuin siitä julkaistaan versio, jota W3C alkaa suositella käytettäväksi. Tosin tämä ei ole niin suuri haitta mitä voisi kuvitella sillä tällä hetkellä vain kaksi internetselainta tukee edes osaa CSS3 vedoksen ominaisuuksista. Kaikki selaimet eivät tue täysin vieläkään edes CSS2.1 määritystä.

```
<!– Ensimmäisen sivun osa –>
<div>
   <p id="crisis">

   </p>
</div>

<!– Toisen sivun osa -->
<div>
   <span id="crisis">

   </span>
</div>

<!– CSS tyylimääritys –>
*#description {
   color: #fff;
   background-color: #000;
   font-size: 12px;
}
```

KUVIO 5. CSS esimerkki id-valitsimesta



KUVIO 6. Pyöristetyt kulmat CSS3:n avulla

3.1.2 Javascriptin valtakausi

Ecma Internationalin standardikieli Ecmascript, jota Javascript noudattaa, saavutti kolmannen versionsa juuri ennen vuosituhannen vaihdetta. Kolmannen version jälkeen standardista ei ole julkaistu uusia versiota, koska neljäs versio jätettiin julkaisematta poliittisista syistä. Tämä ei kuitenkaan tarkoita sitä, etteikö standardia olisi kehitetty. Tällä hetkellä standardista on tekeillä viides versio, jonka arvellaan valmistuvan vuoden 2009 loppupuolella. Javascript itsessään oli kehittynyt samaan aikaan versioon 1.3. Vaikkei Ecma International olekaan julkaissut uutta versiota standardistaan on Javascript siitä huolimatta kehittynyt jatkuvasti ja siitä odotetaankin julkaistavaksi versiota 1.9. Myös selainten tuki on kehittynyt uudella vuosituhannella huomattavasti, tärkein uudistus on W3C:n määrittelemän DOMin (Document Object Model) tuen tuleminen yleiseksi käytänteeksi. W3C kuvaa DOMin alusta- ja kielineutraaliksi rajapinnaksi, joka sallii ohjelmien ja scriptien pääsyn dynaamisesti dokumentin rakenteeseen ja manipuloimaan sen rakennetta, tyylejä ja sisältöä (W3C DOM 2009). DOM

tuen myötä siis sivun sisältöä pystytään muokkaamaan dynaamisesti käyttäjän tekemien valintojen pohjalta, ilman sivun uudelleen päivitystä, Javascriptin avulla.

JavaScript-kirjastojen ja -sovelluskehyksien määrä sekä ominaisuudet ovat myös lisääntyneet huimasti 2000-luvulla. Samaan aikaan Javascriptin suosio on lähtenyt kasvuun pienen taantumakauden jälkeen. Suurin kiitos Javascriptin suosion kasvuun kuuluu AJAX-tekniikkaryhmän syntymiselle sekä sen tuen tuleminen kaikille valtavirran internet-selaimille. AJAXista kerrotaan tarkemmin seuraavassa luvussa. JavaScript-kirjastoja ja -sovelluskehyksiä löytyykin nykypäivänä lukemattomia määriä, joista ajaxlinestä löytyvä artikkeli määrittelee tunnetuimmiksi jQueryn, prototypen, script.aculo.us:n, MooToolsin, ExtJS:n, Qooxdoon, Yahoo! UI Libraryn (YUI), MochiKitin, Midorin sekä The Dojo Toolkitin (Ajaxline – JavaScript Frameworks 2009).

Valmiiden Javascript sovelluskehyksien käyttö helpottaa Javascriptin ominaisuuksien käyttöä. Esimerkiksi JQueryllä välilehtien toteutus vaatii yksinkertaisimmillaan yhdellä rivillä javascriptiä, kuten kuvio 7 osoittaa. Kaikki suosituimmat Javascript-sovelluskehykset tarjoavat websivuillaan API (Application Programming Interface) dokumentaation, jotka helpottavat sovelluskehyksien ominaisuuksien käyttöä sovelluksissa. Sivustot tarjoavat myös kattavat esimerkit sekä paljon ohjeita ominaisuuksien käyttöön. Joidenkin Javascript sovelluskehyksien käyttöön on myös kirjoitettu kirjoja esimerkiksi JQuery in Action, ExtJs in Action sekä Practical Dojo Projects.

Myös monen JavaScript-sovelluskehyksen mukana tulee niiden käyttöliittymäkomponenteille perus-CSS-tyylitiedostot, jolloin projektissa ei välttämättä tarvitse käyttää aikaa komponentin ulkoasun määrittelyyn. Yleensä komponenttien oletusulkoasu ei kuitenkaan ole joko väritykseltään tai asettelultaan aivan sellainen kuin sivustolle halutaisiin, tällöin oletusulkoasu voidaan korvata omalla teemalla. JavaScript-sovelluskehyksissä on pääasiassa ominaisuuksia todella paljon ja useimpia pystytään itse laajentamaan tekemällä omia pluginejä. Yleisimpiä ominaisuuksia, jotka löytyvät käytännössä kaikista yleisimmistä Javascript sovelluskehyksistä, on AJAX-pyyntöjen helppo toteutus, erilaiset drag 'n' drop pluginit joiden avulla voidaan tehdä raahattavia elementtejä, lomakevalidaattoreita, kuvagalleriat, datataulukot, joilla voidaan esittää suuria määriä dataa selkeästi, sekä erilaisia visuaalisia tehoste-efektejä. Käytännössä niistä löytyy kaikki samat ominaisuudet sekä muutamia lisäominaisuuksia mitä löytyy työpöytäsovelluksien käyttöliittymien rakentamiseen tarkoitetuista kirjastoista.

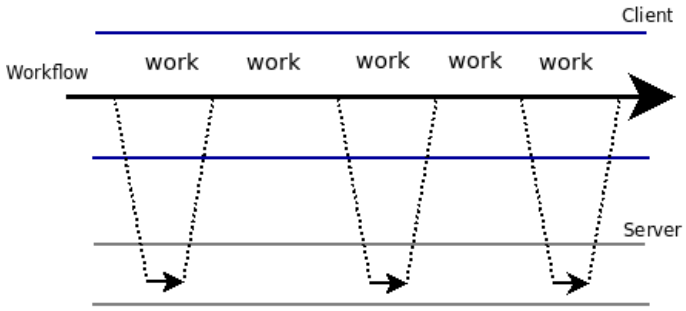
```
<!– HTML osa jotka muutetaan välilehdiksi -->
<div id="tab-container">
   <div id="tab1">
      <!– First tab's content –>
      Lorem ipsum
    </div>
    <div id="tab2">
      <!– Second tab's content –>
      Lorem ipsum
   </div>
</div>

<!– JQuery koodi, joka tekee välilehdet -->
<script type="text/javascript">
   <!--
      $('#tab-container').tabs();
   -->
</script>
```

KUVIO 7. JQuery esimerkki välilehtien toteutuksesta

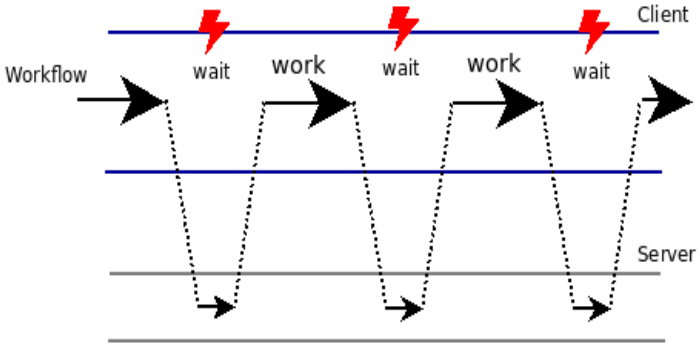
3.1.3 AJAX-teknologiaryhmän nousu

AJAX on uuden vuosituhannen tärkein uudistus internetin käyttöliittymiä ajatellen sillä ilman sitä käyttöliittymät vaatisivat edelleen koko sivun uudelleen latauksen kun haluttaisiin kommunikoida palvelimen ja selaimen välillä. AJAXilla tarkoitetaan tekniikkaryhmää, jolla voidaan tehdä palvelinpyyntöjä ilman, että sivua tarvitsee ladata uudestaan. Sivupyynnön vastauksena saadaan aina XML- tai XSL-muotoista dataa, joka voidaan esittää joko sellaisenaan tai poimia sieltä tarvitut tiedot. AJAXia käyttäessä käytetään JavaScriptiä jolla luodaan XmlHttpRequest-olio joka toteuttaa palvelinpyynnön taustalla kuten kuvio 8 esittää, jolloin sivun käyttöä voidaan jatkaa ilman katkoksia. Tavallisessa sivun uudelleen lataukseen pohjautuvassa menetelmässä sivuston käyttö katkeaa palvelimen sivulatauksen ajaksi kuvion 9 mukaisesti. (Bibeault, Crane, Sonneveld 2007, 4 – 6.)

Terminä AJAX on vielä kohtuullisen nuori sillä se esiteltiin vasta vuoden 2005 helmikuussa Jesse James Garrettin toimesta. Tekniikat jotka AJAX sisältää ovat kuitenkin olleet olemassa termiä pidempään. Esimerkiksi JavaScript, joka sitoo eri tekniikat yhteen, on ollut käytössä jo 90-luvulta ja XML:n perustat ulottuvat 80-luvulle, vaikkakin XML itse julkaistiin vasta 90-luvun lopulla. AJAXin sydän jonka avulla hoidetaan palvelimen ja selaimen välinen kommunikointi, asynkroninen I/O, juontaa myös juurensa 90-luvun puoliväliin Javan ensimmäiseen versioon, mutta itse XmlHttpRequest-olio, jota AJAXissa käytetään, julkaistiin vasta 1999 Internet Explorer 5:n mukana Active X komponenttina. Muiden selainten kehittäjien huomattua 2000-luvulla mitä etuja XmlHttpRequest-olion avulla on mahdollista saavuttaa kehittivät he omat versionsa XmlHttpRequest-oliosta. Oliot eivät kuitenkaan ole täysin yhteensopivia keskenään, tästä syystä W3C alkoikin määritellä XmlHttpRequest-oliota varten standardia huhtikuussa 2006 (The XMLHttpRequest Object draft 2006). Standardi ei ole vielä kuitenkaan ehtinyt valmistua ja siihen on tehty viimeisin päivitys huhtikuussa 2008, eli kehitystä on kuitenkin tapahtunut.



KUVIO 8. Ajax workflow (Bibeault, Crane, Sonneveld 2007, 6).



KUVIO 9. Uudelleen lataukseen pohjautuva workflow (Bibeault, Crane, Sonneveld 2007, 5).

3.1.4 Rikkaat internet sovellukset

Viimeisin tulokas internet käyttöliittymien kehityksessä ovat RIA-teknologiat, jotka mahdollistavat entistä enemmän perinteisiä työpöytäsovelluksia muistuttavien käyttöliittymien toteutuksen, niin ulkoasultaan kuin toiminnoiltaan. Ensimmäisiä RIA-teknologioita alkoi ilmestyä vuosina 2004 - 2005, jonka jälkeen niiden määrä on lähtenyt lähtenyt kasvuun. Nykypäivänä niitä löytyykin jo kymmenkunta. Kuitenkaan itse sovelluksia, jotka on toteutettu RIA-teknologiaa hyväksi käyttäen siten, että ne muistuttaisivat työpöytäsovelluksia on erittäin vähän. Enemmän perinteisiä internet-sivuja muistuttavia sovelluksia on toteutettu huomattavasti enemmän, mutta näitä käyttäjät eivät miellä niin helposti RIA-teknologioilla toteutetuiksi, koska se ei näy päälle päin.

Sovellusten vähyyttä voidaan selittää sillä, että vielä tutustutaan RIA-teknologiohin ja haetaan sopivaa suhdetta niiden käyttämiseen. Samalla varotaan tekemästä liian suuria muutoksia jo olemassa oleviin käyttöliittymiin. Uusia käyttöliittymiä kehitetään pienissä askelissa kohti työpöytäsovelluksien ulkoasua, jottei säikäytettäisi käyttäjiä aivan uudenlaisella tavalla hyödyntää internet-sivustoja.

RIA-teknologioille on yleistä niin asiakasohjelman kuin palvelinpuolen toteutuksen mahdollistaminen. Yleisin käyttötapa on kuitenkin toteuttaa palvelinpuoli jo tunnetulla teknologialla ja tehdä asiakasohjelmasta pyyntöjä sinne. Tämä johtuu paljolti siitä, että palvelinpuolta ollaan kehitetty jo pitkään ja tarvitaan vain uusi käyttöliittymä sen käyttämiseen jolloin ei ole järkevää aloittaa toteuttamaan koko järjestelmää uudelleen.

Asiakasohjelman ja palvelimen välinen keskustelu toteutetaan RIA-teknologioissa pääasiassa käyttäen XmlHttpRequest-oliota, mutta poikkeuskin löytyy. Eli myös RIA-teknologioiden kehittymiselle on ollut elintärkeää, että XmlHttpRequest-olion oli kehittynyt ja sen myötä AJAX-teknologiaryhmä pääsi syntymään. Suurin ero-RIA-teknologioiden ja JavaScript-sovelluskehyksien välillä on palvelinpuolen toteutusmahdollisuuden lisäksi se, että suurimmassa osassa niistä käytetään jotain vahvemmin tyypitettyä kieltä kuin JavaScript, joka käännetään aina ajonaikaiseen ympäristöön sopivaksi koodiksi tai muuten suoritettavaan muotoon. Pääasiassa on kolme eri tapaa ajaa RIA-teknologioilla toteutettuja sovelluksia. Ehkä yleisin on GWT:n tapa, jossa Java-koodi käännetään selaimessa suoritettavaksi Javascript koodiksi. Toinen tapa on Adobe Flexin käyttämä tapa, jossa kirjoitettu koodi käännetään Flash soittimessa ajettavaksi koodiksi ja kolmas tapa on Java Fx:n tapa suorittaa käännetty koodi selaimen JRE-pluginiä käyttäen.

3.2 2000-luvun kehitys yhteenvetona

Yhteenvetona voidaan sanoa, että 90-lukuun verrattuna internet-käyttöliittymäteknologiat ovat kehittyneet huimasti, mutta kaikkia uudistuksien tarjoamia hyötyjä ei vielä hyödynnetä täysin osittaisesta murrosvaiheesta johtuen. Näin ollen voidaankin odottaa, että seuraavien 5-10 vuoden aikana sivustojen ulkoasut ja ominaisuudet tulevat muuttumaan huomattavasti, kun kaupalliset tahot alkavat hyödyntää RIA-teknologioi-

ta täydellä teholla internet-palveluissaan sekä sivustoillaan. Myös teknologiat tulevat kehittymään sekä vakiintumaan jatkossa lisää, varsinkin RIA-teknologiat ovat vielä niin nuoria etteivät ne ole vielä täysin muodostuneet lopulliseen muotoonsa.

Nykyisiin käyttöliittymiin pystytään toteuttamaan helposti JavaScript sovelluskehyksien avulla niin käyttöä helpottavia toiminnallisuuksia kuin sivun ulkoasua kohentavia animaatiota erilaisien siirtymien välille. RIA-teknologioita käyttäessä voidaan käyttöliittymät toteuttaa vahvasti tyypitettyjä kieliä hyödyntäen, jolloin mahdolliset virheet havaitaan jo aikaisemmassa kehitysvaiheessa. Myös sivustojen ulkoasujen määritys on helpottunut sekä tehostunut ja monipuolistunut 1990-luvun alkupuolen ajoista huomattavasti, vaikka uudella vuosituhannella ei sinällään ole vielä julkaistu mitään virallisia uudistuksia määrityksiin. Lähitulevaisuudessa on kuitenkin luvassa uudistuksia myös sille rintamalle, joita toivottavasti myös internet-selaimet alkavat tukea yhteneväisesti.

4 RIA-TEKNOLOGIOIDEN HYÖDYT JA HAITAT

RIA-teknologia tuovat paljon hyötyä web-sovellusten kehitykseen, mutta niiden mukana tulevat myös omat haittansa. Onkin tärkeää tuntea teknologiat, joiden väliltä valinta täytyy tehdä projektia aloittaessa. Tällöin voidaan ennakoida mahdollisia ongelmia joihin voidaan törmätä projektin edetessä. Tässä luvussa käydään läpi mitä yleisiä haittoja ja hyötyjä RIA-teknologioista on käyttöliittymien kehityksessä ja seuraavassa luvussa pureudutaan tarkemmin teknologiakohtaisiin ongelmiin, jos niitä löydetään esimerkkisovellusta toteuttaessa.

4.1 RIA-teknologioiden mukanaan tuomat edut

Verrattaessa eri JavaScript-sovelluskehyksien tuomiin hyötyihin RIA-teknologiat eivät tuo suurta määrää lisähyötyä, jos niitä käytetään pelkästään käyttöliittymien kehityksessä. RIA-teknologioiden toteutuksessa käytetään yleensä vahvasti tyypitettyjä kieliä, mikä puolestaan vähentää mahdollisten virheiden määrää jo toteutusvaiheessa. Esimerkiksi merkkijono-tyyppisiä muuttujia ei voida sekoittaa numeraalisiin muuttujiin, sillä tämän tyyppiset virheet eivät mene kääntäjästä lävitse, vaan kääntäjä antaa virheilmoituksen, jossa kerrotaan virheen syy ja kohta missä se ilmenee koodissa. Vahvasti tyypitetyille kielille on myös ominaista, että niille tarjotaan tehokkaat debug-työkalut, joiden avulla voidaan seurata rivi riviltä mitä koodissa tapahtuu. Tämä helpottaa virhetilanteiden syiden selvitystä ja sitä kautta myös nopeuttaa virheiden korjausta huomattavasti.

Kun käyttöliittymä toteutetaan RIA-teknologioita käyttäen, voidaan se toteuttaa siten että se muistuttaa erittäin paljon työpöytäsovellusta niin toiminnoiltaan kuin ulkoasultaan. Sinällään ei vaadita RIA-teknologioita siihen, että web-käyttöliittymät saadaan muistuttamaan työpöytäsovelluksia sillä myös RIA-teknologiat käyttävät käyttöliittymän toteutukseen samoja HTML-elementtejä ja CSS-määrityksiä kuin perinteiset teknologiat. Toiminnallisuuksien toteuttaminen ei kuitenkaan onnistu perinteisillä teknologioilla samaan tapaan mitä se on mahdollista tehdä RIA-teknologioilla, vaikka ne on

mahdollista tehdä JavaScript-sovelluskehyksiä käyttäen. Sillä käyttöliittymien toiminnallisuuksien toteuttaminen pohjautuu molemmissa tapauksissa DOM-puun manipulointiin sekä palvelinpyyntöjen toteuttamiseen taustalla XmlHttpRequest-oliota käyttäen. RIA-teknologiat kuitenkin mahdollistavat toiminnallisuuksien toteutuksen yksinkertaisemmin ja yritysten yrityssalaisuuksia ajatellen myös turvallisemmin. Kun toteutettu käyttöliittymä käännetään vaikeasti luettavaan muotoon, sen toteutuksen kopiointi kilpailevan tahon toimesta on huomattavasti vaikeampaa ja koodin ajonaikainen manipulointi vaikeutuu.

RIA-teknologioilla on kuitenkin mahdollista toteuttaa sellaisia toiminnallisuuksia mitä ei JavaScript-sovelluskehyksillä voida toteuttaa. Näistä esimerkkeinä voidaan sanoa hiiren oikealla näppäimellä avautuva konteksti menu sekä pikanäppäimet. Tämän tyyliset niin sanotut piilotetut ominaisuudet lisäävät käyttöliittymien käytettävyyttä paljon, kun lisätoiminnallisuudet ovat piilossa peruskäyttäjiltä, jotka eivät niitä kaipaa, mutta silti tehokäyttäjät pääsevät niihin helposti käsiksi. Monen alustan tuki voidaan myös kokea suurena etuna RIA-teknologioille sillä niillä on mahdollista toteuttaa käyttöliittymiä ja sovelluksia, jotka toimivat niin internetissä ja työpöydällä kuin mobiililaitteissakin. Jotta sovellus saataisiin toimimaan eri alustalla tarvitaan korkeintaan käännösvaiheessa , teknologiasta riippuen, eri käännösparametrejä. Myös eri selainten tuki on huomioitu yleensä suoraan RIA-teknologioissa eikä niihin tarvitse erikseen kiinnittää huomiota käyttöliittymää kehitettäessä.

Toteutetuilta sivulta löytyvien erilaisien toimintojen ja palvelujen hajauttaminen helpottuu myös sillä palvelut voidaan noutaa helposti AJAXilla eri palvelimilta ja palvelun näyttävä komponentti voidaan piirtää näytölle esimerkiksi vasta sitten kun kaikki siihen tarvittavat tiedot on saatu noudettua. Samaan aikaan kun esimerkiksi sääpalvelun tietoja vasta noudetaan voidaan muuta sivua piirtää eteenpäin ja hakea muiden toiminnallisuuksien tarvitsemia tietoja toisaalta. Palveluja hajauttamalla useammalle palvelimelle saavutetaan myös parempi kuormituksen kesto sivustolle kun kuorma jakautuu useammalle palvelimelle. Vaikka yksi palvelimista kaatuisikin kuorman alla se ei aiheuta kuin yhden palvelun puuttumisen tai toiminnallisuuden vajaavaisuuden. Kuitenkin kuten ulkoasun työpöytämäisyys niin tämäkin on mahdollista saavuttaa ilman RIA-teknologioita käyttämällä JavaScript-sovelluskehyksiä. RIA-teknologioilla on kuitenkin etuna, että niille voidaan palvelujen osoitteet antaa esimerkiksi Spring Fra-

meworkin kautta, jolloin ne on helposti uudelleen konfiguroitavissa ja niihin voidaan vaikuttaa toteutuskoodissa siistillä tavalla.

4.2 RIA-teknologioiden ongelmakohdat

Yksi ehkä suurimmista ongelmakohdista RIA-teknologioissa on niiden vaatimat ajonaikaiset ympäristöt. Niiden vaatimat ajonaikaiset ympäristöt eivät tule oletusarvoisesti selainten mukana vaan ne on erikseen asennettava. Tähän kuitenkin poikkeuksena JavaScript, joka tulee selainten mukana, tosin sekin on mahdollista kytkeä pois käytöstä selaimen asetuksista. Jos selaimesta ei löydy tukea RIA-teknologian vaatimalle ajonaikaiselle ympäristölle ei käyttöliittymää tällöin voida ladata ollenkaan. Tämän johdosta joudutaan usein tekemään vaihtoehtoinen käyttöliittymätoteutus perinteisillä teknologioilla, joka aiheuttaa lisätyötä. Tämä ei ole pakollista, mutta erittäin suositeltavaa, jotta palvelun käyttöaste on mahdollisimman korkea erilaisista tilanteista huolimatta.

Yksittäisten sivupyyntöjen määrä kasvaa paljon, kun moni eri käyttöliittymäkomponentti tekee sivupyyntöjä erikseen taustalla. Yksittäisten sivupyyntöjen ja vastausten koko kuitenkin jää pienemmiksi kuin jos kaikki ladattaisiin kerralla sivun uudelleen latauksen yhteydessä. Tästä kuitenkin voi kuitenkin koitua ongelmia, jos sivupyyntöjen määrä kasvaa rajusti, esimerkiksi suuren käyttäjämäärän johdosta, jolloin palvelin ei ehdi vastaamaan kaikkiin pyyntöihin vaan tukkeutuu. Tätä voidaan välttää, tai vähintäänkin ongelman todennäköisyyttä laskea rajusti, edellisessä kappaleessa mainitulla palveluiden hajauttamisella. On myös muita tapoja vähentää sivupyyntöjen määrää kuten asettaa kaikki pienet kuvat yhteen kuvaan, jossa on läpinäkyvä taustaväri ja CSS:ää käyttäen näyttää isosta kuvasta oikea kohta, jolloin näkyy vain yksi pieni kuva. Ensimmäistä kertaa sivustolle saavuttaessa on kuitenkin ladattavaa enemmän mitä perinteisillä menetelmillä toteutetuissa käyttöliittymissä, sillä selain joutuu lataamaan HTML-koodin sekä sen lisäksi toiminnallisuudet toteuttavaa koodia. Tämä saattaakin haitata käyttäjiä, joilla on hitaat internet-yhteydet kuten mobiiliyhteyksiä käyttävät.

Kaikkien RIA-teknologioiden integrointi jo olemassa olevan sovelluksen päälle ei

välttämättä onnistu täysin saumatta ja tämä onkin yksi vaihe johon kannattaa projekteissa varata aikaa etukäteen. Muun muassa erilaiset ORM (Object Relation Mapping) ratkaisut saattavat aiheuttaa ongelmia käyttöliittymiä toteutettaessa, nämä ongelmat ovat kuitenkin yleensä kierettävissä, mutta vaativat kuitenkin aikaa tai hyvää tuntemusta käytetyistä teknologioista ja niiden toimintamalleista.

5 RIA-TEKNOLOGIOIDEN EROT

RIA-teknologioiden eroja tutkitaan toteuttamalla jokaisella teknologialla yksinkertainen kuvagalleria, johon käyttäjät voivat kirjoittaa kommentteja kuvista. Kommentointi-lomakkeen kentiksi tulee käyttäjänimi sekä viestikenttä. Viestin lomake validoidaan sekä asiakasohjelmassa sekä palvelimen päässä. Asiakasohjelma validoi kentät reaaliaikaisesti, sitä mukaa kun käyttäjä täyttää lomaketta, virheellisen datan varalta ja näyttää virheen löydyttäessä ilmoituksen siitä. Jos lomakkeesta ei löydy validointivirheitä tallennetaan kommentointilomakkeen tiedot tietokantaan, josta viesti ladataan näytettäväksi kuvan yhteydessä.

Kun jokaisella teknologialla on toteutettu kuvagalleria niille suoritetaan suorituskykytestaus, jolla saadaan selville miten eri teknologioilla toteutetut ratkaisut kestävät kuormitusta. Suorituskyky testauksen toteutukseen käytetään Jmeter-sovellusta. Jmeterillä voidaan suorittaa palvelimelle sivupyyntöjä ja tarkkailla tarkkaillaan palvelimen vastausaikoja. Vertailukohtana RIA-teknologioiden suorituskyvylle käytetään JSF:llä toteutetun kuvagallerian testien tuloksia sekä muiden RIA-teknologioiden tuloksia.

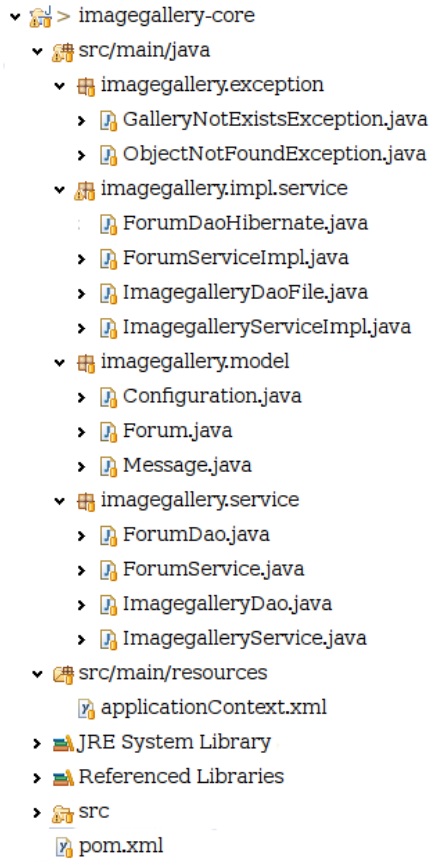
5.1 Toteutukset

Toteutettavat ohjelmat jaetaan kahteen eri osaan, joista toinen osa käyttöliittymäprojekti ja toinen bisneslogiikkaprojekti. Bisneslogiikkaprojekti on kaikille käyttöliittymäprojekteille sama. Se kuvastaa jo entuudestaan olemassa olevaa sovellusta, jolle toteutetaan vain uusi käyttöliittymä. Näin ollen voidaan vertailla eri RIA-teknologioiden integrointia jo olemassa oleviin sovelluksiin.

5.1.1 Sovelluksien perusrunko

Perusrunko eli bisneslogiikkaprojekti toteutetaan puhtaasti Javalla siten, ettei se ota kantaa siihen missä ympäristössä sitä käytetään. Projektissa käytetään Spring Frameworkia bisneslogiikan käyttämien service- ja DAO-papujen (Beans) sekä transak-

tioiden määrittelyyn. Spring Framework hoitaa myös papujen riippuvuuksien injektoinnin (Dependency injection) ajon aikana. Tietokantatapahtumat hoidetaan Hibernaten avulla, jonka käyttöönotto on määritelty Spring Frameworkin konfiguraatiossa. Konfiguraatio on esitetty liitteessä 1. Lisätietoa Spring Frameworkista ja sen konfiguroinnista löytyy osoitteesta http://www.springsource.org. Spring Frameworkin konfiguraatiossa määritellään tietokannan tyyppi sekä mitkä model-luokat kuvaavat tietokantatauluja. Tietokantataulujen attribuutit sekä yhteydet toisiin tauluihin määritellään käyttämällä annotaatiota model-luokkien get-metodien yhteydessä. Toinen vaihtoehto olisi määritellä attribuutit sekä tauluyhteydet XML-muotoisessa Hibernate konfiguraatiossa. Lisätietoa Hibernatesta ja sen konfiguroinnista löytyy osoitteesta http://www.-hibernate.org. Projektin kääntämiseen käytetään Mavenia, joka mahdollistaa helpon tavan hallita projektin riippuvuuksia ja pakottaa projektin rakenteen selkeäksi. Lisätietoa Mavenista löytyy osoitteesta http://maven.apache.org. Bisneslogiikkaprojektin pakettirakenteessa on eroteltu omiin paketteihinsa model-luokat, service- ja dao-rajapinnat sekä niiden toteutukset. Spring Frameworkin konfiguraatiotiedosto on myös eroteltu muista tiedostoista erilleen resources-kansioon. Projektin tarkempi rakenne on esitetty kuviossa 10.

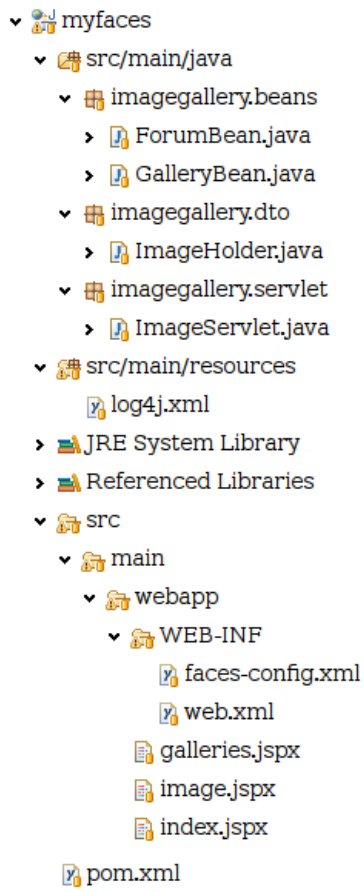


KUVIO 10. Bisneslogiikkaprojektin rakenne

5.1.2 JSF - Vertailukohde

JSF-käyttöliittymäprojektissa käytettiin Sun Microsystemsin JSF-spesifikaatiosta tarjoaman esimerkkitoteutuksen sijaan Apache Software Foundationin tekemää MyFaces-toteutusta ja MyFaces Trinidad-sovelluskehystä. MyFaces Trinidadin mukana saatiin käyttöön muun muassa suuri määrä toteutusta helpottavia valmiita komponentteja. Esimerkiksi forEach-silmukka, jonka avulla voidaan käydä helposti läpi lista-tyyppisien muuttujien arvot. JSF-tyyppisten projektien kääntäminen onnistui Mavenilla ongelmitta ja se mahdollistaa myös JSF-projektien rungon generoinnin. Apache Software Foundation tarjoaa Mavenille archetypeCatalogin, josta löytyy viisi erilaista projektirunkoa JSF-projekteille. Projektin luonnissa käytettiin ”myfaces-archetype-trinidad” -nimistä archetypeä, joka sisältää valmiiksi MyFaces- ja trinidad-konfiguraatiot. Koska projekti on generoitu Mavenilla, muistuttaa sen rakenne paljon perusprojektin rakennetta. Projektin rakenne on nähtävissä kuviosta 11. Projektin

web.xml ja faces-config.xml -tiedostot löytyvät Java-web-projekteille ominaisesti WEB-INF kansiosta. Faces-config.xml on liitteenä 2 ja web.xml liitteenä 3. Pakettirakenne on jaettu kolmeen osaan: dto-oliot, servletit ja managed-beanit. Managed-beanit ovat tavallisia Java-papuja, jotka on määritelty faces-configissa. JSF-käyttää managed-beaneja näkymissä olevan datan väliaikaiseen tallentamiseen näkymäsiirtymien välillä. Kun JSF-sivulla klikataan linkkiä tai lähetään lomake, JSF-sovelluskehys asettaa tarvittavat tiedot managed-beaniin, josta tiedot voidaan palauttaa näkymään. Hyvä esimerkki tästä on lomakkeen validointi, jossa validoinnin epäonnistuessa kenttien arvot palautetaan managed-beanista.



KUVIO 11. JSF-projektin rakenne

Integraatio perusprojektin päälle onnistui suoraan ilman ongelmia. Spring Framework täytyi konfiguroida käynnistymään projektin web.xml:ssä määrittelemällä Spring Frameworkin ContextLoaderListener. ContextLoaderListenerin määrittely on kuitenkin aina pakollista Spring Frameworkia käyttöönotettaessa, joten se ei varsinaisesti liity integrointiin. Itse integraatio saatiin toimimaan määrittelemällä faces-config.xml -tiedostossa variable-resolveriksi Spring Frameworkin tarjoama DelegatingVariableResolver. DelegatingVariableResolverin määrittelyn jälkeen voitiin Spring Frameworkin konfiguraatiossa määriteltyjä papuja injektoida suoraan faces-config.xml:ssä määritellyille managed-beaneille. Injektointi onnistui käyttämällä managed-property tageja, joille annettiin muuttujan nimi ja arvo. Muuttujan arvo tuli antaa Unified EL (Unified Expression Language) -kieltä käyttäen, esimerkiksi #{pavun_id}.

Unified EL on JSF:ää varten suunniteltu ilmaisukieli. Se on syntaksiltaan samanlainen kuin JSP-sivuilla käytetyn EL:n (Expression language) syntaksi, ainoastaan lausekkeen aloittava merkki on vaihdettu $:sta #:aan. Unifiel EL kehitettiin, koska EL-kieltä käytettäessä tiettyjen JSP-tagien kanssa saattoi aiheutua ongelmia. EL-kielten avulla voidaan viitata JSP- ja JSF-sivuilta Java-oliossa oleviin attribuutteihin. (Sun Microsystems – Unified EL 2009.)

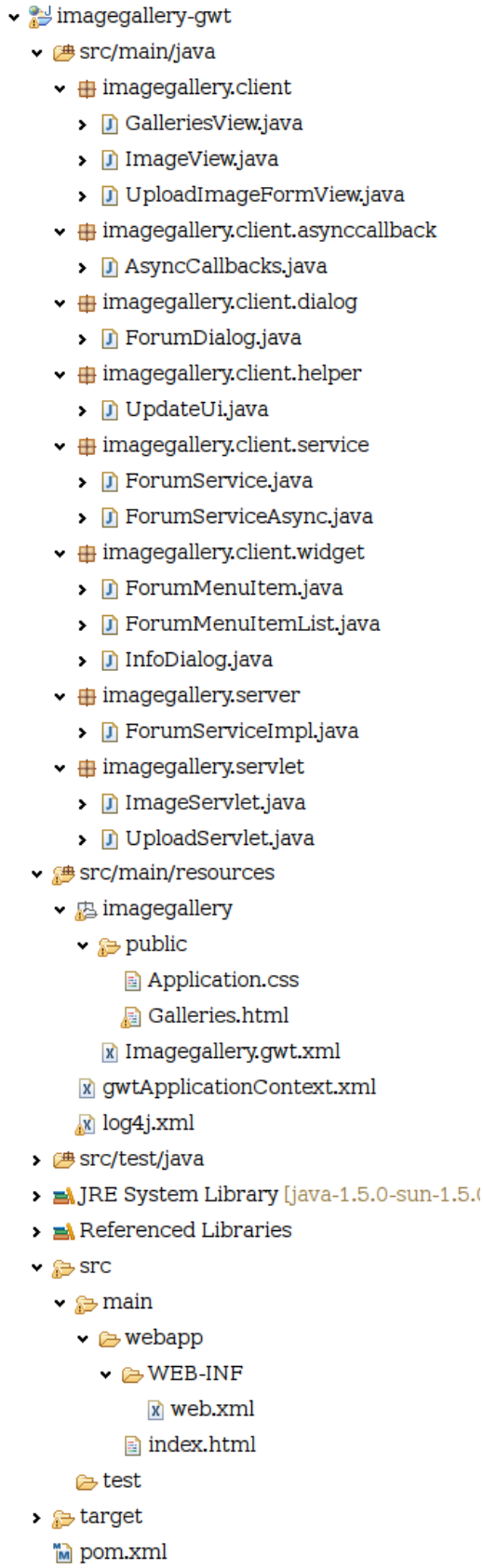
5.1.3 GWT-toteutus

Google Web Toolkit-projektien kääntämiseen ja luomiseen Mavenilla löytyi Maven plugin, jolla molemmat onnistuivat. Pluginin nimi on gwt-maven-plugin, lisätietoa pluginista ja sen käyttöönotosta löytyy osoitteesta http://mojo.codehaus.org/gwt-maven-plugin/. Projektirungon luominen onnistui komennolla:

```
mvn archetype:generate
-DarchetypeGroupId=org.codehaus.mojo
-DarchetypeArtifactId=gwt-maven-plugin
-DarchetypeVersion=1.1-SNAPSHOT
-DarchetypeRepository=http://snapshots.repository.codehaus.org/
-DgroupId=imagegallery -DartifactId=imagegallery-gwt
```

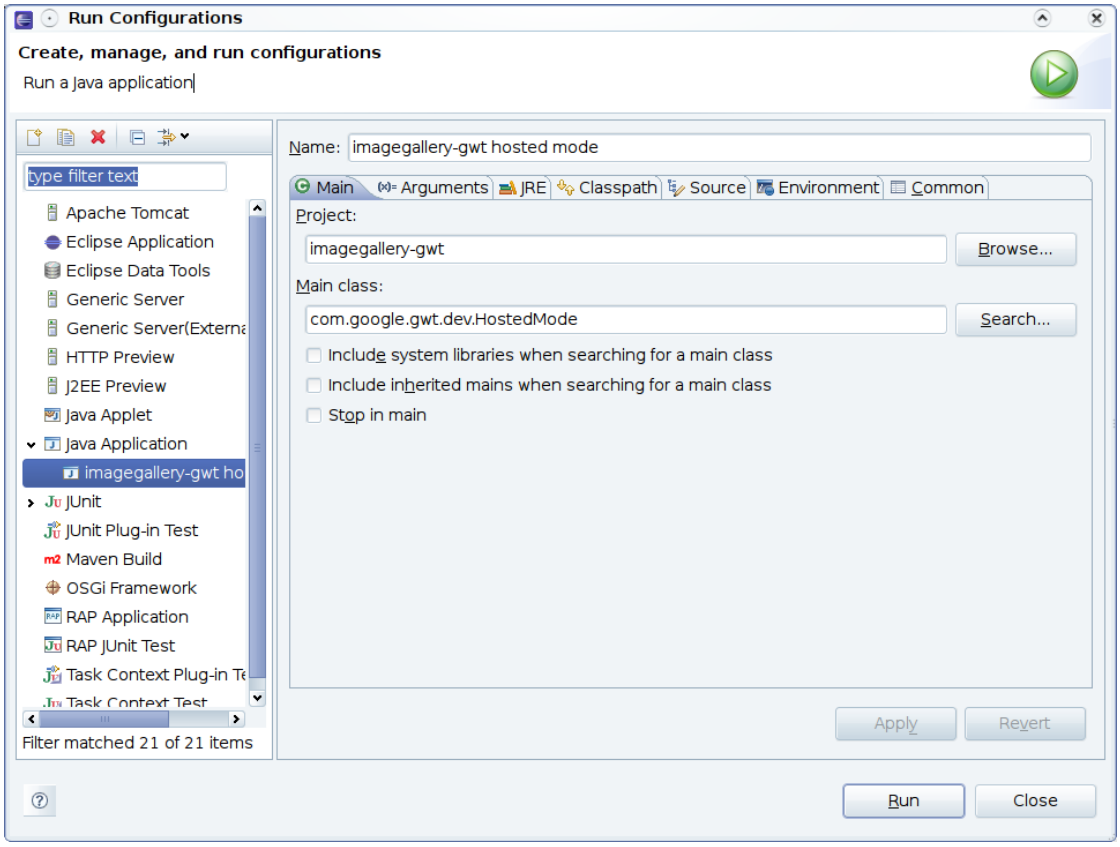
Generoidussa projektirungossa on käyttöliittymäluokkia varten vain yksi paketti, imagegallery.client. Sen rinnalle luotiin käsin kaksi muuta pakettia: imagegallery.server ja imagegallery.servlet, joista toiseen tuli kuvien palvelimelle lähettämiseen käytetty servletti sekä kuvapyyntöjä uudelleenohjaava servletti. Toiseen pakettiin tulivat palvelinpyyntöjä varten toteutetut service-luokat. Projektin rakenne esitellään kokonaisuu-

dessaan kuviossa 12. GWT-moduulien konfiguraatiot toteutetaan xml-tiedostoilla, joiden nimien tulee olla muotoa projektinimi.gwt.xml. Tässä projektissa konfiguraatiotiedoston nimi oli siis Imagegallery.gwt.xml. Konfiguraatio tiedosto on liitteenä 4. Konfiguraatiossa määritellään GWT-moduulin asetukset, kuten ohjelman käynnistävä luokka (entry-point), moduulin tarvitsemat resurssit sekä muut moduulit joita tarvitaan. GWT:n käyttämät servletit voitaisiin myös konfiguroida moduulin xml-tiedostoon, mutta tässä projektissa ne konfiguroitiin web.xmlään, joka on liitteenä 5.

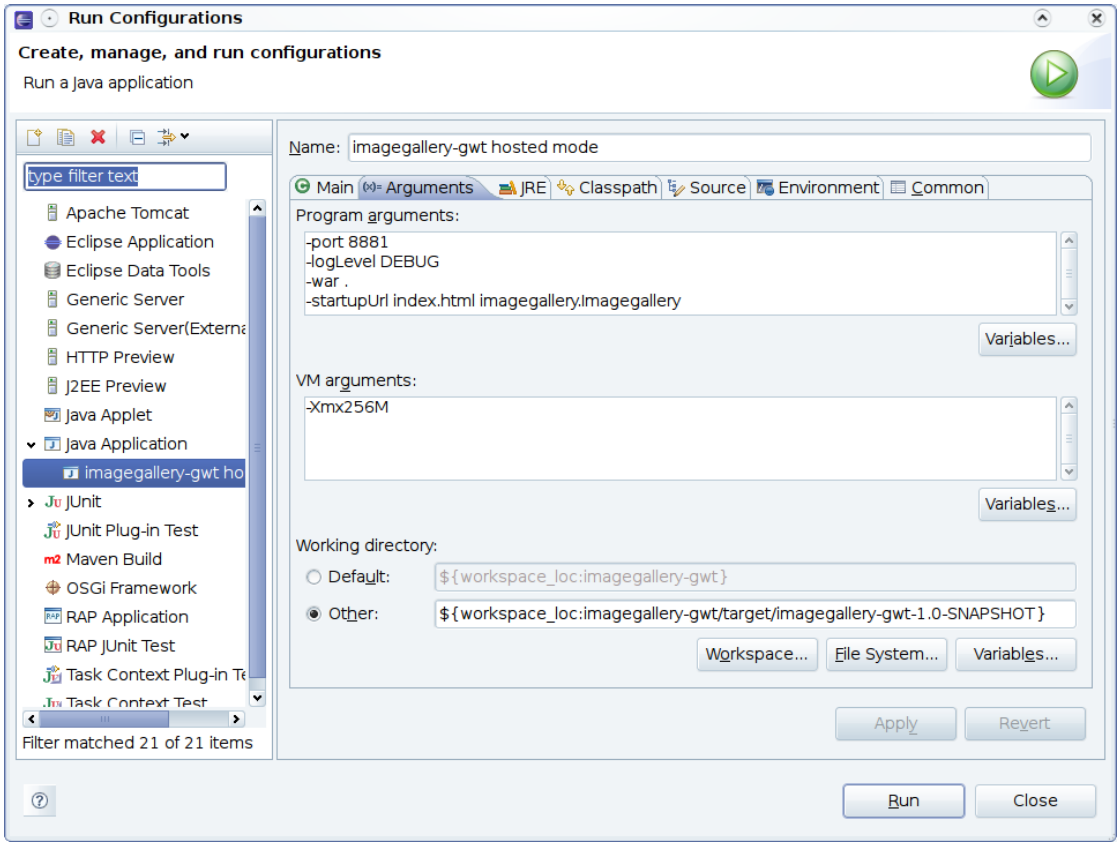


KUVIO 12: GWT-projektin rakenne

Projektissa käytetty konfiguraatio löytyy projektin pom.xml:stä, joka on liitteenä 6. Plugin-konfiguraation lisäksi pom.xmlään täytyi määritellä ANT-taski, joka kopioi GWT-kääntäjän tuotokset oikeaan kansioon. GWT:n omien työkalujen käyttäminen, kuten hosted mode browser, saatiin myös toimiaan Eclipsen kautta luomalla uusi projektin suorituskonfiguraatio. Kuvioissa 13 ja 14 esitetään main- ja arguments-välilehdille tulevat konfiguraatiot. Näiden konfiguraatioiden lisäksi tulee tarvittavat riippuvuudet lisätä classpath-välilehdeltä classpathiin.



KUVIO 13. GWT-hosted moden suorituskonfiguraation main-välilehti



KUVIO 14. GWT-hosted moden suorituskonfiguraation arguments-välilehti

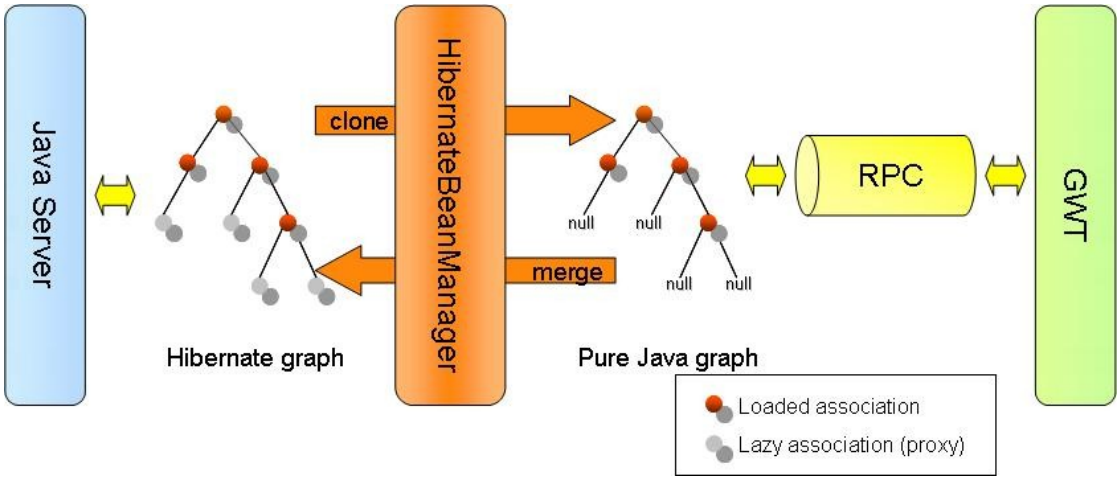
Toteutuksen edetessä törmättiin yhteen isompaan ongelmaan, jonka takia datan lataaminen tietokannasta ja sen tallentaminen takasin ei onnistunut. Ongelma johtui siitä, että ladatut model-oliot jouduttiin lähettämään JVM:n ulkopuolelle GWT-asiakasohjelmalle, jolloin GWT ei osannut käsitellä modeleita, joissa oli persistent-tietoa. Ongelmaan oli kuitenkin olemassa jo valmiiksi ratkaisu vaihtoehtoja. Aluksi ongelmaa yritettiin ratkaista käyttämällä Gilead-nimistä projektia, joka on suunniteltu yleiskäyttöiseksi komponentiksi RIA-projekteille model-olioiden persistent-tietojen poistamiseksi ja niiden tallentamiseen väliaikaisesti muualle.

Gilead tarjoaa kolme erilaista konfiguraatio mahdollisuutta: statefull-, stateless- ja dynamic proxy-mode. Stateless- ja dynamic proxy-modet eivät vaadi model-luokkien muokkausta kuitenkin edellyttäen, että model-luokat ovat serialisoitavissa. Kolmas tapa, statefull mode, vaatii että model-luokat periytyvät Gileadin tarjoamasta LightEntity-luokasta. Tätä vaihtoehtoa ei haluttu ottaa käyttöön, koska silloin kaikki muut teknologiat olisivat myös olleet riippuvaisia Gileadistä. Stateless-modea käytettäessä tal-

lennetaan persistent-tiedot LightEntity-luokan muuttujiin ja näin ollen palvelin ei tiedä käyttöliittymän tilasta mitään. Statefull-modessa taas persistent-tiedot tallennetaan palvelimelle ja näin ollen GWT-asiakasohjelma on riippuvainen palvelimesta, josta tiedot on ladattu. Dynamic proxy-mode on laajennus stateless-modelle, jolla voidaan saavuttaa sekä stateless-moden että statefull-moden edut. Dynamic proxy mode vaatii tosin sen, että model-luokat toteuttavat Serializable-rajapinnan, jota käytettään merkkinä GWT-generaattorille.

Projektin yritettiin saada toimimaan Gilead dynamic proxy-modessa, mutta konfiguraation jälkeen ilmeni, että se ei toimi vielä täysin kaikissa tapauksissa classloader-bugien takia. Ongelmaa yritettiin selvittää Gileadin pääkehittäjän Bruno Marchessonin kanssa Gileadin keskustelupalstan välityksellä, mutta ongelmaa ei saatu ratkaistua. Tämän takia jouduttiin turvautumaan Hibernate4Gwt-projektiin, jonka pohjalta Gilead on tehty. Käytännössä Gilead on vain uudempi versio Hibernate4Gwt:stä, mutta sitä on laajennettu paljon, jotta se toimii myös GWT:n lisäksi muiden RIA-teknologioiden kanssa. Hibernate4Gwt:stä otettiin käyttöö versio 1.1.1, joka ei virallisesti tue GWT:n versiota 1.6. Konfiguraation jälkeen kuitenkin ilmeni, että myös uusin versio GWT:stä toimii hibernate4gwt:llä, joten GWT:n versiota ei tarvinnut vaihtaa vanhempaan.

Hibernate4Gwt ja Gileadin käyttöönotto vaatii suhteellisen monen riippuvuuden lisäämistä projektiin. Tästä syystä tuleekin olla tarkkana riippuvuuksien versioiden kanssa, sillä oikeita versionumeroita ei ilmoiteta kummankaan projektin sivuilla. Osa riippuvuuksien versioista muuttuvat jokaisessa Hibernate4gwt/Gileadin julkaisussa, joten on tärkeää muistaa päivittää kaikki riippuvuudet samalla kun päivittää Hibernate4gwt/Gileadin versiota. Monesti ongelmat käyttöönoton yhteydessä johtuvatkin juuri vääristä riippuvuuksista. Itse konfiguraatio on helppo sillä Hibernate4gwt/Gilead tarjoaa yksityiskohtaiset ohjeet eri konfiguraatioiden tekemiseen. Kuvion 15 esittää Hibernate4Gwt ja Gileadin toimintaperiaatetta.



KUVIO 15. Hibernate4Gwt/Gilead toimintaperitaate (Hibernate4Gwt 2008).

GWT-tarjoaa melko laajan komponenttikirjaston, mutta komponentit ovat yksinkertaisia, eikä niihin ole toteutettu valmiiksi juurikaan mitään erikoisempia toimintoja. Komponenttien laajentaminen ja toiminnallisuuksien lisääminen onnistuu kuitenkin helposti. Toteuttamalla itse komponenttien lisäominaisuudet saadaan niistä juuri omiin tarpeisiin sopivia komponentteja. Tämä hieman hidastaa kehitystä, mutta samalla mahdollistaa erinomaisesti komponenttien räätälöinnin. GWT:lle löytyy myös kolmansien osapuolien tekemiä komponenttikirjastoja, kuten ExtGwt. Näiden komponenttikirjastojen avulla kehitys on nopeampaa, mutta komponentteja ei välttämättä pysty räätälöimään yhtä hyvin omiin tarpeisiin sopiviksi.

Käyttäjän syötteen validointi pystyttiin suorittamaan kätevästi erityyppisten Handlereiden avulla, jotka kiinnitettiin syötekenttiin tai painikkeisiin. Esimerkiksi jokainen kenttä voitiin validoida aina kun kenttä menetti fokuksensa BlurHandlerin avulla. Koko lomakkeen validointi taas onnistui lomakkeen lähetysnappiin kiinnitetyn ClickHandlerin avulla. Myös täysin reaaliaikainen validointi oli mahdollista toteuttamalla ChangeHandler ja kiinnittämällä se validoitavaan komponenttiin, jolloin aina kun komponentin arvo muuttui validointi suoritettiin.

5.1.4 Echo-toteutus

Echo Frameworkin kolmas versio mahdollistaa asiakasohjelman sekä palvelinpuolella toimivien käyttöliittymien toteutukset. Palvelinpuolella toimiva käyttöliittymä toteutetaan Java-kielellä, kun taas asiakasohjelman puolella toimiva käyttöliittymä toteutettaisiin JavaScriptillä. Tästä syystä toteutusvaihtoehdoksi valittiin palvelinpuolella toimiva käyttöliittymä, sillä asiakasohjelman puolella toimivan käyttöliittymän toteutus olisi koskenut enemmän JavaScript-sovelluskehysten vertailua.

Pohjaksi projektille luotiin Mavenilla yksinkertainen Web-projekti, jossa ei ollut mitään ylimääräisiä riippuvuuksia. Projektiin toteutettiin Echo Frameworkin vaatimat luokat, jotka perivät ApplicationInstance- ja WebContainerServlet-luokat. ApplicationInstance-luokan täytyy ylikirjoittaa init-metodi, joka palauttaa Window-tyyppisen olion, eli ohjelman ”ikkunan”. WebContainerServlet-luokkaa taas käytetään Echo-ohjelman käynnistämiseen ja siinä tulee ylikirjoittaa newApplicationInstance-metodi. Ylikirjoitettu metodi palauttaa olion luokasta, joka perii ApplicationInstance-luokan. Käyttöliittymän toteuttaminen Echo Frameworkilla onnistui erittäin suoraviivaisesti eikä mihinkään suurempiin ongelmiin törmätty. Mavenin käyttö projektissa onnistui ongelmitta ja vaikkei Echo Frameworkin kehittäneellä NextAppilla olekaan tarjota Maven-repositorya, saatiin Echo jar-paketit Mavenin käyttöön asentamalla ne paikalliseen repositoryyn. Myös integraatio Spring Frameworkin kanssa saatiin toimimaan täysin ongelmitta. Spring Frameworkin Java-pavut voidaan tarjota ohjelmalle esimerkiksi WebContainerServlet-luokasta periytyvän Servletin kautta, jossa pavut haetaan ServletContextista. Sovelluksen pom.xml on liitteenä 7 ja web.xml liitteenä 8. Debuggaus onnistuu saman tapaan kuin minkä tahansa muun web-projektin debuggaus.

Käyttöliittymän toteuttaminen muistutti paljon käyttöliittymän toteuttamista Swing- tai AWT-kirjastoilla. Uskoisinkin että henkilölle, joka on käyttänyt jompaa kumpaa kirjastoa käyttöliittymien toteuttaminen Echolla on erittäin helppoa. Kaikille klikattaville komponenteille tuli rekisteröidä ActionListenerit ja näissä määritellä mitä klikkauksesta tapahtui. Näin ollen kaikki tiedon tallennus pystyttiin toteuttamaan suoraan service-rajapintojen kautta ilman, että tietoja tarvitsi lähettää erikseen millekään servletille. Myös lähetettävien tietojen validointi pystyttiin suorittamaan ActionListenerien metodeissa tai vaihtoehtoisesti PropertyChangeListenerien metodeissa, jolloin kent-

tien validointi onnistui reaaliaikaisesti. Echo Frameworkin Listenerit vastaavat GWT:n Handlereitä. Suurin puute mihin törmättiin käyttöliittymää toteutettaessa oli CSS-tuen puuttuminen. Echo-sovelluksien tyylit määritellään joko Java-koodissa MutableStyle-luokan avulla tai vaihtoehtoisesti käyttämällä XMLää tyylien määrittelyyn. XML-tyylitiedoston syntaksi ei muistuta millään tapaa CSSän syntaksia. Kuviossa 16 esimerkki tyylien määrittelystä XML:llä. Esimerkki määrittelee napin tyylit.

```
<?xml version="1.0" encoding="UTF-8"?>
<ss>
 <s n="Default" t="nextapp.echo.app.Button">
  <p n="background" t="Color">#6f87af</p>
  <p n="foreground" t="Color">#bcbcbc</p>
  <p n="pressedbackground" t="Color">#fafafa</p>
  <p n="border" t="Border">2px groove #afffaf</p>
 </s>
</ss>
```

KUVIO 16. Echo Frameworkin esimerkki tyylitiedostosta

Echo Frameworkin peruskomponenttikirjasto on melko suppea, mutta NextApp on kehittänyt myös kolme laajennuskirjastoa: Extra, Chart ja File transfer. Extra-kirjastosta löytyy yleishyödyllisiä komponentteja kuten CalendarSelect-komponentti, joka mahdollistaa päivämäärän valinnan kalenterista. Chart-kirjastosta mahdollistaa tiedon esittämisen kaavioina JFreeChart-kirjastoa hyödyntäen. File transfer-kirjasto tarjoaa komponentteja, jotka mahdollistaa tiedostojen lähettämisen palvelimelle ja niiden lataamisen sieltä.

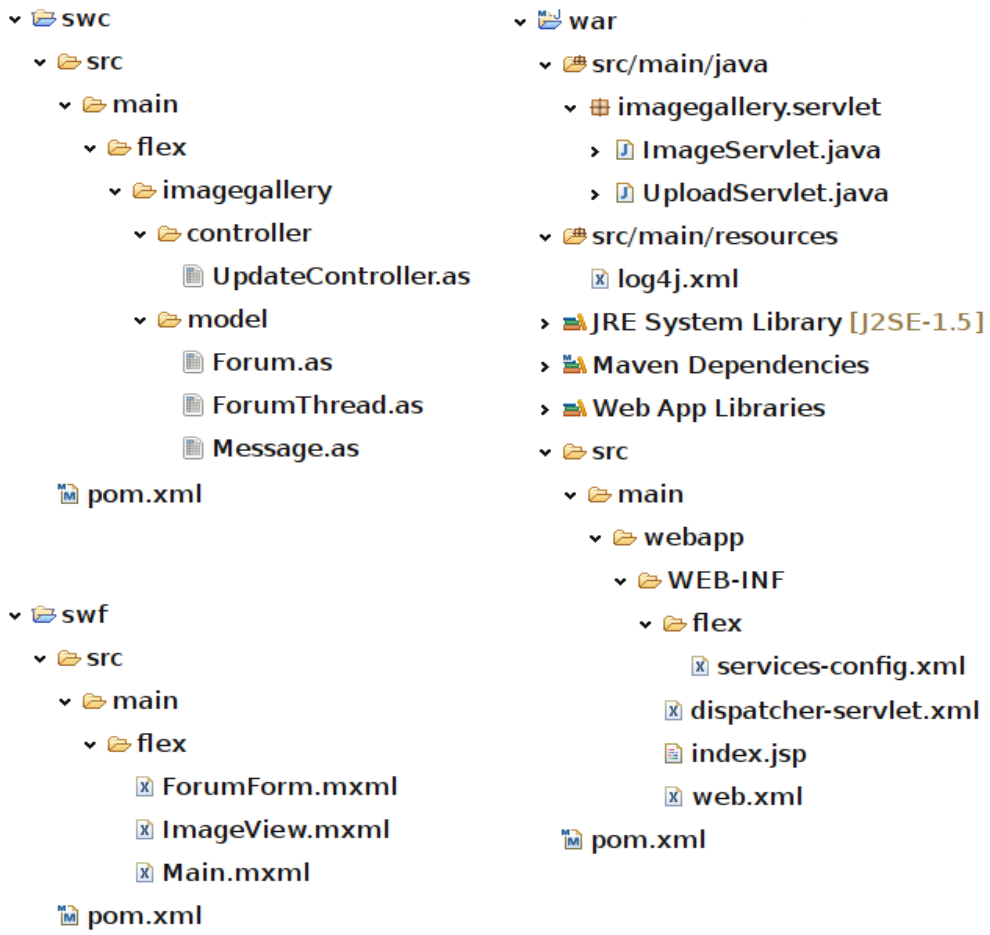
5.1.5 Adobe Flex-toteutus

Kuten muidenkin teknologioiden kohdalla, Flex-sovelluksen toteutus aloitettiin tutkimalla mahdollisuuksia käyttää Mavenia projektin kääntämiseen. Jo pienen etsimisen jälkeen löytyikin Sonatypen kehittämä Maven plugin, joka mahdollisti MXML- ja ActionScript-tiedostojen kääntämisen Mavenillä. Pluginin nimi oli flexmojos-maven-plu-

gin. Sonatype tarjosi myös kolme erilaista archetypeä Flex-projektien luontiin: Flex-kirjastoprojektille, yksinkertaiselle Flex-sovellukselle sekä modulaariselle Flex-sovellukselle. Käytettäväksi valittiin modulaarisen projektin luova archetype. Maven-komento, jolla projekti luotiin oli seuraava:

```
mvn org.apache.maven.plugins:maven-archetype-plugin:generate
-DarchetypeRepository=http://repository.sonatype.org/content/groups/p
ublic
-DarchetypeGroupId=org.sonatype.flexmojos -DarchetypeArtifactId=flex-
mojos-archetypes-modular-webapp -DarchetypeVersion=3.1.0
```

Archetype loi projektin, jolla oli kolme moduulia: swc-, swf- ja war-moduulit. Moduuleiden tarkoituksena on selventää projektinrakennetta, moduulien yksityiskohtaiset rakenteet esitetään kuviossa 17. Swc-moduuliin kirjoitettiin kaikki ActionScript-koodit, pois lukien MXML-tiedostoihin <script>-tagien sisään kirjoitetut ActionScript-koodit. Kääntäessä swc-moduulin paketoitiin .swc-tiedostoksi, jolloin sitä voitiin käyttää swf-moduulissa kirjastona. Swf-moduuliin toteutettiin MXML:llä sovelluksen näkymät. Swf-moduuli paketoitiin .swf-tiedostoksi, eli flash-toistimessa toimivaan muotoon. War-moduulia taas käytettiin integraation toteuttamiseen sekä web-sovelluksen rakentamiseen Flex-sovelluksen ympärille, jotta sitä voitiin ajaa Java-sovelluspalvelimella. Projektin luonnin yhteydessä eri modulien pom.xml -tiedostoihin generoitiin valmiiksi plugin-konfiguraatiot projektien kääntämistä varten, eli muuta konfigurointia Maven-integraatioon ei tarvinnut tehdä. Moduulien pom.xml:ät ovat liitteinä; swc-moduulin pom.xml liitteenä 9, swf-moduulin pom.xml liittenä 10, war-moduulin pom.xml liittenä 11 sekä projektin pää-pom.xml liitteenä 12.



KUVIO 17. Flex-projektin moduulien rakenteet.

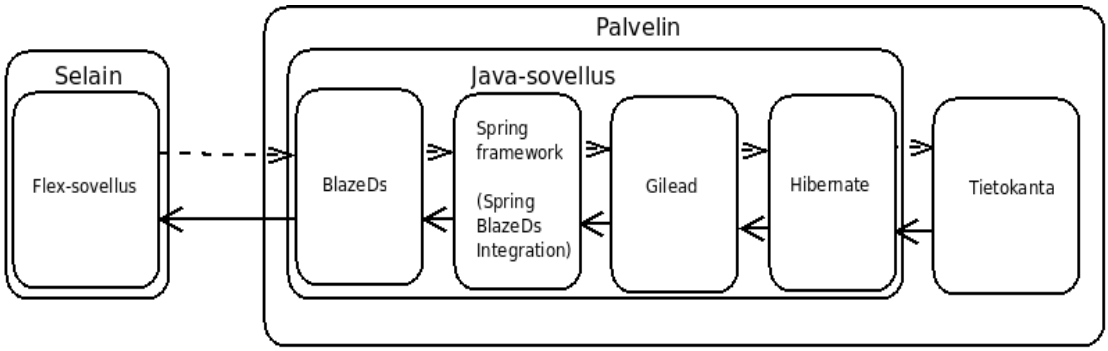
Seuraavaksi Eclipse-kehitysympäristöön ladattiin Adoben tarjoama Flex-builder pluginin kokeiluversio. Adobella ei kuitenkaan ollut tarjota Linux-käyttöjärjestelmille kuin Flex neljälle tarkoitettu ALPHA-tason versio Flex-builderista, joka kuitenkin soveltui Flex kolme sovellusten kehitykseen. Pluginin varhaisesta kehitysvaiheesta johtuen se ei kuitenkaan vielä toiminut kunnolla Eclipsen 3.4-version kanssa. Parhaimmillaan plugini saatiin värjäämään ActionScript-tiedostojen koodi, mutta enempää hyötyä siitä ei saatu. MXML-tiedostoja ei pystytty edes avaamaan käyttämällä Flex-builder-pluginiä vaan niiden toteuttamiseen jouduttiin käyttämään Eclipsen teksti-editoria, joka hankaloitti kehittämistä huomattavasti. Kehitysympäristön puutteita pystyttiin kuitenkin jonkin verran paikkaamaan Adoben Flex-dokumentaation avulla, joka oli erittäin kattava. Dokumentaatiosta löytyi niin luokkien API-rajapinnat kuin runsaasti ohjeita erilaisten toimintojen toteuttamiseen. Käyttöliittymän toteuttaminen sujuikin suoraviivaisesti ja ilman suurempia ongelmia, kehitysympäristön puutteista

huolimatta. Suurimmat hankaluudet kohdattiin integroitaessa Flex-käyttöliittymää Spring Frameworkin ja hibernaten kanssa.

Adobe Flex käyttää SOA-konseptin (Service-oriented architecture) komponentteja tiedonsiirtoon asiakasohjelman ja palvelimen välillä ja tarjoten siihen kolme eri tyylistä lähestymistapaa: REST-tyyliset palvelut, SOAP web-service ja remote-object service. REST-tyylisissä (Representational State Transfer) palveluissa Flex-sovellus lähettää joko GET- tai POST-tyylisen palvelinpyynnön palvelimelle, esimerkiksi JSP-sivulle, joka lähettää vastauksen joko XML- tai HTML-muodossa. Flex-sovelluksessa käytetään HTTPservice komponenttia sivupyynnön suorittamiseen. Myös SOAP (Simple Object Access Protocol) web-service tyyppinen tiedonsiirto on REST-tyyppistä, mutta silloin sivupyyntö lähetetään suoraan SOAP-päätepisteeseen (Endpoint), josta lähetetään takaisin XML-tyyppinen vastaus. Remote-objet service-tyylisessä tiedonsiirrossa siirrettävä data serialisoidaan ja sen jälkeen siirretään asiakasohjelman ja palvelimen välillä. Tiedonsiirtotavaksi valittiin remote-object service-tyylinen tiedonsiirto. Jotta tiedonsiirto toimisi, palvelinpuolelle tuli konfiguroida käyttöön BlazeDs-teknologia, joka loi päätepisteet palvelinpyynnöille. BlazeDs on teknologia, joka on Adoben kehittämä palvelinpuolen teknologia Flex- ja Air-sovelluksia varten. Lisätietoa BlazeDs:tä löytyy osoitteesta: http://opensource.adobe.com/wiki/display/blazeds. BlazeDs:n konfiguraatio pystyttiin hoitamaan täysin Spring BlazeDs integration -komponentin avulla, joka on toteutettu helpottamaan Spring Frameworkin ja blazedsän integraatiota ja tekemään blazeds:n konfiguroinnista samankaltaista kuin Spring Frameworkin konfigurointi on.

Spring BlazeDs integration -komponentin versio 1.0.0 julkaistiin 10. heinäkuuta 2009, eli vain muutama viikko ennen kuin Flex-toteutusta aloitettiin tekemään. Tästä syystä sen dokumentaatio oli vielä Flex-toteutuksen alkuvaiheissa hieman heikko, mutta se kuitenkin kehittyi nopeasti hyvälle tasolle. Verrattuna Spring BlazeDs integration aiempiin versioihin oli versio 1.0.0 kokenut suuria muutoksia ja siinä konfigurointi tapahtui siinä määrin eri tavalla, ettei aiempien versioiden dokumentaatioita voitu hyödyntää. Kun dokumentaatio oli parantunut hieman huomattiin, että integraation konfigurointi oli erittäin yksinkertaista ja nopea toteuttaa. Kaiken kaikkiaan konfigurointia tarvitsi tehdä kolmeen tiedostoon: web.xml, dispatcher-servlet.xml ja services-config.xml. Integraatioon käytettyjen komponenttien kutsujärjestys selviää kuviosta 18.

Aluksi web.xml:n tuli konfiguroida Spring Frameworkin dispatcher-servlet, joka ohjaisi pyynnöt BlazeDs:lle, web.xml on liitteenä 13. Tämän jälkeen dispatcher-servletille piti lisätä konfiguraatio Hibernate adapterille, jona käytettiin GWT-projektissa mainitun Gilead-komponentin tarjoamaa PersistentAdapteria. Adapteri asetettiin BlazeDsän message-brokerille oletus adapteriksi. BlazeDs:n message-broker toimii päätepisteiden ja service-tasojen välillä ohjaimena, joka ohjaa päätepisteisiin tulleet pyynnöt oikealla servicelle ja serviceltä tulevat vastaukset oikeaan päätepisteeseen. Viimeinen konfiguraatio dispatcher-servlettiin oli service-beanien altistaminen BlazeDs:lle käyttäen <flex:remoting-destination /> -tagia. Dispatcher-servletin konfiguraatiot toteutettiin dispatcher-servlet.xml -tiedostoon, joka on liitteenä 14. Services-config.xml:n konfiguroitiin aluksi kanavat ja niille päätepisteet, tässä tapauksessa kanavia ja päätepisteitä tarvittiin vain yksi. Päätepiste oli dispatcher-servletin osoite plus vapaasti valittu nimi, amf ja kanavan nimeksi konfiguroitiin my-amf. My-amf-kanava asetettiin oletuskanavaksi, jolloin sitä käytettäisiin aina ellei toisin määrättäisi. Näiden konfiguraatioiden lisäksi services-config.xml:n konfiguroitiin ylimääräisenä BlazeDs yhteyksille logitus, jotta virheet löydettäisiin helpommin kehityksen aikana. Services-config.xml on kokonaisuudessaan liitteenä 15.



KUVIO 18. Flex-sovelluksen integraatio Java-palvelimeen

Tässä vaiheessa Flex-sovellusta kehitettiin sen verran, että integraatiota pystyttiin koekäyttämään. Tuloksena kuitenkin oli ettei integraatio vielä toiminut täysin, vaan Flex-sovellukselle palautettiin tyhjiä olioita. Pienen lisäselvittelyn jälkeen huomattiin, että

Flex-sovellukseen oli toteutettava Java-modeleita vastaavat luokat ActionScriptillä ja niiden, kuten myös Java-modeleiden, tuli periytyä Gileadin tarjoamasta LightEntity-luokasta. Java-modeleiden periytymisongelmaan yritettiin etsiä samantapaista ratkaisua mitä GWT-projektissa käytetiin, mutta tuloksetta. Yksi vaihtoehto olisi ollut toteuttaa Flex-sovelluksen war-moduuliin DTO-oliot (Data Transfer Object), joihin modeleiden tiedot olisi siirretty aluksi ja sen jälkeen lähetetty ne Flex-sovellukselle. Projektin luonteen ja pienen koon takia päätettiin kuitenkin periyttää modelit LightEntity-luokasta, jotta kehitystä päästäisiin jatkamaan nopeasti. ActionScriptillä toteutettujen model-luokkien vastineiden ja model-luokkien periyttämisen jälkeen tiedonsiirto palvelimen ja Flex-asiakasohjelman välillä toimi täysin.

MXML-tiedostojen tuottaminen oli helppoa ja siinä käytettyjen tagien nimet olivat hyvin kuvaavia ja useat olivat tuttuja HTML-tageja. MXML tukee nimiavaruuksia, joten siihen pystyttiin tuomaan itse ActionScriptillä luotuja tageja. Omia tageja voitiin ottaa käyttöön määrittelemällä niille oma nimiavaruus sekä paketti josta ne löytyvät. ActionScriptiä käytettiin pääasiassa sovelluksen logiikan toteutukseen, esimerkiksi nappuloiden toimintojen toteuttamiseen. Sillä olisi voitu toteuttaa myös omia tageja jos tällaiselle olisi ollut tarvetta. ActionScriptistä pystyttiin kutsumaan suoraan MXML:ään määriteltyjä tageja. Tämä helpotti logiikan toteutusta paljon, osa käytetyistä ominaisuuksista oli helpompi ja nopeampi määritellä tageja käyttämällä mitä luomalla uusia olioita ActionScriptissä. Tageihin pystyttiin viittaamaan käyttämällä tagille annettua id:tä. ActionScriptin syntaksi muistutti paljolti Javan syntaksia, mutta eroavaisuuksiin törmättiin nopeasti. Pieni eroavaisuus syntakseissa aiheutti aluksi pientä pään vaivaa, mutta kun erot sisäisti myös ongelmat hävisivät.

Flex-sovelluksia pystyttiin tyylittämään käyttämällä CSS-määrityksiä. Määritykset voitiin ladata joko ulkoisesta tyylitiedostosta <style>-tagilla, jolle annettiin source-attribuuttiin tiedoston sijainti tai määrittelemällä sovelluksen sisäiset CSS-tyylit. Tyylien <style>-tagi tuli aina sijoittaa heti päätason tagista seuraavaksi, eli MXML-tiedoston alkuun. CSS-määritykset toimivat samalla tavalla Flex-sovelluksissa mitä muissa sovelluksissa, jotka tukevat CSS:ää. Suurin osa CSS-määrityksistä tuki myös periytymistä tagi-hierarkiassa suoraan lapsielementeille päin. Tageille pystyttiin asettemaan sekä id- että styleClass-attribuutit, joista id-attribuutti toimi samalla tapaa kuin HTML-tagien id ja styleClass-attribuutti taas määritteli elementille HTML-tageilta tu-

tun class-attribuutin. Tyylimäärityksiä voitiin myös asettaa MXML:ssä käytetyille tageille suoraan esimerkiksi näin <mx:Button value=”Click me” font=”arial” />. Tällä tapaa tyylejä määriteltäessä attribuuttien nimet kuitenkin hieman poikkesivat CSS-määrityksen mukaisista nimeämisistä. Flex-mahdollisti myös vielä yhden eri tavan altistaa elementtejä css-tyyleille, tätä tapaa käytettiin ActionScriptissä. Jokaiselle visuaalisen elementin tuottavalla luokalla oli määritelty setStyle(”style name”, ”value”)-metodi, jolla tyylit asetettiin.

5.1.6 Java2Script PaceMaker-toteutus

Java2Script PaceMaker-toteutuksen ollessa vielä alkuvaiheessa huomattiin puutteita kehitysympäristön tuen suhteen sekä kohdattiin ongelmia integraation kanssa. Nämä syyt johtivat siihen, että Java2Script PaceMaker-toteutus päätettiin jättää tekemättä. Projektissa ei voitu käyttää Mavenia ollenkaan, koska kääntämiseen täytyi käyttää Java2Script PaceMaker-yhteisön tarjoamaa Eclipsen laajennusta, joka toimii pelkästään Eclipsestä käytettynä. Tämä ei vielä kuitenkaan haitannut suuremmin kehitystyötä sillä Eclipsen laajennus loi projektin luonnin yhteydessä ANT-scriptin WAR-paketin luontia varten. Scripti loi WAR-paketin Eclipsen luoman JAR-paketin pohjalta ja lisäsi siihen web-projektiin kuuluvat resurssitiedostot. Integrointiongelmat perusprojektin kanssa aiheuttivat teknologian hylkäämisen. Integrointi saatiin toimimaan osittain, eli pystyttiin suorittamaan tietokantahakuja. Hakujen tuloksia ei kuitenkaan onnistuttu välittämään käyttöliittymälle asti onnistuneesti. Ongelmaa selvitettäessä huomattiin myös, ettei Java2Script PaceMaker-yhteisö ollut enää aktiivinen. Viimeisin viesti yhteisön keskustelupalstalla oli jätetty noin vuosi sitten ja viimeisin kehittäjien jättämä uutinen oli julkaistu joulukuussa 2008. Näyttäisikin siltä ettei kyseistä teknologiaa enää kehitettäisi ollenkaan.

Java2Script PaceMakerillä kuitenkin toteutettiin pieni teknologiademo, jotta teknologiaan pystyttiin tutustumaan paremmin. Sovelluksen ulkoasu määriteltiin Java-koodissa, mutta kääntäjä muutti tyylimääritykset CSS-määrityksiksi ja lisäsi ne ohjelman käynnistävään HTML-tiedostoon. Joten periaatteessa ohjelman ulkoasu on mahdollista määritellä pelkästään CSS:ää käyttäen. Käyttöliittymän itsensä toteutus tapahtui täysin samalla lailla mitä SWT-käyttöliittymän toteutus ja siinä käytettiinkin SWT-kirjastoa. Tottuneelle SWT-kirjastojen käyttäjälle Java2Script PaceMakerin käytön ei siis

pitäisi tuottaa mitään ongelmia. Vaikka itse en ole SWT:llä juuri mitään kehittänyt oli käyttöliittymän toteuttaminen suhteellisen helppoa. Java2Script PaceMaker-kääntäjään on implementoitu paljon GWT:tä suppeampi emulaatiokirjasto Javan peruskirjastoista ja tämä saattaakin osaltaan hankaloittaa kehitystä.

Java2Script vaatisi aktiivisen käyttäjäkunnan sekä elonmerkkejä kehityksen jatkumisesta jotta se olisi varteen otettava teknologia, törmätyistä ongelmista huolimatta. Teknologian hylkäämiseen johtanut ongelma olisi varmasti ollut selvitettävissä, mutta se olisi vaatinut liian paljon aikaa ja resursseja hyötyyn nähden, ilman yhteisöä, joka tuntee teknologian läpikotaisin.

5.2 Teknologioiden suorituskyky

5.2.1 Apache Jmeter ja suorituskykytestausten toteuttaminen

Jmeter on Apache Software säätiön kehittämä Java-pohjainen sovellus, joka on suunniteltu suorituskykytestien tekemiseen. Apache Jmeterin esittelyssä kerrotaan, että Sitä (Apache Jmeteriä) voidaan käyttää simuloimaan raskasta kuormitus palvelimelle, verkolle ja Java-olioille suorituskyvyn testaukseksi tai analysoida suorituskykyä erilaisissa rasitustilanteissa (Apache Jmeter n.d.). Yksinkertaisimmillaan Jmeterin käyttö websovelluksia testatessa tarkoittaa sitä, että sillä luodaan proxy-palvelin, jota käyttäen voidaan ”nauhoittaa” suoritettavat testit. Nauhoittaessa Jmeter tallentaa jokaisen palvelinpyynnön, joka kulkee luodun proxy-palvelimen lävitse. Yhden nauhoitussession aikana tallennetuista palvelinpyynnöistä muodostuu aina yksi testitapaus. Tallennettuja palvelinpyyntöjä voidaan kuitenkin muokata vielä ennen kuin testejä aletaan suorittamaan, esimerkiksi lisäämällä virheellisiä parametrejä. Nauhoituksen jälkeen testit ovat suoritettavissa Jmeterissä valmiina ja niille voidaan valita käyttäjämäärä sekä suorituskerrat per käyttäjä. Käyttäjämäärällä tarkoitetaan montako eri käyttäjää suorittaa testin ja suorituskerroilla tarkoitetaan monestiko yksi käyttäjä suorittaa yhden testin. Esimerkiksi jos testissä on viisi käyttäjää ja suorituskerroiksi asetetaan kaksi, suoritetaan testi yhteensä kymmenen kertaa.

Tässä opinnäytetyössä käytettiin yllä esitettyä tapaa suorittaa suorituskykytestäjä, ainoana erona oli se, että testitapauksiin lisättiin Aggregate report -niminen raportointikomponentti, joka esitti suorituskykytestien tuloksia taulukkona. Jmeterin (Apache Jmeter: aggregate report n.d.) käyttöoppaassa avataan aggregate reportin sarakkeet seuraavasti:

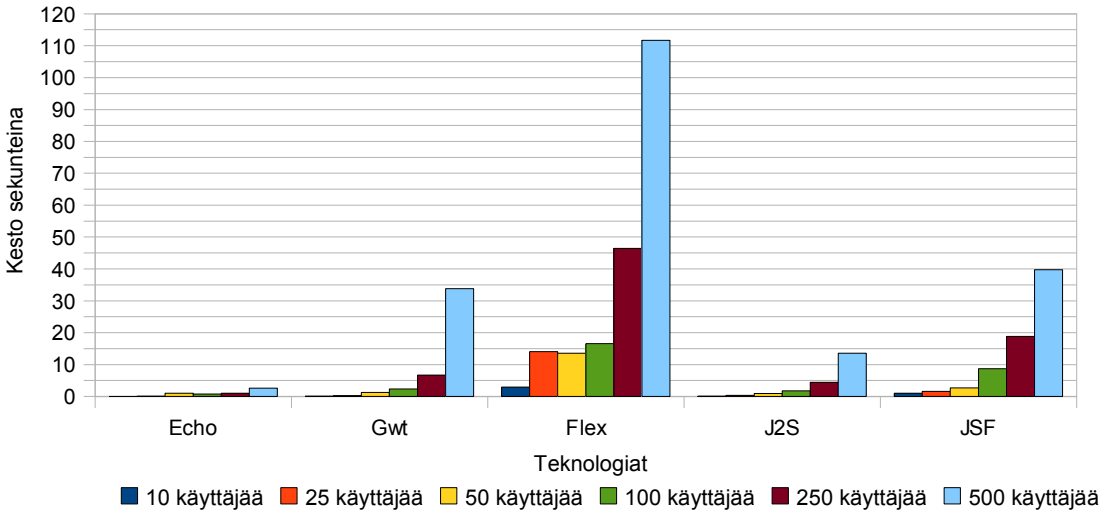
- Palvelinpyyntöjen nimet
- Palvelinpyyntöjen määrä
- Keskimääräisen suoritusaika per käyttäjä
- Mediaani suoritukseen käytetystä ajasta per käyttäjä
- Aika jonka 90% palvelinpyynnöistä alitti
- Pisin ja lyhyin suoritusaika
- Virheellisten vastausten prosenttimäärä
- Moneenko palvelinpyyntöön palvelin pystyi vastaamaan per sekunti/minuutti/tunti. Yksikkö valitaan siten, että tarkkuus on vähintään 1.0.
- Siirretyt kilotavut

Raportointikomponentti listasi erilaiset palvelinpyynnöt omille riveilleen, jolloin pystyttiin helposti seuraamaan sekä testien kokonaistietoja että yksittäisien palvelinpyyntöjen tietoja. Jmeter mahdollisti myös erilaisien raporttien koonnin kuten aggregate raportin pohjalta luotavan graafin tai yhteenvetoraportin. Näytteiden kelpoisuutta pystyttiin myös rajaamaan erilaisin rajoittein kuten esimerkiksi vastausten keston ja koon perusteella

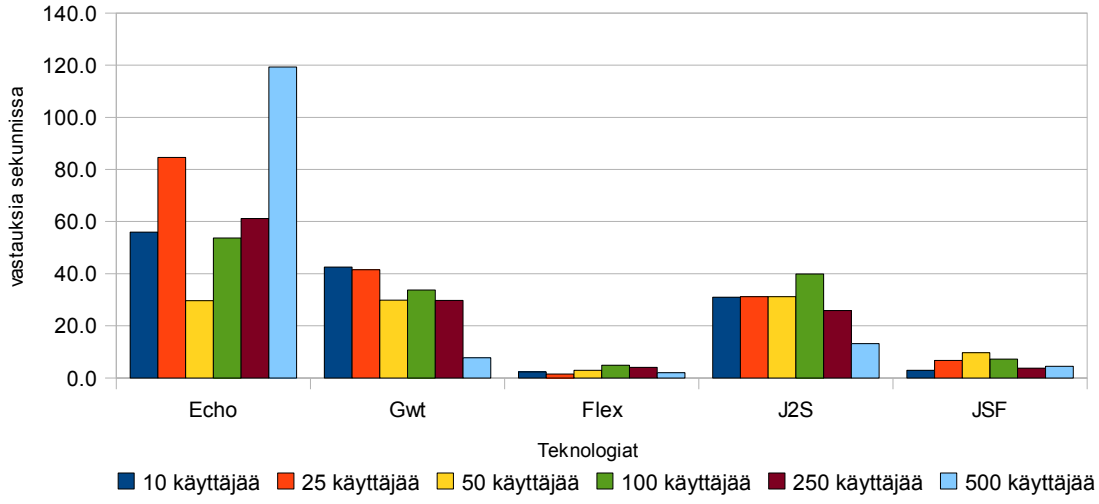
5.2.2 Suorituskykytulokset

Suorituskykytestausta varten Jmeterillä nauhoitettiin, jokaisella teknologialla kaksi erilaista testitapausta. Ensimmäinen testitapaus oli yksinkertainen sovelluksen etusivun lataus. Etusivun lataus suoritettiin myös Java2Script-teknologialla toteutetulle teknologiademolle, vaikkakin Java2Scriptin tuloksia ei voitu pitää kuin suuntaa antavina. Ensimmäisen suorituskykytestin tulokset esitellään kuvioissa 19, 20 ja 21. Toi-

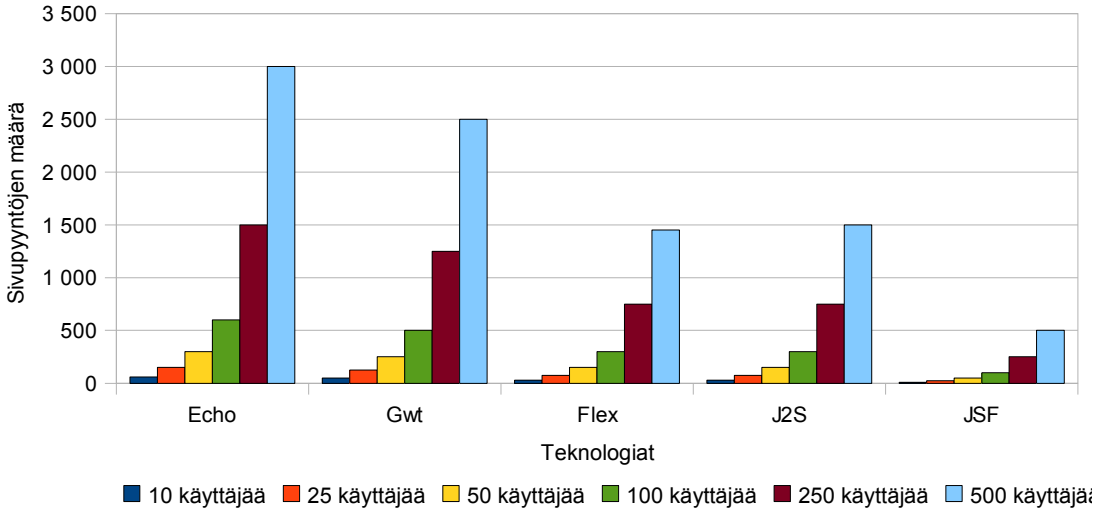
nen suorituskykytesti kuvasi hieman laajempaa sivustonselausta. Testissä ladattiin ensiksi etusivu jonka jälkeen avattiin ensimmäinen galleria ja sieltä yksi kuva ja sen jälkeen avattiin toinen galleria ja sieltä myös yksi kuva. Toisen suorituskykytestin tulokset esitellään kuvioissa 22, 23 ja 24. Ennen suorituskykytestien aloittamista tietokantaan syötettiin testidataa, joka säilyi muuttumattomana koko testaussession ajan. Näin pyrittiin lisäämään suorituskykytestien tuloksien luotettavuutta.



KUVIO 19. Ensimmäisen suorituskykytestin keston keskiarvot



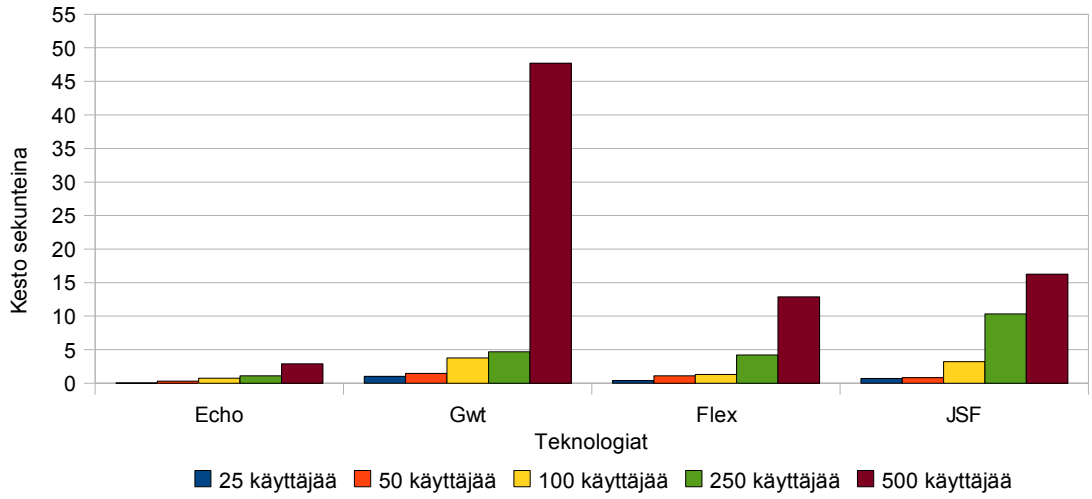
KUVIO 20. Palvelimen keskimääräinen vastauskyky ensimmäisessä suorituskykytestissä



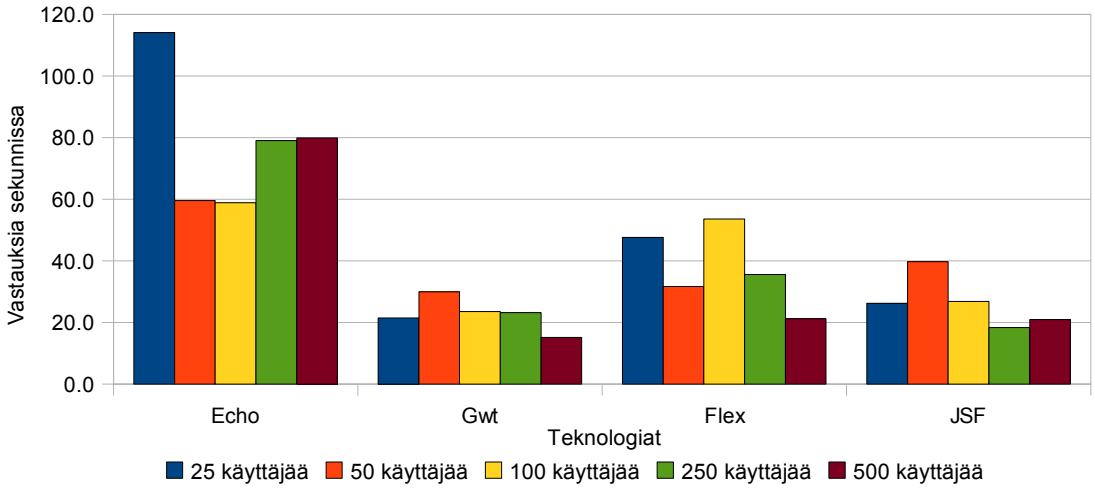
KUVIO 21. Näytteiden määrä ensimmäisessä suorituskykytestissä

Ensimmäisessä suorituskykytestissä ilmeni isoja eroja eri teknologioiden välillä, kun taas teknologia kohtaiset tulokset kehittyivät odotetusti käyttäjämäärän lisääntyessä. Etusivun latauksesta hitaimmin suoriutui Adobe Flex-teknologia, jolla kesti suoriutua testistä yli 110-sekuntia käyttäjämäärän ollessa 500. Vastaavasti taas nopeimmin Echo Framework, joka suoriutui testistä käyttäjämäärästä riippumatta aina alle viidessä sekunnissa. Muut teknologiat sijoittuivat näiden arvojen väliin ollen kuitenkin selkeästi Adobe Flexiä nopeampia, selviytyen kaikista käyttäjämääristä alle 45 sekunnissa. Vaikka Java2Script oli vain hieman hitaampi kuin Echo Framework ei voida väittää, että se olisi nopeampi mitä GWT tai JSF teknologioilla tehdyt toteutukset sillä siinä ei suoritettu tietokanta hakua, joka muilla teknologioilla suoritettiin. Suorituskykytestin kesto kasvoi melko lineaarisesti käyttäjämäärän kasvaessa, pois lukien GWT- ja Flex-toteutukset, jotka hidastuivat käyttäjämäärien ollessa suuria. Palvelimen vastauskyky tukee testin kestosta saatuja tietoja sillä se oli suoraan verrannollinen testin suoritusaikaan. Echo Framework pystyi parhaimmillaan vastaamaan noin 120 sivupyyntöön sekunnissa käyttäjämäärän ollessa 500, joka osoittaa että se olisi pystynyt suoriutumaan suuremmistakin käyttäjämääristä ongelmitta. Palvelimen vastauskyvyssä oli kuitenkin suuria heittelyitä käyttäjämäärien suhteen eikä palvelimen vastauskyky kasvanut tai heikentynyt lineaarisesti, vastaus kyvyn ollessa heikoin viidelläkymmenellä käyttäjällä. Muilla teknologioilla paras vastauskyky saavutettiin jo paljon pienemmillä käyttä-

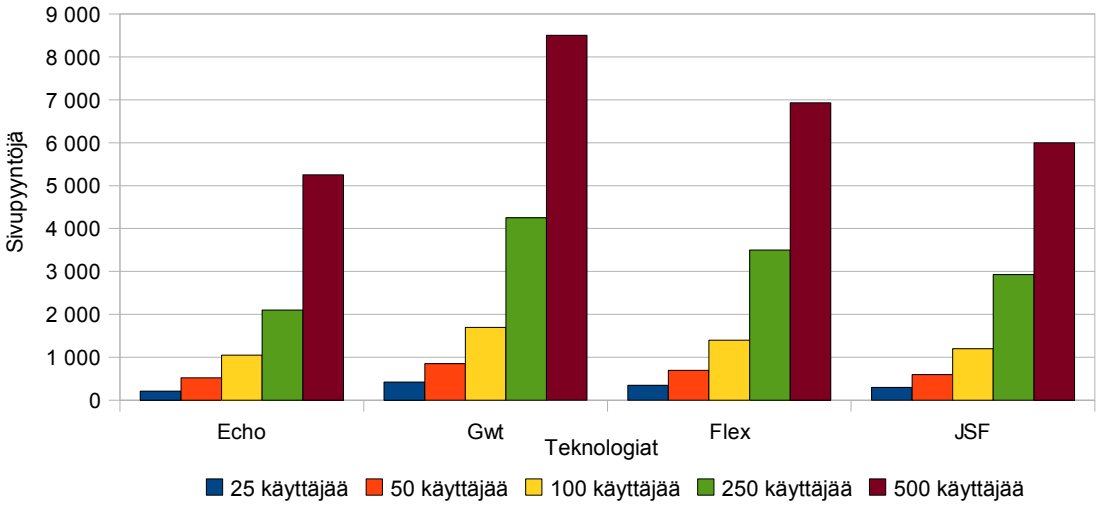
jämäärillä, jonka jälkeen palvelimen vastauskyky alkoi laskea. Toisin kuin Echo Frameworkin kohdalla muiden teknologioiden vastauskyvyissä ei havaittu yhtä suurta heittelehtimistä vaikkakin sitä oli havaittavissa. Java2Script oli ainoa joka pystyi vastaamaan melko yhteneväisesti alle 500 käyttäjän määriin. Adobe Flex ei pystynyt vastaamaan parhaimmillaankaan kuin noin kymmeneen palvelinpyyntöön sekunnissa, joka myös näkyi selvästi suoritusajassa. Palvelinpyyntöjen eli näytteiden määrä taas oli yllättäen suurin Echo Frameworkilla ja pienin JSF:llä toteutetussa käyttöliittymässä. GWT:llä, joka suoriutui toiseksi nopeiten suorituskykytesteistä, tehtiin myös toiseksi eniten palvelinpyyntöjä ja Adobe Flexillä kolmanneksi eniten vaikka se olikin hitain.



KUVIO 22. Toisen suorituskykytestin keston keskiarvot



KUVIO 23. Palvelimen keskimääräinen vastauskyky toisessa suorituskykytestissä



KUVIO 24. Näytteiden määrä toisessa suorituskykytestissä

Toisessa suorituskykytestissä tulokset olivat saman kaltaisia kuin ensimmäisessä ja myös tässä testissä nopeimmaksi osoittautui Echo Framework, joka myös tällä kertaa selviytyi alle viidessä sekunnissa suorituskykytestistä. Tällä kertaa Echo Framework

suoritti vähiten palvelinpyyntöjä, GWT:n suorittaessa eniten yli 8000 kappaletta. Suuri sivupyyntöjen määrä näkyi myös GWT:llä ajettujen suorituskykytestien suoritusajassa, joka oli hieman pidempi 20-250 käyttäjällä muihin RIA-teknologioihin verrattaessa. Kun käyttäjiä oli 500 suoritusaika pomppasi GWT:llä melkein viiteenkymmeneen sekuntiin, joka oli yli kolme kertaa pidempään kuin muilla. Palvelimen vastauskyky myös hieman tasoittui teknologioiden kesken, mutta siitä huolimatta Echo Framework pystyi vastaamaan useiten tehtyihin palvelinpyyntöihin. Toisin kuin ensimmäisessä suorituskykytestissä, tällä kertaa Echo Frameworkin paras vastauskyky oli sadalla käyttäjällä, jonka jälkeen vastauskyky laski 60-80 vastaukseen sekunnissa. Muilla teknologioilla palvelin pystyi vastaamaan noin 20-50 kertaa sekunnissa.

5.2.3 Yhteenvetona

Suorituskykytesteissä ilmeni, että Echo Frameworkilla toteutettu käyttöliittymä oli selkeästi muita valittuja teknologioita nopeampi ja se pystyi vastaamaan suurempaan määrään sivupyyntöjä huomattavasti nopeammin. Normaalissa sivuston käytössä, jossa sivustoa käytetään enemmän kuin yhden sivulatauksen verran Echo Frameworkin tekemät sivupyyntöjen määrä laski, kun muilla teknologioilla sivupyyntöjen määrä nousi. Myös Adobe Flex paransi suorituskykyä selkeästi testien muuttuessa vastaamaan normaali käyttöä vaikka se säilytti paikkansa kolmantena palvelinpyyntöjen määrässä. JSF ja GWT olivat molemmissa testeissä tasavahvoja eikä testitulokset juurikaan muuttuneet, paitsi GWT:llä toisen testin suoritusaika piteni räjähdysmäisesti kun käyttäjiä oli 500. Tämä kuitenkin voi johtua myös ympäristöstä, esimerkiksi hetkellisestä verkon tukkeutumisesta, eikä pelkästään teknologiasta.

6 POHDINTA JA JOHTOPÄÄTÖKSET

6.1 Tutkimuskysymysten analysointi

Tutkimuskysymykset olivat onnistuneesti valittuja ja tukivat tutkimuksen toteutusta alusta loppuun saakka. Niiden avulla pystyttiin tutkimaan web-käyttöliittymien kehittymistä ja RIA-teknologioita suhteellisen laajasti ja usealta eri kantilta. Kysymykset auttoivat myös rajaamaan aiheen sopivan kokoiseksi, sillä aihetta olisi pystynyt laajentamaan helposti liian laajaksi ja hajanaiseksi. Esimerkiksi olisi voitu tukia millaisissa yhteyksissä RIA-teknologioita yleensä käytetään ja kuinka laajasti ne ovat vallanneet alaa Internetissä. Kysymysten järjestys oli myös hyvin onnistunut ja sen avulla tutkimuksen rakenteesta saatiin tehtyä johdon mukainen ja selkeä kokonaisuus. Ensimmäisen kysymyksen avulla saatiin mahdollistettua hyvät lähtökohdat RIA-teknologioiden tarkempaan tutkimiseen ja aina edelleen omien RIA-toteutuksien tekemiseen asti. Toisen kysymyksen kautta saatiin RIA-teknologioista lisää teoriatietoa, joka edesauttoi lopputuloksen saavuttamisessa. Kolmanteen ja viimeisen kysymykseen etsiessä vastausta, saatiin hieman käytännön kokemusta erilaisten RIA-teknologioiden käytöstä niin integraatiosta valmiiseen projektiin kuin Mavenin käytöstä niiden kanssa.

6.2 Tutkimuksen tulokset ja niiden hyödyntäminen

Java2Script toteutuksen teon yhteydessä ongelmat, joihin törmättiin olivat hienonen pettymys, koska sillä oli helppo toteuttaa käyttöliittymiä ja se vaikutti mielenkiintoiselta, vaikkakin myös muut teknologiat olivat myös mielenkiintoisia. GWT-toteutuksen yhteydessä Gileadia koskeviin ongelmiin törmätessä saatiin tuotettua hyödyllistä tietoa Gileadin pääkehittäjälle, vaikkei ongelmaa itseään saatu ratkaistua. Konfiguraatioiden kannalta oli harmillista ettei Gileadia voitu käyttää vaan jouduttiin turvautumaan huomattavasti vanhempaan versioon Gileadistä. Jos GWT:n kanssa olisi voitu käyttää uusinta Gileadia olisivat niin konfiguraatiot kuin GWT-palvelinpyytöjä varten

toteutettu Service-tason luokka olleet hieman erilaisia. Myös projektin uudelleenkäytettävyys olisi parantunut, sillä Gilead olisi voitu tällöin korvata pelkällä konfiguraatio muutoksella toisin kuin nyt.

Adobe Flexin kautta tutustuttiin useampaankin eri integraatio teknologiaan, jotta yhteys käyttöliittymän ja palvelimen välille saatiin muodostettua. Spring Integraatio-projektin nuoruuden takia konfiguraation toteutuksessa oli pieniä ongelmia dokumentaation puutteen vuoksi. Dokumentaation puutteista huolimatta saatiin projektin konfiguraatiot toimimaan ja ne olivat monipuoliset ja jakautuivat useampiin tiedostoihin. Tutkimuksesta kuitenkin saatiin jokaisen teknologian kohdalta hyödyllisiä esimerkkikonfiguraatioita eri teknologioiden käyttöönottoa varten valmiiseen projektiin, jossa käytetään Mavenia, Spring Frameworkia tai Hibernatea. Suoritettujen suorituskykytestien tuloksia voidaan käyttää teknologia valintaa tehdessä hyvänä lisätieto lähteenä, josta selviää suorituskyvyn lisäksi myös suuntaa antavaa tietoa teknologioiden suorittamien palvelinpyyntöjen määristä. Saatuja tuloksia voidaan hyödyntää esimerkiksi RIA-teknologiaa valittaessa sekä varsinkin valittua teknologiaa käyttöön otettaessa, jolloin alustavat konfiguraatiot voidaan kopioida melkein suoraan liitteistä. Myös RIA-teknologiohin alustavasti tutustuvat henkilöt voivat löytää hyödylliseksi RIA-teknologioiden historiaa sekä RIA ja perinteisten -teknologioiden eroja käsittelevät osa-alueet.

Codecenter Oy:n käyttöön suositeltavan teknologian valinta oli hankalaa sillä jokaisella teknologialla oli omat hyvät ja huonot puolensa. Varsinkin GWT:n ja Echo Frameworkin välillä valinta oli erittäin hankala. Saatujen suorituskykytulosten ja tehtyjen toteutuksien perusteella voidaan kuitenkin suositella Webtrix-julkaisujärjestelmässä käytettäväksi Echo Frameworkia. Valintaan vaikutti käyttöliittymän toteuttamisen helppous ja käyttöönoton yksinkertaisuus, joka onnistui määrittelemällä web.xml:n ohjelman käynnistävä servlet. Myös saadut suorituskykytulokset vahvistivat valintaa osoittaen ettei Echo Framework hyydy isommallakaan kuormituksella. Ainoa miinus teknologiassa oli CSS-tuen puuttuminen, jonka takia tyylittäminen jouduttiin tekemään Echo Frameworkin omaa xml-syntaksia käyttäen. Haittaa voidaan kuitenkin pitää hyvin pienenä

LÄHTEET

A brief history of the internet 2009. Viitattu 11.03.2009 http://www.walthowe.com/navnet/history.html

Adobe Macromedia: Rich client 2004. viitattu: 25.02.2009 http://www.adobe.com/resources/business/rich_internet_apps/whitepapers.html, Macromedia Flash MX: A Next-Generation Rich Client

Adobe Systems: Flex Framework 2008. Viitattu: 04.03.2009 http://www.adobe.com/products/flex/features/flex_framework/

Ajaxline – JavaScript Frameworks 2009. Viitattu 15.04.2009 http://www.ajaxline.com/10-most-popular-javascript-frameworks

Apache Jmeter n.d. Viitattu 25.09.2009 http://jakarta.apache.org/jmeter

Apache Jmeter: aggregate report n.d. Viitattu: 29.09.2009 http://jakarta.apache.org/jmeter/usermanual/component_reference.html#Aggregate_Report

Bibeault, Crane, Sonneveld 2007. Ajax in Practice, Manning Publications Co. 4 - 6

Echo Framework 2007. Viitattu 04.05.2009. http://echo.nextapp.com/

Echo Sovelluskehyksen toimintaperiaate 2007. Viitattu 04.05.2009. http://echo.nextapp.com/site/node/32

Gulli & Signorini 2005. Viitattu 08.04.2009 http://www.cs.uiowa.edu/~asignori/websize/

Hanson & Tacy 2007. GWT in Action, Manning Publications Co.

Hibernate4Gwt 2008. Viitattu 29.06.2009 http://hibernate4gwt.sourceforge.net/, Overview

Java2Script – yleiskuva 2008. Viitattu 04.03.2009 http://j2s.sourceforge.net/, Overview

Routia 2007. Tuote ja tieto. Tuotteiden tutkimus ja kehittäminen. Viitattu 16.02.2009 http://www2.uiah.fi/projects/metodi/f00.htm, Vertailu

Saarsoo 2006. Viitattu 07.04.2009 http://triin.net/2006/06/12/Coding_practices_of_web_pages, css

Sun MicroSystems – Unified EL 2009. Viitattu: 04.06.2009
http://java.sun.com/products/jsp/reference/techart/unifiedEL.html

Sun Microsystems: JSF yleiskuva 2009. Viitattu 03.03.2009 http://java.sun.com/javaee/javaserverfaces/overview.html

Teoreettis-käsitteellinen tutkimus n.d. Kajaanin Ammattikorkeakoulu Viitattu 16.02.2009. http://193.167.122.14/Opari/ontTukiTutkTeoreettis.aspx

W3C DOM 2009. Viitattu 15.04.2009 http://www.w3.org/DOM/

LIITTEET

Liite 1. Spring Framework konfiguraatio

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                            http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                            http://www.springframework.org/schema/tx
                            http://www.springframework.org/schema/tx/spring-tx-2.0.xsd"
        default-lazy-init="false">

<!-- PostgreSQL tietokanta määrittelyt -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
   <property name="driverClassName" value="org.postgresql.Driver" />
   <property name="url" value="jdbc:postgresql://localhost/imagegallery?protocolVersion=2" />
   <property name="username" value="imagegallery" />
   <property name="password" value="" />
</bean>
        <bean id="sessionFactory" parent="parentSessionFactory">
   <property name="hibernateProperties">
      <value>
         hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
         #hibernate.dialect=org.hibernate.dialect.HSQLDialect
      </value>
   </property>
   <!-- päivitetäänkö Model luokkien muutoksen automaattisesti kantaan -->
   <property name="schemaUpdate" value="true"/>
</bean>

<!-- Transaktio managerin määrittely -->
<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
   <property name="sessionFactory" ref="sessionFactory"/>
</bean>
        <bean id="parentSessionFactory"
             class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
   <property name="dataSource" ref="dataSource"/>
   <!-- Määritellään annotoiduit model-luokat, jotka kuvaavat tietokantatauluja -->
   <property name="annotatedClasses">
      <list>
         <value>imagegallery.model.Forum</value>
         <value>imagegallery.model.Message</value>
         <value>imagegallery.model.Configuration</value>
      </list>
   </property>
</bean>

<!-- määritellään transaktion sisällä suoritettavat  metodit/luokat -->
<tx:advice id="txAdvice" transaction-manager="txManager">
   <tx:attributes>
      <tx:method name="get*" read-only="true" />
      <tx:method name="*" propagation="REQUIRED"/>
   </tx:attributes>
```

```
</tx:advice>

        <!-- Services and Daos →
<bean id="forumService" class="imagegallery.impl.service.ForumServiceImpl">
   <property name="forumDao" ref="forumDao" />
</bean>
        <bean id="forumDao" class="imagegallery.impl.service.ForumDaoHibernate">
   <property name="sessionFactory" ref="sessionFactory" />
</bean>
        <bean id="imagegalleryService" class="imagegallery.impl.service.ImagegalleryServiceImpl">
    <property name="imagegalleryDao" ref="imagegalleryDao" />
</bean>
        <bean id="imagegalleryDao" class="imagegallery.impl.service.ImagegalleryDaoHibernate">
   <property name="forumDao" ref="forumDao" />
   <property name="sessionFactory" ref="sessionFactory" />
</bean>

</beans>
```

Liite 2. Myfaces – faces-config.xml

```
<?xml version="1.0"?>
<faces-config version="1.2"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xi="http://www.w3.org/2001/XInclude"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                  http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
       <application>
            <locale-config>
                 <default-locale>en</default-locale>
                 <supported-locale>en</supported-locale>
                 <supported-locale>fi</supported-locale>
            </locale-config>
            <variable-resolver>
                 org.springframework.web.jsf.DelegatingVariableResolver
            </variable-resolver>
            <default-render-kit-id>
                 org.apache.myfaces.trinidad.core
            </default-render-kit-id>
       </application>
      <managed-bean>
            <managed-bean-name>forumBean</managed-bean-name>
            <managed-bean-class>imagegallery.beans.ForumBean</managed-bean-class>
             <managed-bean-scope>request</managed-bean-scope>
            <managed-property>
                 <property-name>forumService</property-name>
                 <value>#{forumService}</value>
            </managed-property>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>galleryBean</managed-bean-name>
            <managed-bean-class>imagegallery.beans.GalleryBean</managed-bean-class>
            <managed-bean-scope>request</managed-bean-scope>
            <managed-property>
                 <property-name>imagegalleryService</property-name>
                 <value>#{imagegalleryService}</value>
```

```
            </managed-property>
            <managed-property>
                <property-name>forumService</property-name>
                <value>#{forumService}</value>
            </managed-property>
        </managed-bean>

        <!-- navigation rules -->
        <navigation-rule>
            <from-view-id>/galleries.jspx</from-view-id>
            <navigation-case>
                <from-outcome>messageSuccess</from-outcome>
                <to-view-id>/page2.jspx</to-view-id>
            </navigation-case>
            <navigation-case>
                <from-outcome>uploadSuccess</from-outcome>
                <to-view-id>/page2.jspx</to-view-id>
            </navigation-case>
            <navigation-case>
                <from-outcome>uploadFailed</from-outcome>
                <to-view-id>/page2.jspx</to-view-id>
            </navigation-case>
            <navigation-case>
                <from-outcome>displayGallery</from-outcome>
                <to-view-id>/galleries.jspx</to-view-id>
            </navigation-case>
            <navigation-case>
                <from-outcome>displayGallery</from-outcome>
                <to-view-id>/galleries.jspx</to-view-id>
            </navigation-case>
            <navigation-case>
                <from-outcome>showImage</from-outcome>
                <to-view-id>/image.jspx</to-view-id>
            </navigation-case>
        </navigation-rule>
        <navigation-rule>
            <from-view-id>/image.jspx</from-view-id>
            <navigation-case>
                <from-outcome>back</from-outcome>
                <to-view-id>/galleries.jspx</to-view-id>
            </navigation-case>
        </navigation-rule>
</faces-config>
```

Liite 3. Myfaces – web.xml

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                http://java.sun.com/xml/ns/j2ee/webapp_2_4.xsd"
            version="2.4">
     <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                classpath:/applicationContext.xml
            </param-value>
```

```
</context-param>

<context-param>
	<param-name>javax.faces.STATE_SAVING_METHOD</param-name>
	<param-value>client</param-value>
</context-param>

<context-param>
	<param-name>
		org.apache.myfaces.trinidad.USE_APPLICATION_VIEW_CACHE
	</param-name>
	<param-value>false</param-value>
</context-param>

<context-param>
	<param-name>
		org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION
	</param-name>
	<param-value>true</param-value>
</context-param>

<context-param>
	<param-name>org.apache.myfaces.trinidad.CHANGE_PERSISTENCE</param-name>
	<param-value>request</param-value>
</context-param>

<filter>
	 <filter-name>sessionFilter</filter-name>
	 <filter-class>
		org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
	</filter-class>
</filter>

<filter-mapping>
	<filter-name>sessionFilter</filter-name>
	 <url-pattern>/faces/*</url-pattern>
</filter-mapping>

<filter>
	<filter-name>trinidad</filter-name>
	<filter-class>org.apache.myfaces.trinidad.webapp.TrinidadFilter</filter-class>
</filter>

<filter-mapping>
	<filter-name>trinidad</filter-name>
	<servlet-name>faces</servlet-name>
</filter-mapping>

<!-- Servlets -->
<servlet>
	<servlet-name>faces</servlet-name>
	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>

<servlet>
	<servlet-name>imageServlet</servlet-name>
	<servlet-class>imagegallery.servlet.ImageServlet</servlet-class>
</servlet>

<servlet>
```

```
        <servlet-name>resources</servlet-name>
        <servlet-class>org.apache.myfaces.trinidad.webapp.ResourceServlet</servlet-class>
    </servlet>

    <!-- Servlet Mappings -->
    <servlet-mapping>
        <servlet-name>faces</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>imageServlet</servlet-name>
        <url-pattern>/images/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>resources</servlet-name>
        <url-pattern>/adf/*</url-pattern>
    </servlet-mapping>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!-- Welcome Files -->
    <welcome-file-list>
        <welcome-file>index.jspx</welcome-file>
    </welcome-file-list>
</web-app>
```

Liite 4. GWT-moduulin konfiguraatio

```
<!DOCTYPE module PUBLIC "//gwt-module/" "http://google-web-
toolkit.googlecode.com/svn/tags/1.6.2/distro-source/core/src/gwt-module.dtd">
<module>
    <!-- Inherit the core Web Toolkit stuff. -->
    <inherits name='com.google.gwt.user.User' />
    <inherits name='com.google.gwt.user.theme.standard.Standard' />
    <inherits name="net.sf.hibernate4gwt.Hibernate4Gwt15" />
    <!-- Specify the application entry point class.  -->
    <entry-point class='imagegallery.client.GalleriesView' />
    <!-- Inherit the Core module -->
    <source path="client" />
    <source path="model"/>
    <source path="exception" />
    <!-- Proxy generator -->
    <generate-with class="net.sf.hibernate4gwt.rebind.Gwt15ProxyGenerator">
        <when-type-assignable class="java.io.Serializable" />
    </generate-with>
    <!-- Specify the application specific style sheet. -->
    <stylesheet src='Application.css' />
</module>
```

Liite 5. GWT-projektin web.xml

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>
```

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd"
                version="2.4">
        <display-name>Imagegallery - GWT</display-name>
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>
        classpath:/applicationContext.xml
        classpath:/gwtApplicationContext.xml
      </param-value>
        </context-param>
        <servlet>
                <servlet-name>imageServlet</servlet-name>
                <servlet-class>imagegallery.servlet.ImageServlet</servlet-class>
        </servlet>
        <servlet>
                <servlet-name>uploadServlet</servlet-name>
                <servlet-class>imagegallery.servlet.UploadServlet</servlet-class>
        </servlet>
        <servlet>
                <servlet-name>ForumService</servlet-name>
                <servlet-class>imagegallery.server.ForumServiceImpl</servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>imageServlet</servlet-name>
                <url-pattern>/images/*</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>uploadServlet</servlet-name>
                <url-pattern>/upload.do</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>ForumService</servlet-name>
                <url-pattern>/imagegallery.Imagegallery/forum.rpc</url-pattern>
        </servlet-mapping>

        <filter>
          <filter-name>sessionFilter</filter-name>
          <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-
class>
        </filter>

        <filter-mapping>
          <filter-name>sessionFilter</filter-name>
          <url-pattern>/*</url-pattern>
        </filter-mapping>

        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>

        <welcome-file-list>
                <welcome-file>index.html</welcome-file>
        </welcome-file-list>
</web-app>
```

Liite 6. GWT-projektin pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLS-
chema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">

	<modelVersion>4.0.0</modelVersion>
	<parent>
		<artifactId>imagegallery</artifactId>
		<groupId>imagegallery</groupId>
		<version>1.0-SNAPSHOT</version>
	</parent>
	<groupId>${pom.parent.groupId}</groupId>
	<artifactId>imagegallery-gwt</artifactId>
	<packaging>war</packaging>
	<version>${pom.parent.version}</version>
	<name>gwt-maven-archetype-project</name>

	<properties>
		<gwt.version>1.6.4</gwt.version>
		<maven.compiler.source>1.5</maven.compiler.source>
		<maven.compiler.target>1.5</maven.compiler.target>
		<gwt.output.directory>${basedir}/target/gwt</gwt.output.directory>
		<gwt.module.alias>Imagegallery</gwt.module.alias>
	</properties>

	<dependencies>
		<dependency>
			<groupId>${pom.groupId}</groupId>
			<artifactId>core</artifactId>
			<version>${pom.version}</version>
		</dependency>
		<!-- Core module's sources for gwt compiler -->
		<dependency>
			<groupId>${pom.parent.groupId}</groupId>
			<artifactId>core</artifactId>
			<version>${pom.parent.version}</version>
			<classifier>sources</classifier>
		</dependency>
		<dependency>
			<groupId>commons-fileupload</groupId>
			<artifactId>commons-fileupload</artifactId>
			<version>1.2.1</version>
		</dependency>
		<dependency>
			<groupId>org.apache.commons</groupId>
			<artifactId>commons-io</artifactId>
			<version>1.3.2</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>2.5.5</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
```

```
        <artifactId>spring-web</artifactId>
        <version>2.5.5</version>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>2.5.5</version>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>2.5.5</version>
</dependency>
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>2.5.5</version>
</dependency>
<dependency>
        <groupId>aspectj</groupId>
        <artifactId>aspectjlib</artifactId>
        <version>1.5.3</version>
</dependency>
<dependency>
        <groupId>aspectj</groupId>
        <artifactId>aspectjrt</artifactId>
        <version>1.5.3</version>
</dependency>
<dependency>
        <groupId>aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.5.3</version>
</dependency>
<!--  GWT dependencies (from central repo) -->
<dependency>
        <groupId>com.google.gwt</groupId>
        <artifactId>gwt-servlet</artifactId>
        <version>${gwt.version}</version>
        <scope>runtime</scope>
</dependency>
<dependency>
        <groupId>com.google.gwt</groupId>
        <artifactId>gwt-dev</artifactId>
        <version>${gwt.version}</version>
        <classifier>linux</classifier>
        <scope>runtime</scope>
</dependency>
<dependency>
        <groupId>com.google.gwt</groupId>
        <artifactId>gwt-user</artifactId>
        <version>${gwt.version}</version>
        <scope>provided</scope>
</dependency>

<!-- Hibernate4Gwt and it's dependencies -->
<dependency>
        <groupId>com.sf.hibernate4gwt</groupId>
        <artifactId>hibernate4gwt</artifactId>
        <version>1.1.1</version>
</dependency>
```

```
        <dependency>
            <groupId>net.sf.beanlib</groupId>
            <artifactId>beanlib</artifactId>
            <version>3.3.0beta20</version>
        </dependency>
        <dependency>
            <groupId>net.sf.beanlib</groupId>
            <artifactId>beanlib-hibernate</artifactId>
            <version>3.3.0beta20</version>
        </dependency>
        <dependency>
            <groupId>javassist</groupId>
            <artifactId>javassist</artifactId>
            <version>3.6.0.GA</version>
        </dependency>
        <dependency>
            <groupId>com.thoughtworks.xstream</groupId>
            <artifactId>xstream</artifactId>
            <version>1.3</version>
        </dependency>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
            <version>2.4</version>
        </dependency>


        <!-- test -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.4</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.15</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.5.8</version>
        </dependency>
    </dependencies>

    <build>
        <outputDirectory>${gwt.output.directory}/WEB-INF/classes</outputDirectory>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>gwt-maven-plugin</artifactId>
                <version>1.1-SNAPSHOT</version>
                <executions>
                    <execution>
                        <phase>process-classes</phase>
                        <configuration>
                            <output>${gwt.output.directory}</output>
                            <logLevel>INFO</logLevel>
                        </configuration>
```

```
                <goals>
                    <goal>compile</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.0.2</version>
        <configuration>
            <source>${maven.compiler.source}</source>
            <target>${maven.compiler.target}</target>
            <encoding>UTF-8</encoding>
            <excludes>
                <exclude>javax.servlet/**</exclude>
            </excludes>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-resources-plugin</artifactId>
        <version>2.3</version>
        <configuration>
            <encoding>UTF-8</encoding>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-antrun-plugin</artifactId>
        <version>1.2</version>
        <executions>
            <execution>
                <id>war-folder-creation</id>
                <phase>process-resources</phase>
                <goals>
                    <goal>run</goal>
                </goals>
                <configuration>
                    <tasks>
                        <copy todir="${gwt.output.directory}">
                            <fileset dir="${basedir}/src/main/webapp" />
                        </copy>
                    </tasks>
                </configuration>
            </execution>
        </executions>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
            <webResources>
                <resource>
                    <directory>${gwt.output.directory}</directory>
                    <targetPath>.</targetPath>
                </resource>
            </webResources>
        </configuration>
    </plugin>
```

```
            </plugins>
        </build>
</project>
```

Liite 7. Echo-projektin pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <artifactId>imagegallery</artifactId>
                <groupId>imagegallery</groupId>
                <version>1.0-SNAPSHOT</version>
        </parent>
        <groupId>imagegallery</groupId>
        <artifactId>echo3</artifactId>
        <version>1.0-SNAPSHOT</version>
        <packaging>war</packaging>
        <name>Imagegallery - Echo WebApp</name>
        <build>
                <plugins>
                        <plugin>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <configuration>
                                        <source>1.5</source>
                                        <target>1.5</target>
                                </configuration>
                        </plugin>
                        <plugin>
                                <artifactId>maven-war-plugin</artifactId>
                        </plugin>
                        <plugin>
                                <groupId>org.mortbay.jetty</groupId>
                                <artifactId>maven-jetty-plugin</artifactId>
                                <configuration>
                                        <scanIntervalSeconds>1</scanIntervalSeconds>
                                        <connectors>
                                                <connector implementation="org.mortbay.jetty.nio.SelectChannelCon-
nector">
                                                        <port>8080</port>
                                                        <maxIdleTime>60000</maxIdleTime>
                                                </connector>
                                        </connectors>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
        <dependencies>
                <dependency>
                        <groupId>${pom.parent.groupId}</groupId>
                        <artifactId>core</artifactId>
                        <version>${pom.parent.version}</version>
                        <exclusions>
                                <exclusion>
                                        <groupId>xerces</groupId>
                                        <artifactId>xercesImpl</artifactId>
                                </exclusion>
```

```
        </exclusions>
    </dependency>
    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.2.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-io</artifactId>
        <version>1.3.2</version>
    </dependency>
    <dependency>
        <groupId>nextapp.echo</groupId>
        <artifactId>app</artifactId>
        <version>3.0.b7</version>
    </dependency>
    <dependency>
        <groupId>nextapp.echo</groupId>
        <artifactId>webcontainer</artifactId>
        <version>3.0.b7</version>
    </dependency>
    <dependency>
        <groupId>nextapp.echo</groupId>
        <artifactId>echopoint</artifactId>
        <version>3.0.0a7</version>
    </dependency>
    <dependency>
        <groupId>nextapp.echo</groupId>
        <artifactId>echo-file-transfer-app</artifactId>
        <version>3.0</version>
    </dependency>
    <dependency>
        <groupId>nextapp.echo</groupId>
        <artifactId>echo-file-transfer-webcontainer
        </artifactId>
        <version>3.0</version>
    </dependency>
    <dependency>
        <groupId>com.thoughtworks.xstream</groupId>
        <artifactId>xstream</artifactId>
        <version>1.3</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.1</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.14</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.4</version>
        <scope>provided</scope>
    </dependency>
```

```
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
                <version>2.5.5</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-web</artifactId>
                <version>2.5.5</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-beans</artifactId>
                <version>2.5.5</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-aop</artifactId>
                <version>2.5.5</version>
            </dependency>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-tx</artifactId>
                <version>2.5.5</version>
            </dependency>
            <dependency>
                <groupId>aspectj</groupId>
                <artifactId>aspectjlib</artifactId>
                <version>1.5.3</version>
            </dependency>
            <dependency>
                <groupId>aspectj</groupId>
                <artifactId>aspectjrt</artifactId>
                <version>1.5.3</version>
            </dependency>
            <dependency>
                <groupId>aspectj</groupId>
                <artifactId>aspectjweaver</artifactId>
                <version>1.5.3</version>
            </dependency>
        </dependencies>
        <reporting>
            <plugins>
                <plugin>
                    <artifactId>maven-jxr-plugin</artifactId>
                </plugin>
                <plugin>
                    <artifactId>maven-javadoc-plugin</artifactId>
                </plugin>
                <plugin>
                    <artifactId>maven-surefire-report-plugin
                    </artifactId>
                </plugin>
                <plugin>
                    <artifactId>maven-project-info-reports-plugin
                    </artifactId>
                </plugin>
            </plugins>
        </reporting>
</project>
```

Liite 8. Echo-projektin web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
			http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
	version="2.4">
	<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>
	classpath:/applicationContext.xml
	</param-value>
	</context-param>
	<servlet>
		<servlet-name>EchoServlet</servlet-name>
		<servlet-class>imagegallery.servlet.GalleryServlet</servlet-class>
		<load-on-startup>5</load-on-startup>
	</servlet>
	<servlet>
		<servlet-name>imageServlet</servlet-name>
		<servlet-class>imagegallery.servlet.ImageServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>EchoServlet</servlet-name>
		<url-pattern>/app</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>imageServlet</servlet-name>
		<url-pattern>/images/*</url-pattern>
	</servlet-mapping>

	<listener>
		<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
	</listener>

	<session-config>
		<session-timeout>10</session-timeout>
	</session-config>
</web-app>
```

Liite 9. Flex swc-moduulin pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLS-
chema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>imagegallery</groupId>
		<artifactId>imagegallery-flex</artifactId>
		<version>1.0-SNAPSHOT</version>
	</parent>
	<groupId>imagegallery</groupId>
```

```
    <artifactId>swc</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>swc</packaging>
    <name>swc Library</name>
    <build>
        <sourceDirectory>src/main/flex</sourceDirectory>
        <testSourceDirectory>src/test/flex</testSourceDirectory>
        <plugins>
            <plugin>
                <groupId>org.sonatype.flexmojos</groupId>
                <artifactId>flexmojos-maven-plugin</artifactId>
                <version>3.1.0</version>
                <extensions>true</extensions>
                <configuration>
                    <locales>
                        <locale>en_US</locale>
                    </locales>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>com.adobe.flex.framework</groupId>
            <artifactId>flex-framework</artifactId>
            <version>3.2.0.3958</version>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>net.sf.gilead</groupId>
            <artifactId>adapter-actionscript</artifactId>
            <version>1.2.3</version>
            <type>jar</type>
        </dependency>
    </dependencies>
</project>
```

Liite 10. Flex swf-moduulin pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLS-
chema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>imagegallery</groupId>
        <artifactId>imagegallery-flex</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <groupId>imagegallery</groupId>
    <artifactId>swf</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
	<packaging>swf</packaging>
	<name>swf Application</name>

	<build>
		<sourceDirectory>src/main/flex</sourceDirectory>
		<testSourceDirectory>src/test/flex</testSourceDirectory>
		<plugins>
			<plugin>
				<goals>
					<goal>compile-swf</goal>
				</goals>
				<groupId>org.sonatype.flexmojos</groupId>
				<artifactId>flexmojos-maven-plugin</artifactId>
				<version>3.2.0</version>
				<extensions>true</extensions>
				<configuration>
					<output>
						${basedir}/../war/src/main/webapp/${project.build.finalName}/$
{project.build.finalName}.swf
					</output>
					<services>
						${basedir}/../war/src/main/webapp/WEB-INF/flex/services-config.xml
					</services>
					<contextRoot>/</contextRoot>
					<locales>
						<locale>en_US</locale>
					</locales>
				</configuration>
			</plugin>
		</plugins>
	</build>

	<dependencies>
		<dependency>
			<groupId>com.adobe.flex.framework</groupId>
			<artifactId>flex-framework</artifactId>
			<version>3.2.0.3958</version>
			<type>pom</type>
		</dependency>
		<dependency>
			<groupId>imagegallery</groupId>
			<artifactId>swc</artifactId>
			<version>1.0-SNAPSHOT</version>
			<type>swc</type>
		</dependency>
	</dependencies>
</project>
```

Liite 11. Flex war-moduulin pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLS-
chema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
```

```
        <groupId>imagegallery</groupId>
        <artifactId>imagegallery-flex</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <groupId>imagegallery</groupId>
    <artifactId>war</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <build>
        <plugins>
            <plugin>
                <groupId>org.sonatype.flexmojos</groupId>
                <artifactId>flexmojos-maven-plugin</artifactId>
                <version>3.1.0</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>copy-flex-resources</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
    <repositories>
        <repository>
            <id>gilead</id>
            <url>https://gilead.svn.sourceforge.net/svnroot/gilead/gilead/maven-repo/</url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>imagegallery</groupId>
            <artifactId>swf</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>swf</type>
        </dependency>
        <dependency>
            <groupId>imagegallery</groupId>
            <artifactId>core</artifactId>
            <version>${pom.parent.artifactId}</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.4</version>
        </dependency>
        <dependency>
            <groupId>commons-fileupload</groupId>
            <artifactId>commons-fileupload</artifactId>
            <version>1.2.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-io</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
```

```
        <groupId>xalan</groupId>
        <artifactId>xalan</artifactId>
        <version>2.7.1</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.14</version>
    </dependency>
    <dependency>
        <groupId>aspectj</groupId>
        <artifactId>aspectjrt</artifactId>
        <version>1.5.3</version>
    </dependency>
    <dependency>
        <groupId>aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.5.3</version>
    </dependency>
    <dependency>
        <groupId>imagegallery</groupId>
        <artifactId>core</artifactId>
        <version>1.0-SNAPSHOT</version>
        <exclusions>
            <exclusion>
                <groupId>xerces</groupId>
                <artifactId>xercesImpl</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.0.71</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.1.2</version>
    </dependency>
    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-core-tiger</artifactId>
        <version>2.0.4</version>
        <exclusions>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-support</artifactId>
            </exclusion>
        </exclusions>
```

```
</dependency>
<dependency>
    <groupId>org.springframework.integration</groupId>
    <artifactId>spring-integration-core</artifactId>
    <version>1.0.2.SR1</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.flex</groupId>
    <artifactId>spring-flex</artifactId>
    <version>1.0.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.6.3</version>
</dependency>
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-core</artifactId>
    <version>5.2.0</version>
    <exclusions>
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging-api</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.apache.camel</groupId>
            <artifactId>camel-core</artifactId>
        </exclusion>
```

```
            </exclusions>
        </dependency>

        <!--  Gilead dependencies -->
        <dependency>
            <groupId>net.sf.gilead</groupId>
            <artifactId>adapter-core</artifactId>
            <version>1.2.3</version>
            <exclusions>
                <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring</artifactId>
        </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>net.sf.gilead</groupId>
            <artifactId>hibernate-util</artifactId>
            <version>1.2.3</version>
        </dependency>
        <dependency>
            <groupId>net.sf.gilead</groupId>
            <artifactId>adapter4blazeds</artifactId>
            <version>1.2.3</version>
        </dependency>

        <dependency>
            <groupId>net.sf.beanlib</groupId>
            <artifactId>beanlib-hibernate</artifactId>
            <version>5.0.2beta</version>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>net.sf.beanlib</groupId>
            <artifactId>beanlib</artifactId>
            <version>5.0.2beta</version>
            <exclusions>
                <exclusion>
                    <groupId>xstream</groupId>
                    <artifactId>xstream</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-hibernate</artifactId>
            <version>1.2.6</version>
        </dependency>
    </dependencies>
</project>
```

Liite 12. Flex-projektin pää pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLS-
chema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<artifactId>imagegallery</artifactId>
		<groupId>imagegallery</groupId>
		<version>1.0-SNAPSHOT</version>
	</parent>
	<groupId>imagegallery</groupId>
	<artifactId>imagegallery-flex</artifactId>
	<version>1.0-SNAPSHOT</version>
	<packaging>pom</packaging>

	<modules>
		<module>swc</module>
		<module>swf</module>
		<module>war</module>
	</modules>
</project>
```

Liite 13. Flex-projektin web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd"
	version="2.4">

	<display-name>Spring BlazeDS Integration Samples</display-name>

	<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>
			classpath:/applicationContext.xml
		</param-value>
	</context-param>

	<listener>
		<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
	</listener>

	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet>
		<servlet-name>imageServlet</servlet-name>
		<servlet-class>imagegallery.servlet.ImageServlet</servlet-class>
	</servlet>
	<servlet>
		<servlet-name>uploadServlet</servlet-name>
```

```
        <servlet-class>imagegallery.servlet.UploadServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/messagebroker/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>imageServlet</servlet-name>
        <url-pattern>/images/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>uploadServlet</servlet-name>
        <url-pattern>/upload.do</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>sessionFilter</filter-name>
        <filter-class>
            org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
        </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>sessionFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

Liite 14. Flex-projektin dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:flex="http://www.springframework.org/schema/flex"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/flex
        http://www.springframework.org/schema/flex/spring-flex-1.0.xsd">


    <bean id="persistenceUtil" class="net.sf.gilead.core.hibernate.spring.HibernateSpringUtil"
        factory-method="getInstance">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <bean id="hibernate-object"
        class="org.springframework.flex.core.ManageableComponentFactoryBean"
        depends-on="sessionFactory">
        <constructor-arg value="net.sf.gilead.blazeds.adapter.PersistentAdapter" />
        <property name="properties">
            <value>
                {"persistence-factory":
                    {
```

```
                                "class":"net.sf.gilead.core.hibernate.spring.HibernateSpringUtil",
                                "singleton" :"true",
                                "method":"getSessionFactory"
                            },
                            "stateless":"false"
                        }
                    </value>
                </property>
        </bean>

        <flex:message-broker>
                <flex:remoting-service default-adapter-id="hibernate-object" />
        </flex:message-broker>

        <!-- Expose the contactService bean for BlazeDS remoting -->
        <flex:remoting-destination ref="forumService" />
</beans>
```

Liite 15. Flex-projektin services-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
        <services>
                <default-channels>
                        <channel ref="my-amf" />
                </default-channels>
        </services>
        <channels>
                <channel-definition id="my-amf"
                        class="mx.messaging.channels.AMFChannel">
                        <endpoint
                                url="http://{server.name}:{server.port}/{context.root}/messagebroker/amf"
                                class="flex.messaging.endpoints.AMFEndpoint" />
                </channel-definition>
        </channels>

        <logging>
                <target class="flex.messaging.log.ConsoleTarget" level="Warn">
                        <properties>
                                <prefix>[BlazeDS] </prefix>
                                <includeDate>true</includeDate>
                                <includeTime>true</includeTime>
                                <includeLevel>true</includeLevel>
                                <includeCategory>true</includeCategory>
                        </properties>
                        <filters>
                                <pattern>Endpoint.*</pattern>
                                <pattern>Service.*</pattern>
                                <pattern>Configuration</pattern>
                        </filters>
                </target>
        </logging>
        <system>
                <redeploy>
                        <enabled>false</enabled>
                </redeploy>
        </system>
</services-config>
```