Joni Korkalainen

WWW-sovelluskehitys Vaadin-kehyksellä

Tietotekniikan
kandidaatintutkielma
14. kesäkuuta 2012

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Joni Korkalainen
Yhteystiedot: joni.korkalainen@student.jyu.fi
Työn nimi: WWW-sovelluskehitys Vaadin-kehyksellä
Title in English: WWW-application development with Vaadin-framework
Työ: Tietotekniikan kandidaatintutkielma
Sivumäärä: 76
Tiivistelmä: Tämä tutkielma käsittelee suomalaista WWW-sovelluskehystä Vaadinta. Se on palvelinpohjainen Java-kehys rikkaiden WWW-sovellusten kehittämiseen. Tutkielmassa esitellään myös WWW-kehyksiä yleisesti sekä WWW-kehysten yleensä käyttämä MVC-arkkitehtuuri. Lisäksi esitellään myös Vaatimen käyttämät tekniikat: AJAX, UIDL, JSON ja DOM sekä kuinka Vaadin niitä käyttää. Vaatimesta käydään läpi käyttöliittymäkomponentit, tapahtumien käsittely, tietomalli ja ulkoasun muokkaus. Lopuksi esitellään Vaatimen käyttöä esimerkkisovelluksen avulla.
English abstract: This thesis talks about a finnish WWW-framework called Vaadin. It is a server-based Java-framework that helps you build rich WWW-applications. In this thesis we also talk about WWW-frameworks in general and explain the MVC-architecture, which is used by most of the WWW-frameworks. We also explain the technologies used by Vaadin: AJAX, UIDL, JSON and DOM and also explain how Vaadin uses them. Then we introduce Vaadin: its UI-components, event handling, data model and layout managing. Finally we introduce the use of Vaadin with an application example.
Avainsanat: Rikas WWW-sovellus, Java, WWW-sovelluskehys, AJAX, Google Web Toolkit
Keywords: Rich internet-application, Java, WWW-framework, AJAX, Google Web Toolkit

Sisältö

1 Johdanto

Käyttäjillä on nykyään kovat vaatimukset WWW-projekteille. He vaativat rikkaita ja käyttäjäystävällisiä WWW-sovelluksia, jotka ovat turvallisia, esteettömiä ja niin edelleen. WWW-sovellusten kehittäminen onkin nykyään entistä monimutkaisempaa. WWW-ohjelmistoteollisuus on kehittynyt siten, että nykyään avoimen lähdekoodin yhteisöt ja ohjelmistoyritykset tarjoavat ohjelmistosuunnittelijoille kehyksiä, joiden avulla WWW-sovellusten kehittämistä saadaan yksinkertaistettua. Monet kehykset perustuvat samanlaiselle idealle, mutta ratkaisevat ongelmat hieman eri tavoin ja keskittyen kerrosarkkitehtuurin eri osa-alueisiin (malli-, näkymä- ja ohjainosat) [8, s. 7].

WWW-sovelluskehykset yksinkertaistavat siis niin sanottujen rikkaiden WWW-sovellusten kehittämistä. Rikkaat WWW-sovellukset (engl. *Rich Internet Applications, RIA*) ovat monipuolisia, dynaamisia WWW-sivuja. Rikkaiden WWW-sovellusten käsite on määrittänyt WWW-sovellukset uudelle tasolle kannattamalla rikkaita ja monipuolisia käyttöliittymiä sekä parantamalla käyttäjän vuorovaikutusta ja sovellusten käytettävyyttä [18]. Rikkaat WWW-sovellukset tarjoavat monimutkaisia käyttöliittymiä monipuolisten prosessien ja datan esittämiseen, kuitenkin minimoimalla asiakas-palvelin datasiirrot [1]. Ne muistuttavat hyvin paljon normaaleja työpöytäsovelluksia. WWW-sovellusten toteuttaminen on rikkaiden WWW-sovellusten myötä tullut yhä hankalammaksi. Nykyään on tarjolla runsaasti WWW-sovelluskehyksiä helpottamaan rikkaiden WWW-sovellusten kehittämistä.

Vaadin on suomalainen WWW-sovelluskehys rikkaiden WWW-sovellusten kehittämiseen. Vaadin tunnettiin aikaisemmin nimellä IT Mill Toolkit, jonka juuret ovat Millstone web UI -kirjastossa. Millstonen kehitys alkoi vuonna 2000, josta alkaen kehys on kehittynyt. Kehys on pidetty taaksepäin yhteensopivana, joten sovellukset, jotka on luotu vuonna 2003, voidaan helposti päivittää käyttämään nykyistä versiota. IT Mill Toolkitista tehtiin avoimen lähdekoodin sovellus vuonna 2007. Vuonna 2009 kirjasto sai nykyisen nimensä Vaadin [17].

Tässä tutkielmassa esitellään ensin WWW-sovelluskehykset yleisesti ja esitellään myös WWW-sovellusten yleensä käyttämä kolmikerrosarkkitehtuuri. Tämän jälkeen siirrytään käsittelemään tarkemmin itse Vaadinta. Ensin esitellään Vaatimen käyttämät tekniikat ja kuinka Vaadin niitä käyttää. Sitten esitellään käyttöliittymäkomponentit Vaatimessa: Mitä komponentteja Vaadin tarjoaa, minkälainen on nii-

den rakenne ja kuinka ne esitetään asiakkaalle. Yksittäisiä komponentteja ei erikseen esitellä, mutta lomake-komponenttia (Form) katsotaan hieman tarkemmin. Sen yhteydessä tarkastellaan myös Vaatimen tarjoamia oikeellisuustarkistuksia lomakekentille. Seuraavaksi esitellään tapahtumien käsittelyä, eli kuinka käyttäjän toimet käyttöliittymässä välitetään palvelimelle ja kuinka niitä käsitellään. Sitten tutkitaan Vaatimen tarjoamaa tietomallia, minkä jälkeen esitellään, kuinka Vaatimella onnistuu sovelluksen ulkoasun muokkaus ja minkälaisia vaihtoehtoja siihen on tarjolla. Lopuksi esitellään vielä esimerkkisovelluksena Vesa Lappalaisen Ohjelmointi 2 -kurssilla esimerkkinä käyttämän Valuuttamuuntimen toteutus Vaatimella WWW-sovellukseksi. Lisäksi liitteenä on saman kurssin esimerkkiharjoitustyö Kerho toteutettuna Vaatimella WWW-sovellukseksi.

2 WWW-kehykset yleisesti

2.1 Mitä ovat WWW-kehykset?

WWW-kehykset ovat kokoelma kirjastoja, komponentteja ja työkaluja, jotka auttavat ohjelmistokehittäjiä kehittämään ja ylläpitämään monimutkaisia WWW-sovelluksia nopeasti ja tehokkaasti [9]. Kehykset mahdollistavat koodin uudelleenkäytettävyyden tarjoamalla valmiita yleiskäyttöisiä osia, joita muokkaamalla kehittäjä voi luoda omia sovelluskohtaisia osia [3, s. 11]. Kehykset tarjoavat valmiit uudelleenkäytettävät koodit yleisimmille funktioille ja luokille [9].

Kehyksien avulla saadaan erotettua ohjelmatoiminnot ja logiikka ulkoasusta (HTML, CSS tai muut muotoilutiedostot). Tämä auttaa ulkoasun suunnittelijoita (vaikka ilman minkäänlaista ohjelmointikokemusta) suunnittelemaan ja muokkaamaan ulkoasua ilman ohjelmoijan apua [9].

WWW-kehyksiä luodaan helpottamaan koodin yksityiskohtien kirjoittamisen taakkaa. Kehyksien tulisi toimia yleisellä tasolla, jotta ne soveltuisivat erilaisille sovelluksille. Kehykset mahdollistavat sovelluskehittäjien keskittymisen sovelluksen vaatimusten täyttämiseen, sen sijaan että käyttäisivät aikaa esimerkiksi tavallisten komponenttien luomiseen [7, s. 57].

WWW-kehyksiä on tarjolla runsaasti ja myös niiden toteutukset vaihtelevat. On sekä palvelin- että selainpohjaisia kehyksiä. Selainpohjaiset kehykset keskittyvät ainoastaan siihen, mitä WWW-selaimessa tapahtuu ja jättävät palvelinpuolen toteutuksen ohjelmistokehittäjän tehtäväksi. Tällöin ohjelmistokehittäjä toteuttaa AJAX-yhteyden palvelimelle itse valitsemallaan ohjelmointikielellä. Nämä kehykset ovat lähes aina toteutettu Javascriptillä. Palvelinpohjaiset kehykset lisäävät AJAX-yhteydet itse WWW-sovellukseen [15, s. 117]. Näissä kehyksissä suurin osa logiikasta suoritetaan palvelimessa. Palvelinpohjaisia kehyksiä on toteutettu eri ohjelmointikielille kuten Java, Python, C++ ja niin edelleen. Varsinkin Java-kehyksiä on tarjolla runsaasti, myös Vaadin on sellainen.

2.2 WWW-kehysten tuomat hyödyt

Kehykset yksinkertaistavat sovelluksen kehittämistä jakamalla ongelman erillisiin osiin, kuten käyttöliittymään, tietokantayhteyteen ja varsinaiseen sovelluslogiikkaan.

Järjestelmien suunnitteleminen modulaarisesti on tärkeää, ja kehykset, jotka ovat luonteeltaan geneerisiä sovelluskehitystyökaluja kannustavat tähän lähestymistapaan. Sovellus, joka on suunniteltu kehyksen avulla, on todennäköisemmin suunniteltu käyttäen modulaarista suunnittelua kuin ilman kehystä tehty sovellus. Modulaarisuuden avulla saavutettava muunneltavuus auttaa uutta sovellusta olemaan kestävämpi jatkuvasti kehittyville vaatimuksille. Moduuleja voidaan vaihtaa tai päivittää, kun määritykset muuttuvat, eikä koko sovelluksen rakennetta tarvitse rakentaa uudestaan [13, s. 55].

Kehykset tarjoavat sovelluksen kehittämiseen perustan, josta sovellusta voi lähteä rakentamaan. Ne varmistavat, että sovellus sisältää kaikki tärkeimmät elementit kuten turvallisuuden, tietokantayhteyden, käyttöliittymän ja niin edelleen. Kehykset tarjoavat eräänlaisen luonnoksen, johon tietyn sovelluksen yksityiskohtia voi alkaa täyttämään. Tällainen perusta mahdollistaa helpon rinnakkaisohjelmoinnin. Useat kehittäjät voivat työskennellä eri osien parissa tietäen, että palaset sopivat myöhemmin yhteen [13, s. 56].

Kehysten ansiosta sovellukseen on mahdollista saada projektin aika- tai hintarajoissa enemmän toimintoja. Esimerkiksi sovellus voisi hyötyä kehittyneemmästä virheiden hallinnasta, mutta sen tekemiseen ei ole aikaa. Tämä toiminto voi kuitenkin olla jo valmiina jossakin kehyksessä, jolloin sitä käyttämällä saataisiin aikaan parempi ja valmiimpi ohjelma [13, s. 57–58].

2.3 WWW-kehysten ominaisuuksia

Seuraavaksi esitellään joitakin yleisimpiä kehysten ominaisuuksia. Joitakin ominaisuuksia on useammissa kehyksissä kuin toisia, ja eri kehyksillä on myös eri vahvuusalueita [13, s. 57–89] .

- **Komponenttien tarjoaminen:** Kehykset tarjoavat komponentit, joiden avulla sovellukset rakennetaan. Joillakin kehyksillä on omat komponenttinsa, jotkut taas tarjoavat pääsyn esimerkiksi sovelluspalvelimen komponentteihin.

- **Sovelluslogiikka:** WWW-arkkitehtuuri perustuu pyyntö/vastaus -malliin. Asiakas täyttää lomakkeen ja hyväksyy sen, palvelin vastaa. Useimmat kehykset yrittävät toimia tämän mallin mukaan ja tarjoavat siihen erilaisia keinoja. Sovelluslogiikka-yksikön pitäisi myös tehdä sovelluksen logiikan luominen mahdollisimman helpoksi ja mahdollisuuksien mukaan standardeja noudattaen.

- **Tietokantayhteys:** Monet nykypäivän sovellukset ovat yhteydessä jonkinlaiseen tietokantaan. Monet kehykset tarjoavatkin helpomman pääsyn tietokantoihin ja jotkut mahdollistavat myös tietokannan ylläpidon.

- **Kirjaukset:** Kirjaukset (engl. *logging*) voivat lisätä minkä tahansa koodin luettavuutta. Kirjauksien tekemiseen on rakennettu monia työkaluja ja monet kehykset sisältävät näitä työkaluja. Näin on helppo tehdä koodia, joka mahdollistaa sovelluksen kirjaavan kehityksensä tavalla tai toisella. Usein kirjaukset ovat myös sisäänrakennettuina itse kehyksen komponentteihin.

- **Tapahtumankäsittely:** Useimmat kehykset tarjoavat yhden tai useamman keinon tapahtumien käsittelyyn (engl. *Event Handling*). Tapahtumankäsittelyä käsitellään tarkemmin luvussa 7.

- **Virheidenhallinta:** Java tarjoaa *exceptions*-mekanismin virheiden käsittelyyn. Jotkut kehykset tarjoavat tätä kehittyneemmän mekanismin virheiden havaitsemiseen ja hallintaan. Tavoitteita, joita virheidenhallinnalla yritetään saavuttaa, ovat esimerkiksi: Virheen jäljittäminen korjausta varten ja virheen korjaus sekä virheestä ilmoittaminen käyttäjälle tai järjestelmän ylläpitäjälle.

- **Turvallisuus:** Yksi tärkeimmistä asioista WWW-sovelluksien kannalta on oikeanlaisen turvallisuustason määrittäminen ja toteuttaminen. Parhaat kehykset tarjoavat keinoja, joilla turvallisuustason saa valittua.

- **Ulkoasu ja käyttöliittymä:** Kehykset voivat tarjota keinoja sovelluksen visuaalisen ulkoasun luomiseen ja on myös kehyksiä, jotka keskittyvät juuri sovelluksen tähän osaan. Sovelluksen ulkoasun voi myös luoda monella eri tavalla ja parhaat kehykset antavatkin kehittäjän itsensä päättää haluamansa tavan.

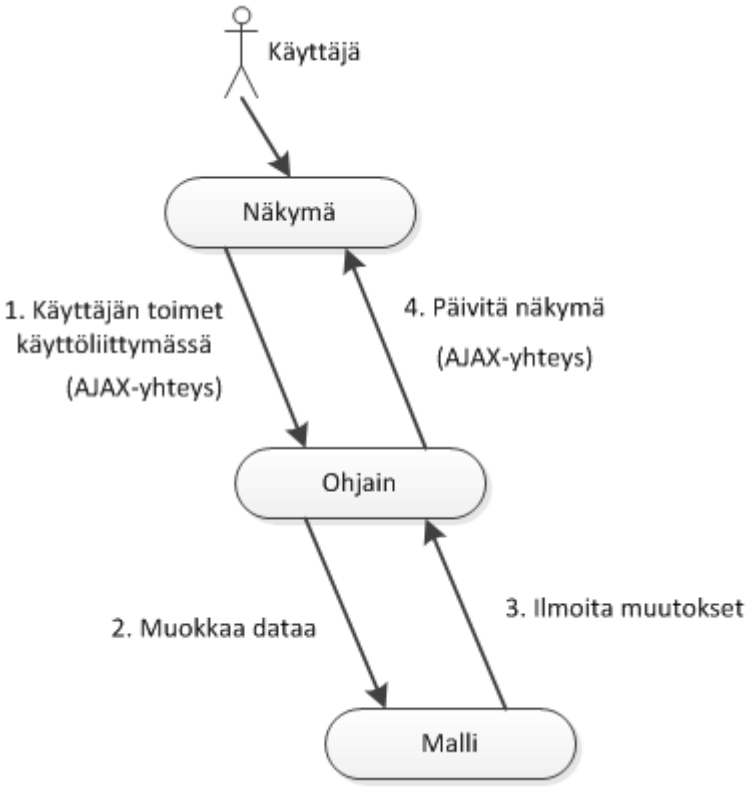
Kehyksillä voi olla myös muitakin ominaisuuksia kuin yllä kuvatut, ja eri kehykset voivat keskittyä eri alueisiin. Ominaisuuksien lisäksi monet kehykset tarjoavat myös työkaluja avuksi sovellusten kehittämiseen, esimerkiksi testaukseen, koodin automaattiseen luomiseen ja tietokannan luomiseen [13, s. 102–103].

3 Malli-Näkymä-Ohjain-arkkitehtuuri

Malli-Näkymä-Ohjain-arkkitehtuuri (engl. *Model-View-Controller, MVC*) erottaa ulkoasun, kontrollilogiikan ja datan käsittelyn toisistaan kolmeen erilliseen komponenttiin. Käyttäjä on vuorovaikutuksessa käyttöliittymän kanssa WWW-selaimen kautta. Ohjain on vastuussa käyttäjän toimien luomista tapahtumista. Kun ohjain vastaanottaa tällaisen tapahtuman, se kutsuu datan käsittelijää ja päivittää ulkoasua käyttäjän tekemien toimien mukaiseksi. Ulkoasukomponentti luo uuden käyttöliittymän datan mukaiseksi, mutta datan käsittelijä ei koskaan ole yhteydessä suoraan ulkoasukomponenttiin. Tämän jälkeen käyttöliittymä alkaa odottaa uutta käyttäjän syötettä ja edellä kuvattu malli alkaa alusta [7, s. 30].

Malli on siis esitys sovelluksen ongelma-alueesta, siitä asiasta, jonka kanssa sovellus tekee töitä. Esimerkiksi tekstinkäsittelyohjelman mallina on dokumentti. Näkymä on sovelluksen se osa, joka esittää asiat käyttäjälle (lomakkeet, kuvat, tekstin ja niin edelleen). MVC:n näkymän ja mallin ei tulisi koskaan olla yhteydessä suoraan toisiinsa. Kun käyttäjä esimerkiksi painaa nappia, ilmoittaa näkymä siitä ohjaimelle. Ohjain sitten muokkaa mallia ja päättää vaatiiko muutokset mallissa näkymän päivittämistä. Jos vaatii, niin ohjain kertoo näkymälle, kuinka sen pitää itseään muuttaa [2, s. 91]. Kuvassa 3.1 esitetään MVC-arkkitehtuurin toiminta.

Palvelinpuolen koodauskielille on nykyään kehyksiä, jotka perustuvat MVC-suunnittelumallille [7, s. 30]. Vaadin on tällainen Java-kehys.



Kuva 3.1: MVC-arkkitehtuuri ja AJAX [2, s. 92], [4]

4 Vaadin lyhyesti

Vaadin on palvelinpohjainen kehys rikkaiden WWW-sovellusten kehittämiseen. Javascript- ja selainpohjaisten ratkaisujen sijaan Vaadinta käytettäessä suurin osa logiikasta suoritetaan palvelimessa. Vaatimen avulla WWW-käyttöliittymiä voi kehittää aivan kuten normaaleja Java-työpöytäsovelluksia. Vaadin hoitaa käyttöliittymän hallitsemisen selaimessa sekä AJAX-yhteydet palvelimen ja selaimen välillä [5, s. 1].

Vaadin käyttää Google Web Toolkitiä (GWT) käyttöliittymän esittämiseen selaimessa. GWT kääntää Javalla tehdyn koodin Javascriptiksi. Siksi ohjelmoijan ei tarvitse osata normaaleja selainteknologioita kuten HTML tai Javascript. Ainoa koodi, jota ohjelmoijan tarvitsee kirjoittaa on Java. Koska selaimen päässä suoritetaan ainoastaan Javascript-koodia, ei sovelluksen käyttäjän tarvitse asentaa selaimeen mitään liitännäisiä kuten Adoben Flash tai Microsoftin Silverlight [5, s. 2]. Javascript myös toimii automaattisesti kaikissa suurimmissa selaimissa.

5 Vaatimen asiakaspuolen moottori

5.1 Asiakaspuolen moottorin tehtävät

Vaatimen asiakaspuolen moottori (engl. *Client-Side Engine*) hoitaa sovelluksen esittämisen WWW-selaimessa GWT:n avulla. Se ilmoittaa käyttäjän toimet ja muutokset käyttöliittymässä palvelinpuolen *Terminal Adapterille* käyttäen UIDL:ää (User Interface Definition Language). UIDL esitellään tarkemmin luvussa 5.3. Ilmoitukset tehdään asynkronisilla HTTP- tai HTTPS-pyynnöillä [5, s. 34–35].

Käyttöliittymää hallitaan *ApplicationConnection*-luokan avulla. Se hoitaa AJAX-pyynnöt palvelimelle ja esittää käyttöliittymän saatujen vastausten mukaisesti. Aluksi selaimeen ladataan tyhjä sivu, joka lataa Vaatimen asiakaspuolen moottorin Javascript-koodin. Kun se on ladattu ja käynnistetty, moottori hoitaa AJAX-pyynnöt palvelimelle. Kaikki yhteydenpito palvelimelle tehdään *ApplicationConnection*-luokan läpi. Yhteydenpito hoidetaan UIDL-viesteillä käyttäen JSON-viestienvaihtoformaattia (JavaScript Object Notation) HTTP(S)-yhteyden läpi [5, s. 38–39]. JSON esitellään tarkemmin luvussa 5.4.

5.2 AJAX

AJAX (Asynchronous JavaScript and XML) on suhteellisen uusi termi, jonka on keksinyt Adaptive Pathin Jesse James Garrett. AJAX kasaa yhteen vanhoja WWW-tekniikoita, jotka on asennettuna useimpiin nykyaikaisiin tietokoneisiin. AJAX tuo esiin näiden vanhojen tekniikoiden aiemmin toteutumattomia mahdollisuuksia. AJAX:in avulla voidaan toteuttaa todellisesti rikkaita WWW-sovelluksia. Esimerkiksi Google käyttää AJAX:ia ja nostaa käyttäjien odotuksia siitä, mitä WWW-sovellus voi tehdä [2, s. 4]. Esimerkkejä AJAX-sovelluksista ovat esimerkiksi Google Maps ja Gmail [4].

Normaalit WWW-sovellukset toimivat seuraavasti: Käyttäjän toimet käyttöliittymässä käynnistävät HTTP-pyynnön WWW-palvelimelle. Palvelin prosessoi tiedon ja palauttaa HTML-sivun asiakkaalle. Ongelmana tässä on, että aina kun palvelin tekee hommiaan, niin asiakas joutuu odottamaan [4].

AJAX-sovellus eliminoi tämän aloita-odota tyylisen vuorovaikutuksen niin sanotun AJAX-moottorin avulla. Tämä AJAX-moottori toimii palvelimen ja asiakkaan

välillä. AJAX-moottori on kirjoitettu Javascriptillä ja on usein piilotettu näkymättömään kehykseen. Sen sijaan, että jokaisen tapahtuman aluksi ladattaisiin uusi WWW-sivu, selain lataakin AJAX-moottorin. Moottorin tehtävinä ovat käyttöliittymän näyttäminen käyttäjälle ja kommunikointi palvelimelle käyttäjän puolesta. AJAX-moottori mahdollistaa vuorovaikutuksen käyttäjän ja sovelluksen välillä tapahtuvan asynkronisesti, erillään kommunikoinnista palvelimen kanssa. Tämän ansiosta käyttäjä ei ikinä joudu katselemaan tyhjää ikkunaa ja odottamaan, kun palvelin tekee hommiaan [4].

Jokainen käyttäjän toiminta, joka normaalisti luo HTTP-pyynnön, toteutetaan Javascript kutsuna AJAX-moottorille *XMLHttpRequest*-objektin avulla. Moottori käsittelee itse sellaiset toiminnot, jotka eivät vaadi palvelimen apua; esimerkiksi yksinkertainen datan validointi tai datan muokkaaminen muistissa. Jos moottori tarvitsee palvelimen apua, se tekee ne pyynnöt asynkronisesti viivästyttämättä käyttäjän vuorovaikutusta sovelluksen kanssa [4]. Palvelinpuolen ohjelma voi luoda vastauksen XML:nä, mutta usein käytetään myös vaihtoehtoisia formaatteja kuten tavallinen teksti, HTML-koodi tai JavaScript Object Notation (JSON) -formaatti, jota myös Vaadin käyttää. Vastaanotettu sisältö käsitellään tyypillisesti Javascriptillä, johon on liitetty Document Object Model (DOM) -metodeja [14, s. 3–5]. DOM esitellään tarkemmin luvussa 5.5.

5.3 UIDL

UIDL (User Interface Definition Language) koostuu erikseen määritellystä korkean tason ohjelmointikielestä, joka kuvaa käyttöliittymän ominaisuudet ottaen huomioon muut osat interaktiivisesta sovelluksesta. Tällainen kieli sisältää määritellyn syntaksin (eli kuinka ominaisuudet voidaan kuvata tämän kielen avulla) ja semantiikan (eli mitä ominaisuudet tarkoittavat oikeassa elämässä). UIDL:ää voidaan pitää yleisenä keinona määritellä käyttöliittymä riippumattomasti mistään kohdekielestä, jota käytetään käyttöliittymän toteutukseen [6, s. 36].

5.4 JSON

Yhteydet selaimen ja palvelimen välillä vaativat usein datan jäsentämistä. Vaatimen versio 4 käytti XML:ää datan välittämiseen, mutta versio 5 käyttää JSON:ia [5, s. 36].

JSON (JavaScript Object Notation) on kevyt datanvaihtoformaatti. Säännöllisen syntaksinsa ansiosta se on ohjelmallisesti helposti käsiteltävissä. JSON on periaat-

teessa Javascriptin osajoukko ja jopa XML:ää helpommin jäsenneltävissä. JSON:ia ja XML:ää käytetään periaatteessa samaan tarkoitukseen eli datan esittämiseen ja välittämiseen. Verrattuna XML:ään JSON on vähempisanainen (kuva 5.1), mikä johtaa suoraan parempaan suorituskykyyn. Vähemmän dataa tarkoittaa vähemmän bittejä siirrettäväksi ja jäsenneltäväksi. JSON vaatii XML:ää vähemmän kaistanleveyttä, muistia ja prosessoritehoja [10].

XML-formaatti	JSON-formaatti

<?xml version='1.0' encoding='UTF-8'?>
<henkilot>
<henkilo>
<sukunimi>Meikäläinen</sukunimi>
<etunimi>Matti</etunimi>
<kotikunta>Jyväskylä</kotikunta>
<ammatti>Ohjelmistosuunnittelija</ammatti>
</henkilo>
<henkilo>
<sukunimi>Mehiläinen</sukunimi>
<etunimi>Maija</etunimi>
<kotikunta>Helsinki</kotikunta>
<ammatti>Opiskelija</ammatti>
</henkilo>
</henkilot>
	{
	"henkilot" :
	[
	{
	"sukunimi" : "Meikäläinen",
	"etunimi" : "Matti",
	"kotikunta" : "Jyväskylä",
	"ammatti" : "Ohjelmistosuunnittelija"
	},
	{
	"sukunimi" : "Mehiläinen",
	"etunimi" : "Maija",
	"kotikunta" : "Helsinki",
	"ammatti" : "Opiskelija"
	}
	]
	}

Kuva 5.1: XML- ja JSON-formaattiesimerkit [10]

Vaikka AJAX alunperin käytti XML:ää datan välittämiseen, mikä tahansa formaatti käy; HTML, tavallinen teksti tai vaikkapa JSON. Kuitenkin koneen näkökulmasta XML, HTML ja JSON ovat kaikki tavallista tekstiä [10].

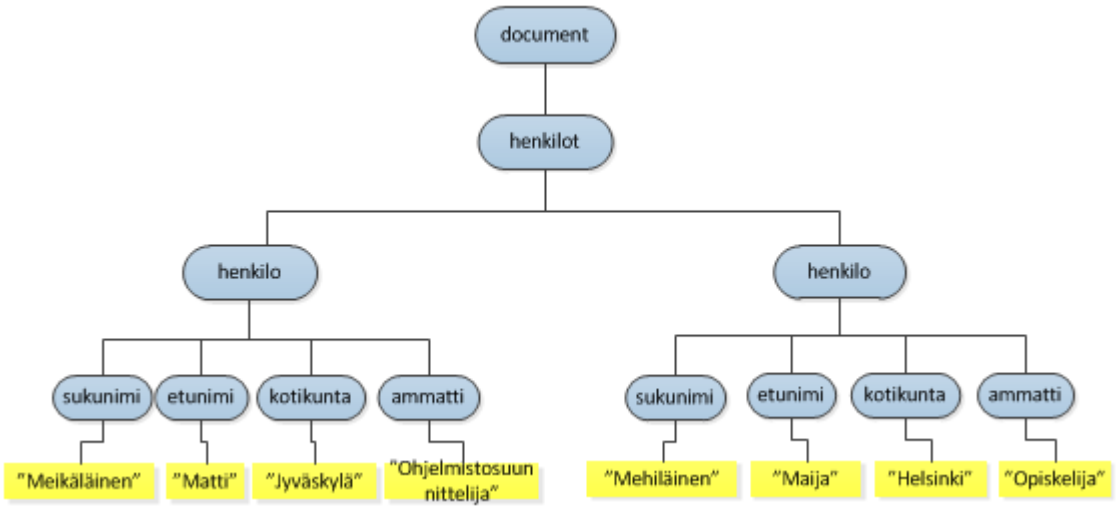
Vaatimen asiakaspuolen moottori käyttää JSON:ia Google Web Toolkitin läpi. GWT tukee JSON yhteyksiä *com.google.gwt.json.client*-paketissa [5, s. 37].

5.5 DOM

Document Object Model (DOM) on laitteistosta ja ohjelmointikielestä riippumaton rajapinta, joka mahdollistaa sovellusten ja skriptien pääsyn käsiksi ja päivittävän dokumentin sisältöä, rakennetta ja tyyliä. Dokumenttia voidaan edelleen käsitellä ja käsittelyn tulokset voidaan sisällyttää takaisin näytettävään sivuun [11, s. 13–14].

DOM on tapa esittää dokumentti (esimerkiksi XML-, HTML- tai JSON-dokumentti) puurakenteena (kuva 5.2). DOM:in metodien ja ominaisuuksien avulla voi

päästä käsiksi mihin tahansa sivun elementtiin, muokata tai poistaa elementtejä tai lisätä uusia elementtejä [16, s. 218].



Kuva 5.2: Kuvan 5.1 XML-esimerkki DOM-puurakenteena [12], [11, s. 15–16]

6 Käyttöliittymäkomponentit

6.1 Vaatimen tarjoamat komponentit

Käyttöliittymä koostuu käyttöliittymäkomponenteista (engl. *User Interface Components*) ja kuten luvussa 2.3 todettiin, yksi WWW-kehyksen perustehtävistä on komponenttien tarjoaminen. Vaadin tarjoaa valikoiman valmiita käyttöliittymäkomponentteja, mutta antaa ohjelmoijalle myös mahdollisuuden luoda omia komponentteja Google Web Toolkitin avulla. Omien komponenttien luominen onnistuu myös Eclipsen Vaadin-liitännäisen työkalulla. Lisäksi komponentteja on saatavilla myös liitännäisinä joko Vaatimen omista hakemistoista tai vapaista lähteistä. Sekä kaupallisia että ilmaisiakin komponentteja on löydettävissä [5, s. 64].

Vaatimen tarjoamissa valmiissa komponenteissa on oikeastaan kaikki yleisimmin tarvittavat komponentit ja vähän enemmänkin. Löytyy label, tekstikenttä (TextField), tekstialue (TextArea), nappi (Button), erilaiset valintakomponentit kuten valintalista (ListSelect) sekä paljon muita komponentteja. Vaadin tarjoaa myös lomakekomponentin (Form), jonka avulla lomakkeiden teko on yksinkertaisempaa. Lomakkeelle saa asetettua otsikon, kuvauksen ja alaviitteen ja se sisältää asettelun (engl. *layout*), jonka sisään kentät voi lisätä [5, s. 136]. Lomakkeen saa myös yhdistettyä suoraan dataan, kuten luvun 8.2 esimerkeissä näytetään. Lomake sisältää myös virheenilmaisimen ja lomakkeen kentät saa liitettyä Vaatimen tarjoamiin oikeellisuudentarkistimiin. Vaatimen tarjoamia tarkistimia ovat esimerkiksi *IntegerValidator* ja *DoubleValidator* numeerisen syötön tarkistamiseen, *EmailValidator* sekä *RegexpValidator*, jolla voi tarkistaa vastaako jono Javan säännöllistä lauseketta [5, s. 141].

Esimerkki Vaatimen oikeellisuustarkistuksesta [5, s. 141]:

```
// Postinumeron pitää olla 5 merkkiä (10000-99999).
TextField tfPostinro = new TextField("Postinumero");
tfPostinro.setColumns(5);

// Luodaan tarkistaja ja lisätään se yllä luotuun
// tekstikenttään.
Validator postinumeroValidator = new RegexpValidator(
  "[1-9][0-9]{4}", "Postinumeron pitää olla numero
```

```
  väliltä 10000-99999.");
tfPostinro.addValidator(postinumeroValidator);
```

Lomakkeen kentän voi myös asettaa pakolliseksi ja laittaa sille virheilmoituksen, jos se on jätetty tyhjäksi [5, s. 142–143]:

```
form.getField("nimi").setRequired(true);
form.getField("nimi").setRequiredError("Nimi puuttuu");
```

6.2 Komponenttien välittäminen asiakkaan ja palvelimen välillä

Jokaisella käyttöliittymän palvelinpuolen (engl. *server-side*) komponentilla on vastakappale asiakaspuolella (engl. *client-side*). Asiakaspuoli saa komponentit ja niissä tapahtuvat käyttäjän toimet *Terminal Adapterin* avulla. Käyttöliittymäkomponentit viestittävät muutoksista Terminal Adapterille, joka esittää ne käyttäjän selaimessa. Asiakaspuoli taas lähettää käyttäjän toimet käyttöliittymässä ensin Terminal Adapterille asynkronisina AJAX-pyyntöinä. Terminal Adapter välittää sitten nämä tapahtumat käyttöliittymäkomponenteille, jotka välittävät ne sovelluksen logiikalle [5, s. 34–35].

6.3 Komponenttien rakenne

Vaatimen käyttöliittymäkomponentit on rakennettu rajapintojen (engl. *interfaces*) ja käsiteluokkien (engl. *abstract classes*) päälle, jotka määrittelevät ja toteuttavat ominaisuudet kaikille kyseisten luokkien komponenteille. Ne määrittelevät myös kuinka komponenttien tilat välitetään palvelimen ja asiakkaan välillä. Esimerkiksi *Button-*, *TextField-*, *DateField-* ja *Form*-komponentit perivät *AbstractField*-luokan, joka perii *Field*-rajapinnan. Kaikki komponentit perivät myös *Paintable*-rajapinnan, jota käytetään komponenttien välittämiseen asiakaspuolelle, sekä vastakkaisen *VariableOwner*-rajapinnan, jota käytetään komponentin tilan tai käyttäjän toimien ilmoittamisessa palvelimelle. Vaatimen rajapintojen lisäksi kaikki komponentit perivät *java.io.Serializable*-rajapinnan salliakseen tietojen välittämisen palvelimen ja asiakkaan välillä [5, s. 64–66].

7 Tapahtumankäsittely

7.1 Tapahtumankäsittely yleisesti

Tapahtuma (engl. *Event*) voi kuvata prosessointia monella eri sovelluksen tasolla. Käyttöliittymään liittyvät tapahtumat syntyvät esimerkiksi, kun käyttäjä painaa nappia tai valitsee toiminnon. Tapahtumat voivat myös olla sovelluksen sisäisiä tai dataan liittyviä. Esimerkiksi tapahtuma voi syntyä, kun dataa tallennetaan tietokantaan. Tapahtuman merkitys riippuu oikeastaan täysin kysessä olevasta kehyksestä. Tapahtumalähtöistä logiikkaa käytetään useimmiten käyttöliittymään liittyvissä toiminnoissa, mutta tekniikka on yleistymässä myös sovelluksen muiden osien toteutuksessa [13, s. 67–68].

Yleinen malli tapahtumien toteutuksessa on, että on elementti, joka synnyttää tapahtuman sekä kuuntelija-elementti (engl. *listener*), joka kuuntelee tapahtumia. Usein toiset luokat tulevat tietoisiksi tietyistä tapahtumista rekisteröimällä itsensä tapahtuman kuuntelijoiksi. Ideaalisesti kehyksen pitäisi käyttää yleistä tapahtumanhallintamekanismia, jolloin tapahtumat kaikilla sovelluksen tasoilla voivat omata rekisteröidyn kuuntelijan, ja voivat saada tietyn toiminnon aikaan, kun tapahtuma ilmaantuu [13, s. 67–68].

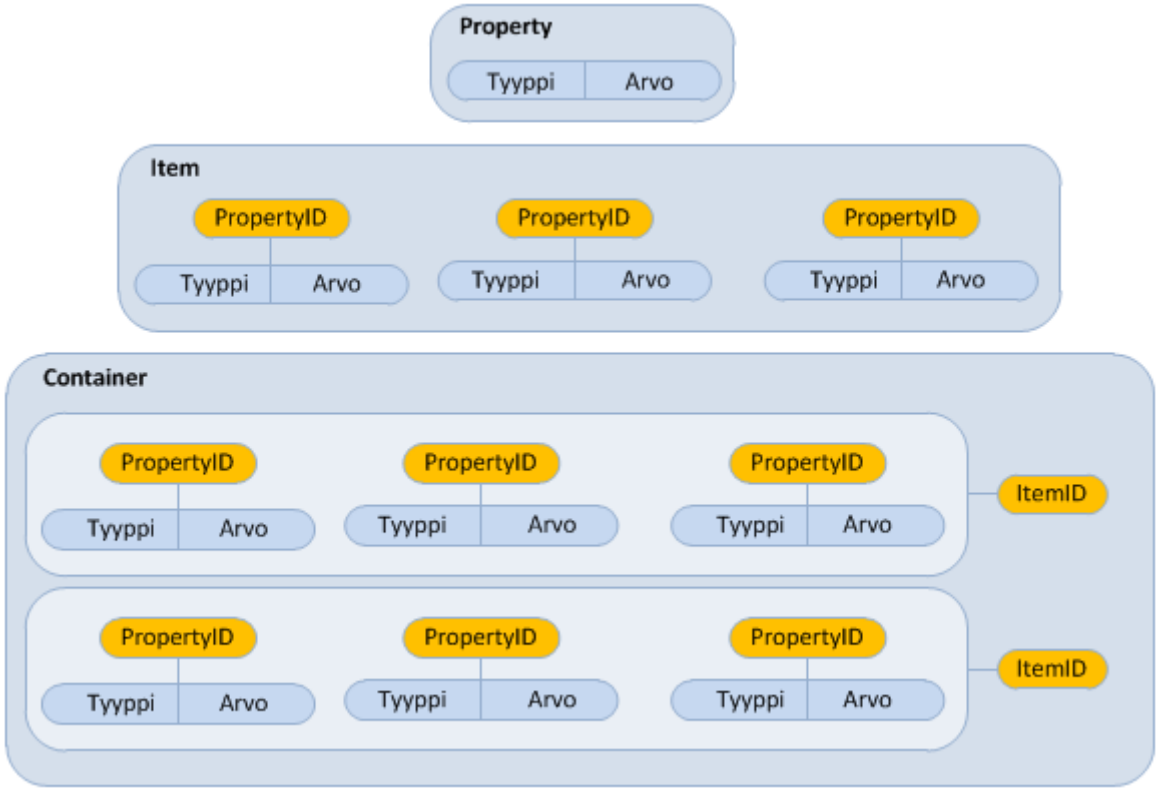
7.2 Tapahtumankäsittely Vaatimessa

Kun käyttäjä tekee jotain käyttöliittymässä, kuten painaa nappia, on sovelluksen tiedettävä siitä. Vaadin hoitaa nämä yhteydenpidot käyttäjän syötteen ja sovelluksen logiikan välillä käyttämällä tapahtuma-kuuntelija-mallia. Tämä malli toimii siten, että olio luo tapahtumia, joita kuuntelijat sitten kuuntelevat. Kun tapahtuma ilmaantuu, olio lähettää ilmoituksen siitä kaikille kuuntelijoille. Normaalisti kuuntelijoita on vain yksi. Vastaanottaakseen tapahtumia sovelluksen on rekisteröitävä kuuntelija-olio tapahtuma-lähteen kanssa. Kuuntelijat rekisteröidään komponenteissa *addListener()*-metodilla. Useimmat komponentit, joilla on samankaltaisia tapahtumia, määrittelevät oman tapahtuma-luokan ja vastaavat kuuntelija-luokat. Esimerkiksi napilla (Button) on *Button.ClickEvent*-tapahtumat, joita kuunnellaan *Button.ClickListener*-rajapinnan läpi [5, s. 39].

8 Vaatimen tietomalli

8.1 Tietomallin osat ja ominaisuudet

Vaatimen tietomallissa (engl. *Data Model*) on kolme sisäkkäistä tasoa: *Property*, *Item* ja *Container*. Esimerkiksi taulukon tapauksessa edelliset vastaisivat tasoja (samassa järjestyksessä) solu, rivi ja taulukko. Kuvassa 8.1 esitetään tietomallin rakenne. Propertyt kerätään Itemeihin ja Itemit voidaan sisällyttää Containeriin. Propertyillä ja Itemeilla on omat ID-tunnuksensa, joilla ne voidaan yksilöidä ja tunnistaa. Esimerkiksi, jos Containerina on taulukko, niin Itemit vastaavat rivejä ja Property vastaa yhtä solua. Tiettyyn riviin tai soluun pääsee siis käsiksi näiden ID-tunnuksella, joka voi olla esimerkiksi rivin nimi [5, s. 213–216].



Kuva 8.1: Vaatimen tietomalli [5, s. 214]

Vaatimen tietomalli ei määritä datan esitystapaa vaan ainoastaa rajapinnat. Tällöin esitystavan määrittäminen tapahtuu Containerien toteutuksen avulla. Esitysta-

pa voi olla melkein mitä vaan, kuten *Plain Old Java Object*-rakenne (POJO), tiedostojärjestelmä tai tietokantakysely [5, s. 214].

Propertyllä tarkoitetaan yhtä tiettyä data-objektia, jonka arvo voidaan lukea tai kirjoittaa. Arvo voidaan lukea *getValue()*-metodilla ja kirjoittaa *setValue()*-metodilla. Palautusarvo on objekti-viite eli se pitää sitten muuntaa oikeaksi tyypiksi ennen käyttöä. Muutokset Propertyn arvossa saadaan *ValueChangeEventin* avulla, jota voidaan kuunnella *ValueChangeListenerin* avulla [5, s. 215–216].

Item-rajapinta tarjoaa pääsyn tiettyyn kokoelmaan Propertyjä. Viittauksen tiettyyn Propertyyn saa *getItemProperty()*-metodilla käyttämällä Propertyn ID-tunnusta [5, s. 218].

Container on Vaatimen tietomallin rajapintojen korkein taso. Sen avulla voidaan hallita kokoelmaa Itemeita, jotka sisältävät samankaltaisia Propertyjä. Itemit voidaan lisätä *addItem()*-metodilla [5, s. 220].

8.2 Komponenttien yhdistäminen dataan

Vaatimen tietomalli mahdollistaa käyttöliittymäkomponenttien (vaatimessa MVC:n näkymä-osa) yhdistämisen suoraan dataan, jota ne käsittelevät [5, s. 213].

BeanItem on toteutus *Item*-rajapinnasta, ja on kääre *Java Bean*-objekteille. Ainoastaan *set*- ja *get*-metodit tarvitaan, jotta *papu* saadaan yhdistettyä komponenttiin [5, s. 219].

Seuraavaksi esimerkki, jossa yhdistetään papu lomakkeeseen. Tehdään ensin henkilöpapu, jossa on attribuuteille *set*- ja *get*-metodit [5, s. 219–220]:

```
public class HenkiloBean {
    String nimi = "";
    String hetu = "";
    String osoite = "";
    String puhelin = "";

    public String getNimi() {
        return nimi;
    }
    public void setNimi(String nimi) {
        this.nimi = nimi;
    }
    public String getOsoite() {
```

```
        return osoite;
    }
    public void setOsoite(String osoite) {
        this.osoite = osoite;
    }
    public String getHetu() {
        return hetu;
    }
    public void setHetu(String hetu) {
        this.hetu = hetu;
    }
    public String getPuhelin() {
        return puhelin;
    }
    public void setPuhelin(String puhelin) {
        this.puhelin = puhelin;
    }

}
```

Pääohjelmassa luodaan uusi henkilöpapu ja yhdistetään se lomakkeeseen [5, s. 220]:

```
Window main = new Window("The Main Window");
setMainWindow(main);

// Luodaan uusi papu
HenkiloBean bean = new HenkiloBean();
// Kääritään se BeanItemiin
BeanItem<HenkiloBean> item = new BeanItem<HenkiloBean>(bean);
// Yhdistetään se lomakkeeseen
Form form = new Form();
form.setItemDataSource(item);

form.setCaption("Henkilötiedot");
main.addComponent(form);
```

Saadaan kuvan 8.2 mukainen lomake.

Kuva 8.2: Lomake, johon on yhdistetty henkilöpapu

BeanContainer on astia *Java Bean*-objekteille. Jokainen sisällytetty papu on kääritty *BeanItem*-kääreen sisään. Itemin ominaisuudet otetaan selville automaattisesti setterien ja getterien avulla. Yhteen astiaan voi laittaa vain samantyyppisiä papuja [5, s. 221].

Seuraavaksi esimerkki, jossa papuja lisätään BeanContaineriin, joka yhdistetään taulukkoon. Käytetään edellisen esimerkin papua, johon pitää vain lisätä konstruktori, jolla saadaan arvot asetettua [5, s. 221]:

```
public HenkiloBean(String hetu, String nimi, String osoite,
                   String puhelin) {
    this.hetu = hetu;
    this.nimi = nimi;
    this.osoite = osoite;
    this.puhelin = puhelin;
}
```

Pääohjelmassa luodaan sitten BeanContainer, johon lisätään papuja ja lopuksi yhdistetään se taulukkoon [5, s. 221–222]:

```
Window main = new Window("The Main Window");
setMainWindow(main);

// Luodaan säiliö pavuille. Item Id:t laitetaan Stringeiksi
BeanContainer<String, HenkiloBean> beans =
new BeanContainer<String, HenkiloBean>(HenkiloBean.class);
// Käytetään nimeä Itemin Id:nä
beans.setBeanIdProperty("nimi");
// Lisätään säiliöön papuja
beans.addBean(new HenkiloBean("121280-1212", "Mainio Matti",
```

```
  "Mattilantie 21", "0202020"));
beans.addBean(new HenkiloBean("111190-1111", "Meikäläinen
  Maija", "Keskuskatu 2", "1212121"));
beans.addBean(new HenkiloBean("211282-2222", "Pätkä Pekka",
  "Pätkätie 42", "1234512"));
beans.addBean(new HenkiloBean("280275-0101", "Sähäkkä Sini",
  "Torikatu 20 A 14", "5433122"));
// Yhdistetään BeanContainer taulukkoon
Table table = new Table("Henkilöt", beans);
table.setHeight("120");
main.addComponent(table);
```

Saadaan kuvan 8.3 mukainen taulukko.

Henkilöt

HETU	NIMI	OSOITE	PUHELIN
121280-1212	Mainio Matti	Mattilantie 21	0202020
111190-1111	Meikäläinen Maija	Keskuskatu 2	1212121
211282-2222	Pätkä Pekka	Pätkätie 42	1234512
280275-0101	Sähäkkä Sini	Torikatu 20 A 14	5433122

Kuva 8.3: Taulukko, johon on yhdistetty BeanContainer

9 Ulkoasun muokkaus

9.1 Ulkoasukomponentit

Vaatimessa käyttöliittymäkomponentit voidaan jakaa kahteen luokkaan: Komponentit, joita käyttäjä voi käyttää, sekä ulkoasukomponentit, joita käytetään ulkoasun muokkauksessa sijoittamaan toiset komponentit tietyille paikoille käyttöliittymässä. *VerticalLayout* ja *HorizontalLayout* ovat kaksi tärkeintä ulkoasukomponenttia Vaatimessa. Niiden avulla komponentit voi sijoittaa joko pysty- tai vaakasuoraan. Esimerkiksi jotkut komponentit kuten *Window* ja *Panel* sisältävät VerticalLayoutin oletusulkoasunaan, mutta näitäkin voi itse halutessaan muuttaa.

9.2 HTML-asettelut

Ulkoasukomponenttien sijaan ulkoasua voi muokata käyttämällä HTML-asetteluja (engl. *HTML-template*) Vaatimen *CustomLayout*-komponentin avulla. Nämä asettelut tarjoavat sijainnit kaikille sovelluksen komponenteille. Tällä tavalla ohjelmoija voi antaa WWW-sivujen suunnittelijan vastata ulkoasusta käyttäen heidän omia työkalujaan, koska ulkoasun saa tällöin täysin erotettua muusta koodista. Toisaalta tällöin menettää mahdollisuuden muokata ulkoasua dynaamisesti [5, s. 155, 186].

9.3 Teemat

Vaadin erottaa käyttöliittymän logiikan ja ulkoasun toisistaan teemojen avulla. Teemat määrittelevät sovelluksen visuaalisen ulkoasun. Teemat voivat sisältää CSS-tyylitiedostoja, muokattuja HTML-asetteluja sekä grafiikkaa. Sovellus voi käyttää eri teemaa eri käyttäjille ja vaihtaa teemoja suorituksen aikana [5, s. 199–200].

Vaadin sisältää valmiita teemoja, joihin ei yleensä tarvitse tehdä paljoa muutoksia, mutta tarvittaessa teemoja on mahdollista myös tehdä itse. Myös teemoja voi luoda Eclipsen Vaadin-liitännäisellä. Teeman pitää sisältää *styles.css*-tyylitiedosto, mutta muun sisällön nimeäminen on vapaata. Vaadin suosittelee nimeämismallia, jossa on kansio *img* kuville, kansio *layouts* ulkoasumalleille ja kansio *css* css-tiedostoille [5, s. 3, 200].

9.4 CSS:n käyttäminen

Jokaisella Vaatimen komponentilla on oma CSS-tyyliluokka, jonka avulla sen ulkoasua ja sijaintia voidaan muuttaa. Esimerkiksi Label-komponentin CSS-tyyliluokka on v-label. Yleensä tyyliluokan nimi on komponentin nimi, jonka eteen liitetään "v-". Metodilla *addStyleName(String tyylinimi)* saa asetettua tietylle komponentille oman tyylinimen, jolloin saadaan muokattua sen tyylinimen omaavia komponentteja [5, s. 77].

Esimerkiksi koodilla

```
Label label = new Label("Jotain tekstiä");
label.addStyleName("tausta");
```

asetetaan labelin tyylinimeksi *tausta*, jota voidaan muokata CSS:ssä seuraavasti:

```
.tausta {
  background: red;
}
```

Tällöin, jos muilla komponenteilla ei ole samaa tyylinimeä, muokataan ainoastaan kyseistä komponenttia. Tyylinimen komponentille saa asetettua myös suoraan visuaalisella Vaadin Editorilla.

Jos sovellukseen haluaa sisällyttää CSS-tyylejä, pitää projektiin luoda ensin uusi teema. Tällöin syntyy tiedosto *styles.css*, johon haluamansa tyylit voi kirjoittaa. Tämä *styles.css* sisältää oletuksena myös Vaatimen oletusteeman tyylit.

Vaadin sisältää myös ulkoasukomponentin *CssLayout*, jonka avulla saa tehtyä CSS:n avulla täsmälleen haluamansa näköisen ulkoasun. CssLayoutin oletusvälistykset ja ryhmittelyt ovat varsin yksinkertaisia, joten siitä on helppo muokata haluamansa näköinen. CssLayoutissa jokainen komponentti sisältyy yksinkertaiseen DOM-rakenteeseen, joka koostuu -elementeistä. Oletuksena komponentit asetellaan vaakariviin ja ne vaihtavat luonnollisesti riviä, kun eivät enää mahdu rivil-le. Näitä ja monia muita ominaisuuksia voi halutessaan muuttaa CSS:n avulla [5, s. 176].

9.5 Visuaalinen ulkoasueditori

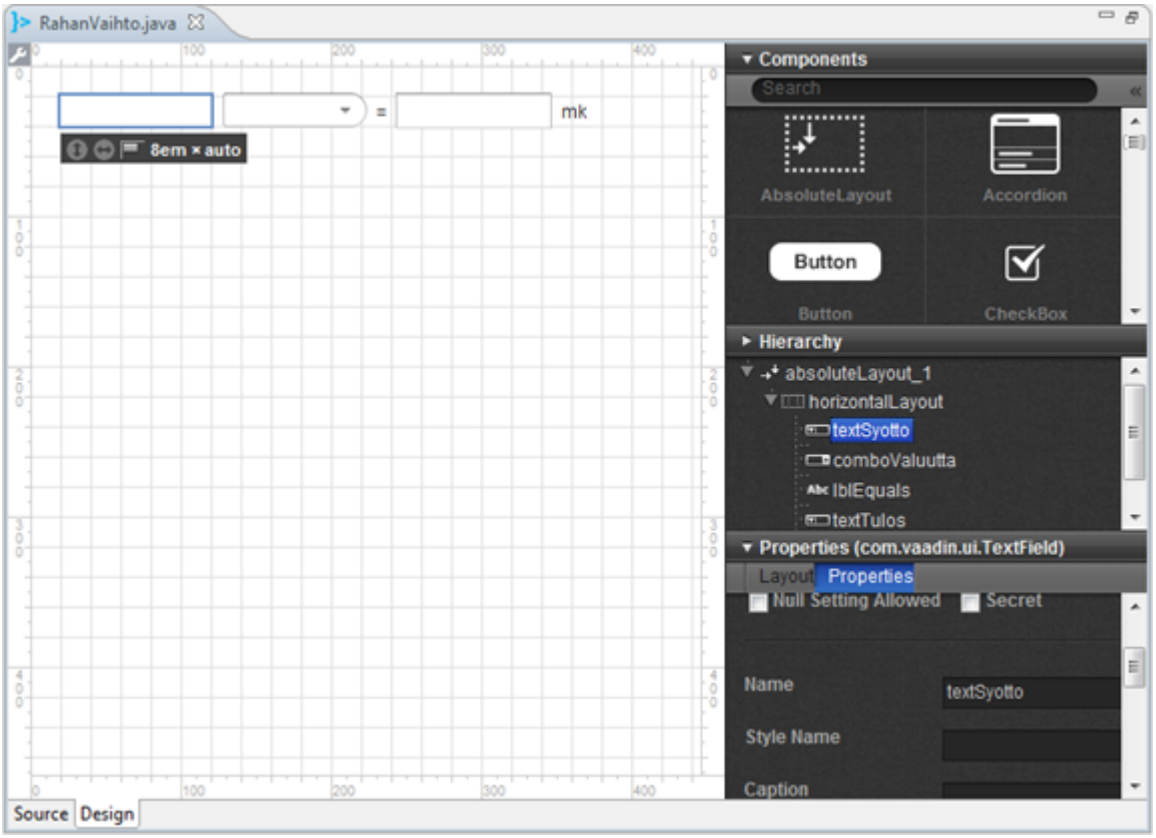
Vaadin tarjoaa lisäksi visuaalisen ulkoasueditorin (kuva 9.1). Eclipsen Vaadin-liitännäinen sisältää editorin, jolla voi luoda käyttöliittymiä visuaalisesti. Editori luo

koodin, joka saa aikaan käyttöliittymän. Koodi on suunniteltu uudelleenkäytettäväksi, joten perusulkoasun voi suunnitelta editorilla, ja rakentaa sen jälkeen vuorovaikutuslogiikan luodun koodin päälle [5, s. 189].

Visuaalista editoria pääsee käyttämään, kun luo projektiin *Vaadin Composite* -komponentin. Tuo komponentti pitää sitten sisällyttää haluamaansa ikkunaan *setContent*-metodilla. Valitsemalla tuossa komponentissa välilehden *Design*, aukeaa visuaalinen editori.

Editorin käyttäminen onnistuu yksinkertaisimmillaan siten, että ikkunaan vetää oikeanpuoleisesta valikosta komponentteja haluamiinsa kohtiin. Pohjana tuossa on siis *AbsoluteLayout*, jonka päälle kaikki muut komponentit laitetaan. Editorilla voi laittaa myös *HorizonalLayout*- ja *VerticalLayout*-ulkoasukomponentteja, jotka siis sijoittavat niihin laitetut komponentit vierekkäin tai alekkain. Ulkoasukomponentteja voi myös laittaa toistensa sisään. Editori sisältää myös *VerticalSplitpanelin* ja *HorizonalSplitpanelin*, jotka kai jotenkin jakavat ikkunan kahtia joko pysty- tai vaakasuunnassa. Noiden käyttäminen on kuitenkin aika hankalaa, eikä ohjeitakaan noiden käyttöön oikein löydy.

Ohjeita tuon editorin käyttöön oli muutenkin aika niukasti. Kirjassa esiteltiin ainoastaan, kuinka komponentteja sijoitellaan tuon AbsoluteLayoutin päälle. Lisäksi editorissa olisi ollut hyvä olla ominaisuus, jolla komponenteille olisi voinut luoda tapahtumankäsittelijät suoraan editorista käsin (kuten esimerkiksi WindowBuilderissa), kun nyt ne joutuu itse kirjoittamaan sinne koodiin. Ongelmia tuon editorin kanssa tuli myös siinä, ettei missään ohjeissa kerrottu, että kun tuon Vaadin Composite -komponentin luo, niin se pitää lisätä siihen ikkunaan setContent-metodilla, eikä *addComponent*-metodilla, jota jokaisessa kirjan esimerkissä oli käytetty. Tuota addComponent-metodia käytettäessä piti *setCompositionRoot*-metodiin muuttaa parametriksi se layout, jonka sisään komponentit oli laitettu. Eikä muut layoutit sitten ikkunassa näkyneet, vaikka niitä olisi siihen laittanut.

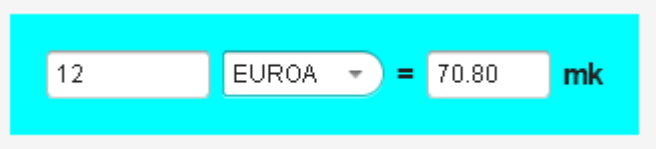


Kuva 9.1: Vaatimen ulkoasueditori

10 Esimerkkisovellus: Valuuttamuunnin

10.1 Ohjelmointi 2 -kurssin esimerkki Vaatimella tehtynä

Tässä luvussa esitellään kuinka Vesa Lappalaisen Ohjelmointi 2 -kurssilla esimerkkinä käyttämä Valuuttamuunnin muutetaan Vaatimella WWW-sovellukseksi. Varsinainen Valuuttamuuntimen koodi (liite B.3) on siis Vesa Lappalaisen tekemää. Myös Vaadin-koodissa (RahanVaihto.java, liite B.2) on joitakin Vesan muokkauksia ja tyylien lisääminen on kokonaan Vesan tekemä. Sovellus tehdään Eclipsen Vaadin-liitännäisellä käyttäen visuaalista Vaadin Editoria. Vaadin-liitännäisen asentaminen Eclipseen ja uuden projektin luominen esitellään liitteessä A. Sovelluksen koodit löytyy kokonaisuudessaan liitteestä B.



Kuva 10.1: Valuuttamuunnin Vaatimella tehtynä

10.2 RahanvaihtoApplication.java

Projektin luominen luo *src*-kansioon automaattisesti paketin *com.example.rahanvaihto* ja sinne luokan *RahanVaihtoApplication*.

Ensin, että voidaan käyttää jo olemassa olevaa koodia, lisätään *src*-kansioon paketti *demo.d11* ja sinne luokka *ValuuttaMuunnos.java* (liite B.3).

Lisätään projektiin myös Ohjelmointi 2 -kurssilla käytetty *Ali.jar*.

RahanVaihtoApplication-luokassa on metodi *init()*, jota kutsutaan kun sovellus käynnistyy. Lisätään sovellukseen ensin pääikkuna ja siihen sitten uusi *Vaadin Composite* -komponentti, joka luodaan seuraavaksi. Lisätään siis *init()*-metodiin:

```
Window mainWindow = new Window("Rahanvaihto");
setMainWindow(mainWindow);
mainWindow.setContent(new RahanVaihto(mainWindow));
```

10.3 RahanVaihto.java

Seuraavaksi luodaan *Vaadin Composite*-komponentti *RahanVaihto*, jota voidaan sitten muokata Vaadin Editorilla.

Komponentti luodaan painamalla *com.example.rahanvaihto*-paketin päällä hiiren oikeaa nappia ja valitsemalla: *New/Other/Vaadin/Vaadin Composite*. Annetaan komponentille nimeksi *RahanVaihto*. Tätä komponenttia voidaan siis muokata Vaadin Editorilla. Jos koodi-ikkunan alapuolella ei ole *Source/Design*-välilehtiä, niin pitää klikata ikkunaa hiiren oikealla napilla ja valita *Open With/Vaadin Editor*. Visuaaliseen editoriin pääsee nyt siis valitsemalla *Design*.

Ensin lisätään *absoluteLayoutin* päälle *horizontalLayout*, joka järjestää komponentit vierekkäin. Laitetaan se vaikka vasempaan yläkulmaan. Vaihdetaan *Properties*-kohdasta horizontalLayoutin nimeksi *rahaLayout*. Editorin oikeanpuoleisen valikon *Properties*-kohdassa *Layout*-välilehdellä saa asetettua layoutille pituuden ja korkeuden. Laitetaan myös ruksit kohtiin *Margin* ja *Spacing*. Margin lisää väliä layoutin ja muiden komponenttien väliin ja Spacing lisää väliä layoutin sisällä olevien komponenttien väliin.

Lisätään sitten tuohon rahaLayoutiin noita tekstilaatikoita (TextField), comboBoxi ja labeleita. Komponentteja voi lisätä myös oikeanpuoleisen valikon *Hierarchy*-kohtaan. Annetaan komponenteille taulukon 10.1 mukaiset nimet.

Taulukko 10.1: Valuuttamuuntimen komponenttien nimet.

editRaha	cbValuutta	labelOn	editTulos	labelKantavaluutta
12	EUROA	=	70.80	mk

Nyt kun komponentit on luotu ja ulkoasua muokattu, niin voidaan kirjoittaa itse koodi, joka hoitaa toiminnot. Valitsemalla *Source*-välilehti pääsee käsittelemään itse koodia. Varsinainen oma koodi tulee nyt tuohon *public RahanVaihto()*-metodiin. Tuon metodin jälkeen oleva editorin luoma koodi on ulkoasukoodia, jonne ei itse tarvitse välttämättä tehdä mitään muutoksia.

Lisätään sitten attribuutit:

```
/**
 * @return loota johon raha kirjoitetaan
 */
protected TextField getEditRaha() { return editRaha; }
```

```
/* static jotta kaikki käyttävät samaa valuuttataulukkoa */
private static ValuuttaMuunnos.Valuutat valuutat;
static {
     valuutat = new ValuuttaMuunnos.Valuutat();
     try {
         valuutat.lue("http://users.jyu.fi/~vesal/kurssit/
         ohj2/luennot/11/luentoW2/servlet/valuutat.dat");
     } catch (IOException e) {
         System.err.println("Valuuttoja ei saada luettua");
     }
}
```

Lisätään sitten koodi tietojen alustamiseksi sekä tapahtumien käsittelemiseksi:

```
public RahanVaihto(Window parent) {
    buildMainLayout();
    setCompositionRoot(mainLayout);

    String[] nimet = valuutat.getValuuttojenNimet();
    cbValuutta.removeAllItems();

    // Asetetaan, ettei null-valinta ole mahdollinen.
    // Näin saadaan ComboBoxin alusta pois yksi tyhjä rivi.
    cbValuutta.setNullSelectionAllowed(false);

    // Täytetään ComboBoxiin valuuttojen nimet
    for (String item:nimet) cbValuutta.addItem(item);
    // Asetetaan ComboBoxille oletusarvo.
    cbValuutta.setValue(nimet[1]);

    labelKantavaluutta.setValue(nimet[0]);

    /* Asetetaan muutokset tekstilaatikossa välittymään
     * nopeammin palvelimelle. Näin saadaan tulos näkyviin
     * välittömästi numeron syöttömisen jälkeen.
     * Oletusmoodi tälle on LAZY joka välittää muutoksen,
     * kun tekstin syötössä on tauko.
     */
```

```
editRaha.setTextChangeEventMode(
  TextChangeEventMode.EAGER);

/* Asetetaan comboboxiin, että laukaisee eventin heti,
 * kun valintaa muutetaan */
cbValuutta.setImmediate(true);

// Luodaan tapahtumienkäsittelijät tekstilaatikolle
// ja comboboxille. Lasketaan tulos aina, kun
// tekstilaatikkoon syötetään arvoja tai kun
// valitaan comboboxista uusi valuutta.
editRaha.addListener(new TextChangeListener() {
    private static final long serialVersionUID = 1L;
    @Override
    public void textChange(TextChangeEvent event) {
        laskeTulos(event.getText());
    }
});
cbValuutta.addListener(new Property.ValueChangeListener()
{
    private static final long serialVersionUID = 1L;
    @Override
    public void valueChange(ValueChangeEvent event) {
        laskeTulos((String)getEditRaha().getValue());
    }
});
// Keyboard navigation - enter key is a shortcut to
// selectAll
parent.addActionHandler(new Action.Handler() {
    private static final long serialVersionUID = 1L;
    @Override
    public Action[] getActions(Object target, Object
    sender) {
        return new Action[] { new ShortcutAction("Login",
        ShortcutAction.KeyCode.ENTER, null) };
    }
```

```
        @Override
        public void handleAction(Action action,
          Object sender, Object target) {
            getEditRaha().selectAll();
        }
    });
}
```

Ja lopuksi vielä metodi, joka laskee tuloksen:

```
 /**
  * Lasketaan rahanvaihdon tulos
  * @param rahaLuettu rahamäärä joka muutetaan
  */
  protected void laskeTulos(String rahaLuettu) {
      String nimi = (String)cbValuutta.getValue();
      if ( nimi == null ) return;
      double maara = Mjonot.erotaDouble(rahaLuettu,0);
      double tulos = valuutat.getVaihdettuMaara(nimi, maara);
      editTulos.setValue(Mjonot.fmt(tulos,4,2));
  }
```

10.4 Tyylien lisääminen

Tyylien lisäämistä varten pitää projektiin luoda uusi teema. Painetaan projektin nimen päällä hiiren oikealla ja valitaan *New/other/Vaadin/Vaadin Theme* ja klikkaillaan hyväksymistä.

Syntyy uusi tiedosto: *WebContent/VAADIN/themes/rahanvaihtotheme/styles.css*, joka siis sisältää oletuksena oletusteeman tyylit. Muokataan *styles.css* muotoon:

```
@import url(../reindeer/styles.css);

.iso {
  font-size: larger;
  font-weight:bold;
}

.tausta {
```

```
  background: cyan;
}
```

Komponenttiin saa tietyn tyylin asetettua lisäämällä komponentille tyylin nimen designerissa (kohtaan *Style name*).

Koodissa näkyy esimerkiksi: *rahaLayout.setStyleName("tausta");*

10.5 Ohjelman ajaminen

Ohjelma kannattaa ajaa klikkaamalla projektin päällä hiiren oikeaa nappia ja valitsemalla *Run as/Run on Server*. Näin ohjelma aukeaa suoraan oikeassa osoitteessa.

Esimerkiksi: *http://localhost:8080/rahanvaihto/*.

11 Yhteenveto

Vaadin on siis suomalainen palvelinpohjainen Java-kehys rikkaiden WWW-sovellusten kehittämiseen. Vaadin muuttaa Java-koodin Google Web Toolkitin avulla Javascriptiksi, joka sitten esitetään selaimessa. Ulkoasua pystyy lisäksi muokkaamaan CSS:n ja HTML-mallinteiden avulla. Eclipseen on tarjolla Vaadin-liitännäinen, jonka avulla Vaadin-sovelluksia voi luoda. Eclipsen Vaadin-liitännäinen sisältää myös visuaalisen Vaadin Editorin, jolla sovelluksen ulkoasun pystyy muokkaamaan visuaalisesti.

Vaatimen käyttö on varsin helppoa, kun sen oppii. Ohjelmointi vastaa hyvin pitkälti Swingillä ohjelmointia. Mutta esimerkiksi WindowBuilderiin verrattuna, Vaadin Editori on varsin yksinkertainen. Editorilla ei pysty esimerkiksi luomaan komponenteille tapahtumankäsittelijöitä, kuten WindowBuilderilla. Vaatimella tapahtumankäsittelijät pitää siis kirjoittaa koodiin itse.

Kokemukseni mukaan HTML:ää tai Javascriptiä ei tosiaankaan Vaadinta käytettäessä tarvinnut osata lainkaan, CSS:ää kylläkin. Esimerkiksi komponenttien taustavärejä ei saanut asetettua suoraan vaan ne piti tehdä CSS-tyylien avulla.

Tässä tutkielmassa ei esitelty kaikkia Vaatimen ominaisuuksia. Jos Vaatimeen haluaa tutustua tarkemmin, suosittelen lukemaan tuota Book Of Vaadin -kirjaa [5].

Lähteet

[1] Alessandro Bozzon, Sara Comai, Piero Fraternali, Giovanni Toffetti Carughi, *Conceptual Modeling and Code Generation for Rich Internet Applications*, Dipartimento di Elettronica e Informazione Politecnico di Milano, 2006.

[2] Dave Crane, Eric Pascarello, Darren James, *Ajax in Action*, Manning Publications Co., 2006.

[3] Neal Ford, *Art of Java Web Development*, Manning Publications Co., 2004.

[4] Jesse James Garrett, *Ajax: A New Approach to Web Applications*, saatavilla WWW-muodossa <URL: `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`>, 18.02.2005.

[5] Marko Grönroos, *Book Of Vaadin: 4th Edition*, Vaadin Ltd, 2011, saatavilla PDF-muodossa <URL: `https://vaadin.com/download/book-of-vaadin/current/pdf/book-of-vaadin.pdf/`>.

[6] Josefina Guerrero-García, Juan Manuel González-Calleros, Jean Vanderdonckt, Jaime Muñoz-Arteaga, *2009 Latin American Web Congress - A Theoretical Survey of User Interface Description Languages: Preliminary Results*, IEEE Computer Society, 2009.

[7] Anthony T. Holdener III, *Ajax: The Definitive Guide*, OReilly Media, Inc, 2008.

[8] Jonas Jacobi ja John R. Fallows, *Pro JSF and Ajax: Building Rich Internet Components*, Apress, 2006.

[9] Bernard Kohan, *Guide to Web Application Development*, saatavilla WWW-muodossa <URL: `http://www.comentum.com/guide-to-web-application-development.html`>, viitattu 30.03.2012.

[10] Alessandro Lacava, *Speed Up Your AJAX-based Apps with JSON*, saatavilla WWW-muodossa <URL: `http://www.devx.com/webdev/Article/32651`>, 05.10.2006.

[11] Stuart Langridge, *DHTML Utopia: Modern Web Design Using JavaScript DOM*, SitePoint Pty. Ltd., 2005.

[12] Arvo Lipitsäinen, *DOM - Document Object Model*, saatavilla WWW-muodossa <URL: `http://myy.helia.fi/~atk90d/domARVO/dom.html`>, viitattu 31.05.2012.

[13] Michael Nash, *Java Frameworks and Components: Accelerate Your Web Application Development*, Cambridge University Press, 2004.

[14] Thomas A. Powell, *Ajax: The Complete Reference*, The McGraw-Hill Companies, 2008.

[15] Nathaniel T. Schutta ja Ryan Asleson, *Pro Ajax and Java*, Apress, 2006.

[16] Stoyan Stefanov, *Object-Oriented JavaScript*, Packt Publishing Ltd., 2008.

[17] *Vaadin FAQ*, saatavilla WWW-muodossa <URL: `https://vaadin.com/faq`>, viitattu 30.05.2012.

[18] Jevon M. Wright ja Jens B. Dietrich, *Requirements for Rich Internet Application Design Methodologies*, Institute of Information Sciences and Technology, Massey University, Palmerston North, New Zealand, 2008.

A Vaatimen asentaminen ja projektin luominen

A.1 Vaadin-liitännäisen asentaminen Eclipseen

- HUOM! Vaadin ei tue vielä Java 7 -versiota. Eli koneelle tulisi olla asennettuna myös vanhempi Java-versio.
- Valitse Eclipsessä: Help/Install New Software
- Laita "Work with:"-kohtaan osoitteeksi http://vaadin.com/eclipse jos käytössäsi on Eclipse 3.5 tai uudempi versio.

  Vanhemmilla versioilla osoite on http://vaadin.com/eclipse/3.4, näihin ei ole kuitenkaan saatavilla visuaalista Vaadin Editoria.
- Listaan pitäisi tulla ainakin kohta "Vaadin", josta nuolen aukaisemalla saat valita mitä haluat asentaa. Valitse ainakin *Vaadin Plug-in for Eclipse*.
- *Book of Vaadin*, jonka tuosta myös voi valita, on ladattavissa ja luettavissa myös osoitteessa *https://vaadin.com/book*.
- Valitse Next ja jatka ohjeiden mukaan.

A.2 Uuden projektin luominen

- Luo uusi projekti: File/New/Other/Vaadin/Vaadin Project
- Anna projektille nimi.
- Katso että Target runtime -kohdassa on palvelin jota aiot käyttää esim. Apache Tomcat v7.0.
- Configuration kohtaan kannattaa varmaan valita mahdollisimman uusi Java- ja Servlet -versio esim. Vaadin, Java 6, Servlet 3.0. Muista, että Vaadin ei tue Java 7:aa.
- Valitse Finish.

B Valuuttamuunnin

Tässä on tutkielman esimerkkisovelluksen koodit kokonaisuudessaan. Tämä on nyt Vesa Lappalaisen tekemänä, jotta se vastaisi suoraan tuota esimerkkiä, koska siihenkin tuli lopulta se Vesan versio.

B.1 RahanvaihtoApplication.java

```
package com.example.rahanvaihto;

import com.vaadin.Application;
import com.vaadin.ui.*;

/**
 * Rahanvaihto Vaadin-kehittimellä
 * @author vesal
 * @version May 6, 2012
 */
public class RahanvaihtoApplication extends Application {
  private static final long serialVersionUID = 1L;

  @Override
  public void init() {
    Window mainWindow = new Window("Rahanvaihto");
    setMainWindow(mainWindow);

    mainWindow.setContent(new RahanVaihto(mainWindow));

    setTheme("rahanvaihtotheme");
  }
}
```

B.2 RahanVaihto.java

```
package com.example.rahanvaihto;

import java.io.IOException;
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
import com.vaadin.event.Action;
import com.vaadin.event.FieldEvents.TextChangeEvent;
import com.vaadin.event.FieldEvents.TextChangeListener;
import com.vaadin.event.ShortcutAction;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.AbstractTextField.TextChangeEventMode;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.ComboBox;
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextField;
import com.vaadin.ui.Window;
import demo.d11.ValuuttaMuunnos;
import fi.jyu.mit.ohj2.Mjonot;

/**
 * Komponentti, joka tekee rahanvaihtoa Ohj2-kurssilla
 * tehdyn Valuutat-luokan avulla.
 * Komponentissa on osat:
 * <pre>
 * editRaha cbValuutta labelOn editTulos labelKantavaluutta
 *   1.0      EUROA       =      5.90      mk
 * </pre>
 * Pääikkunaan lisätään ShortCut Enter näppäimen painamista
 * varten niin, että silloin maalataan editRaha kokonaan.
 * @author vesal
 * @version May 6, 2012
 */
```

```
@SuppressWarnings("deprecation")
public class RahanVaihto extends CustomComponent  {
    @AutoGenerated
    private AbsoluteLayout mainLayout;
    @AutoGenerated
    private HorizontalLayout rahaLayout;
    @AutoGenerated
    private TextField editRaha;
    @AutoGenerated
    private Label labelKantavaluutta;
    @AutoGenerated
    private TextField editTulos;
    @AutoGenerated
    private Label labelOn;
    @AutoGenerated
    private ComboBox cbValuutta;

    private static final long serialVersionUID = 1L;

    /**
     * @return loota johon raha kirjoitetaan
     */
    protected TextField getEditRaha() { return editRaha; }

    // static jotta kaikki käyttävät samaa valuuttataulukkoa
    private static ValuuttaMuunnos.Valuutat valuutat;
    static {
        valuutat = new ValuuttaMuunnos.Valuutat();
        try {
            valuutat.lue("http://users.jyu.fi/~vesal/kurssit/
                          ohj2/luennot/11/luentoW2/servlet/
                          valuutat.dat");
        } catch (IOException e) {
            System.err.println("Valuuttoja ei saada
                               luettua");
        }
    }
```

```
/**
 * The constructor should first build the main layout,
 * set the composition root and then do any custom
 * initialization.
 *
 * The constructor will not be automatically
 * regenerated by the visual editor.
 * @param parent kuka on tämän isäikkuna, jolle voi
 * laittaa mm. shortCutteja
 */
public RahanVaihto(Window parent) {
    buildMainLayout();
    setCompositionRoot(mainLayout);

    // Täytä ComboBoxiin valuuttojen nimet
    String[] nimet = valuutat.getValuuttojenNimet();
    cbValuutta.removeAllItems();
    cbValuutta.setNullSelectionAllowed(false);

    for (String item:nimet) cbValuutta.addItem(item);
    cbValuutta.setValue(nimet[1]);

    labelKantavaluutta.setValue(nimet[0]);

    /* Asetetaan muutokset tekstilaatikossa välittymään
     * nopeammin palvelimelle. Näin saadaan tulos
     * näkyviin välittömästi numeron syöttämisen jälkeen.
     * Oletusmoodi tälle on LAZY joka välittää muutoksen
     * kun tekstin syötössä on tauko.
     */
    editRaha.setTextChangeEventMode(
        TextChangeEventMode.EAGER);

    /* Asetetaan comboboxiin, että laukaisee eventin
     * heti, kun valintaa muutetaan */
```

```
        cbValuutta.setImmediate(true);

        editRaha.addListener(new TextChangeListener() {
            private static final long serialVersionUID = 1L;
            @Override
            public void textChange(TextChangeEvent event) {
                laskeTulos(event.getText());
            }
        });
        cbValuutta.addListener(new
          Property.ValueChangeListener() {
            private static final long serialVersionUID = 1L;
            @Override
            public void valueChange(ValueChangeEvent event) {
                laskeTulos((String)getEditRaha().getValue());
            }
        });
        // Keyboard navigation - enter key is a shortcut to
        // selectAll
        parent.addActionHandler(new Action.Handler() {
            private static final long serialVersionUID = 1L;
            @Override
            public Action[] getActions(Object target,
              Object sender) {
                return new Action[] { new ShortcutAction(
                  "Login", ShortcutAction.KeyCode.ENTER,
                  null) };
            }

            @Override
            public void handleAction(Action action,
              Object sender, Object target) {
                getEditRaha().selectAll();
            }
        });
    }
```

```
/**
 * Lasketaan rahanvaihdon tulos
 * @param rahaLuettu rahamäärä joka muutetaan
 */
protected void laskeTulos(String rahaLuettu) {
    String nimi = (String)cbValuutta.getValue();
    if ( nimi == null ) return;
    double maara = Mjonot.erotaDouble(rahaLuettu,0);
    double tulos = valuutat.getVaihdettuMaara(nimi,
      maara);
    editTulos.setValue(Mjonot.fmt(tulos,4,2));
}


@AutoGenerated
private AbsoluteLayout buildMainLayout() {
    // common part: create layout
    mainLayout = new AbsoluteLayout();
    mainLayout.setImmediate(false);
    mainLayout.setWidth("100%");
    mainLayout.setHeight("100%");
    mainLayout.setMargin(false);

    // top-level component properties
    setWidth("100.0%");
    setHeight("100.0%");

    // rahaLayout
    rahaLayout = buildRahaLayout();
    mainLayout.addComponent(rahaLayout,
    "top:20.0px;left:20.0px;");

    return mainLayout;
}
```

```
@AutoGenerated
private HorizontalLayout buildRahaLayout() {
    // common part: create layout
    rahaLayout = new HorizontalLayout();
    rahaLayout.setStyleName("tausta");
    rahaLayout.setImmediate(false);
    rahaLayout.setWidth("-1px");
    rahaLayout.setHeight("60px");
    rahaLayout.setMargin(true);
    rahaLayout.setSpacing(true);

    // editRaha
    editRaha = new TextField();
    editRaha.setImmediate(false);
    editRaha.setWidth("82px");
    editRaha.setHeight("-1px");
    editRaha.setTextChangeTimeout(0);
    editRaha.setSecret(false);
    rahaLayout.addComponent(editRaha);

    // cbValuutta
    cbValuutta = new ComboBox();
    cbValuutta.setImmediate(false);
    cbValuutta.setWidth("81px");
    cbValuutta.setHeight("-1px");
    rahaLayout.addComponent(cbValuutta);

    // labelOn
    labelOn = new Label();
    labelOn.setStyleName("iso");
    labelOn.setImmediate(false);
    labelOn.setWidth("-1px");
    labelOn.setHeight("-1px");
    labelOn.setValue("=");
    rahaLayout.addComponent(labelOn);
    rahaLayout.setExpandRatio(labelOn, 2.0f);
    rahaLayout.setComponentAlignment(labelOn,
```

```
            new Alignment(48));

        // editTulos
        editTulos = new TextField();
        editTulos.setImmediate(false);
        editTulos.setWidth("62px");
        editTulos.setHeight("-1px");
        editTulos.setSecret(false);
        rahaLayout.addComponent(editTulos);

        // labelKantavaluutta
        labelKantavaluutta = new Label();
        labelKantavaluutta.setStyleName("iso");
        labelKantavaluutta.setImmediate(false);
        labelKantavaluutta.setWidth("-1px");
        labelKantavaluutta.setHeight("-1px");
        labelKantavaluutta.setValue("??");
        rahaLayout.addComponent(labelKantavaluutta);
        rahaLayout.setComponentAlignment(labelKantavaluutta,
          new Alignment(33));

        return rahaLayout;
    }
}
```

B.3 ValuuttaMuunnos.java

Tässä on varsinaisen valuutanmuuntokoodin rajapinta-osuus. Ei siis metodien toteutuksia.

```
public class ValuuttaMuunnos {
    public static class Valuutat {
      private int lkm = 0;
      private Valuutta alkiot[] = new Valuutta[20];

      public boolean lue(BufferedReader fi)
        throws IOException {}
```

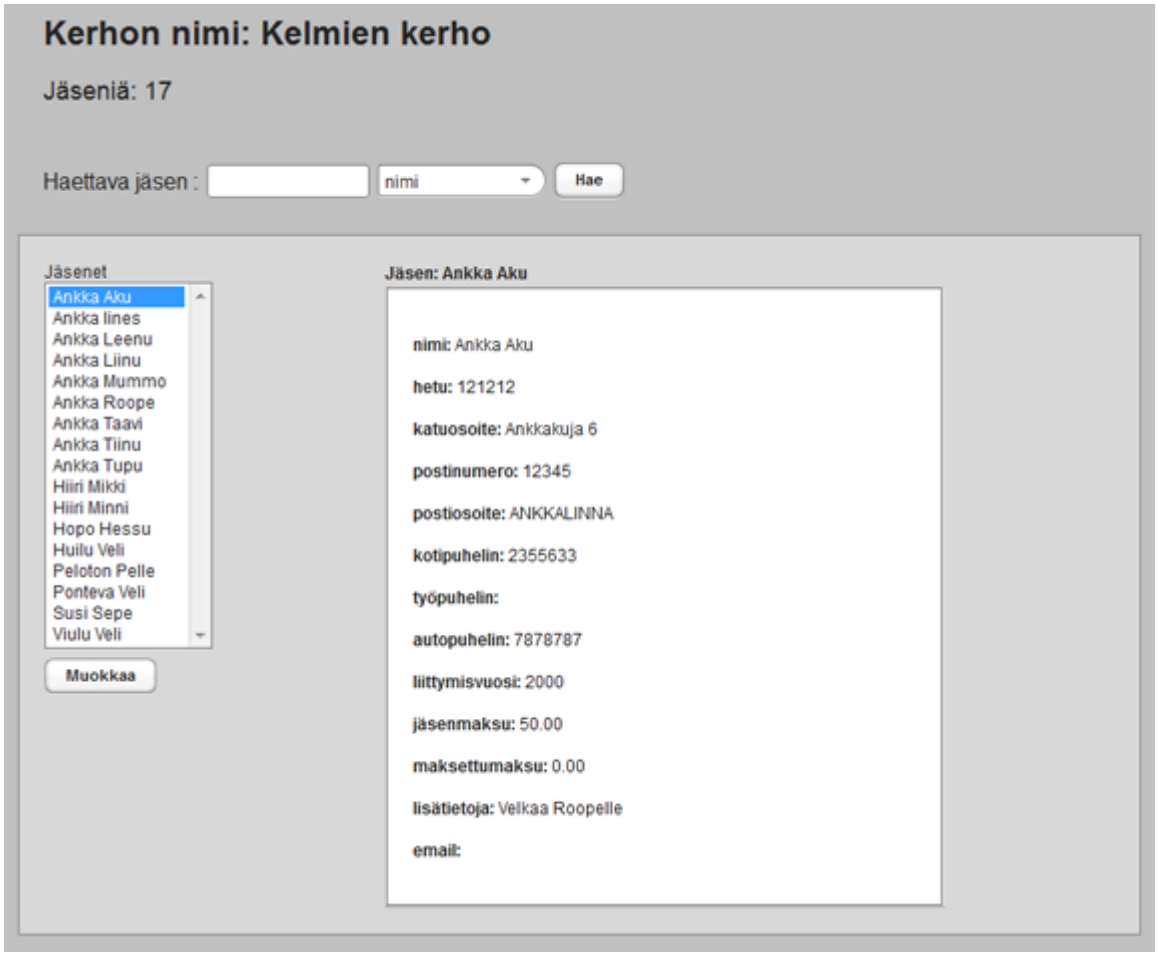
```
        public boolean lue(String nimi) throws IOException {}

        public String[] getValuuttojenNimet() {}

        public double getVaihdettuMaara(String nimi,
          double maara) {}
    }
}
```

C Kerho Vaatimella tehtynä

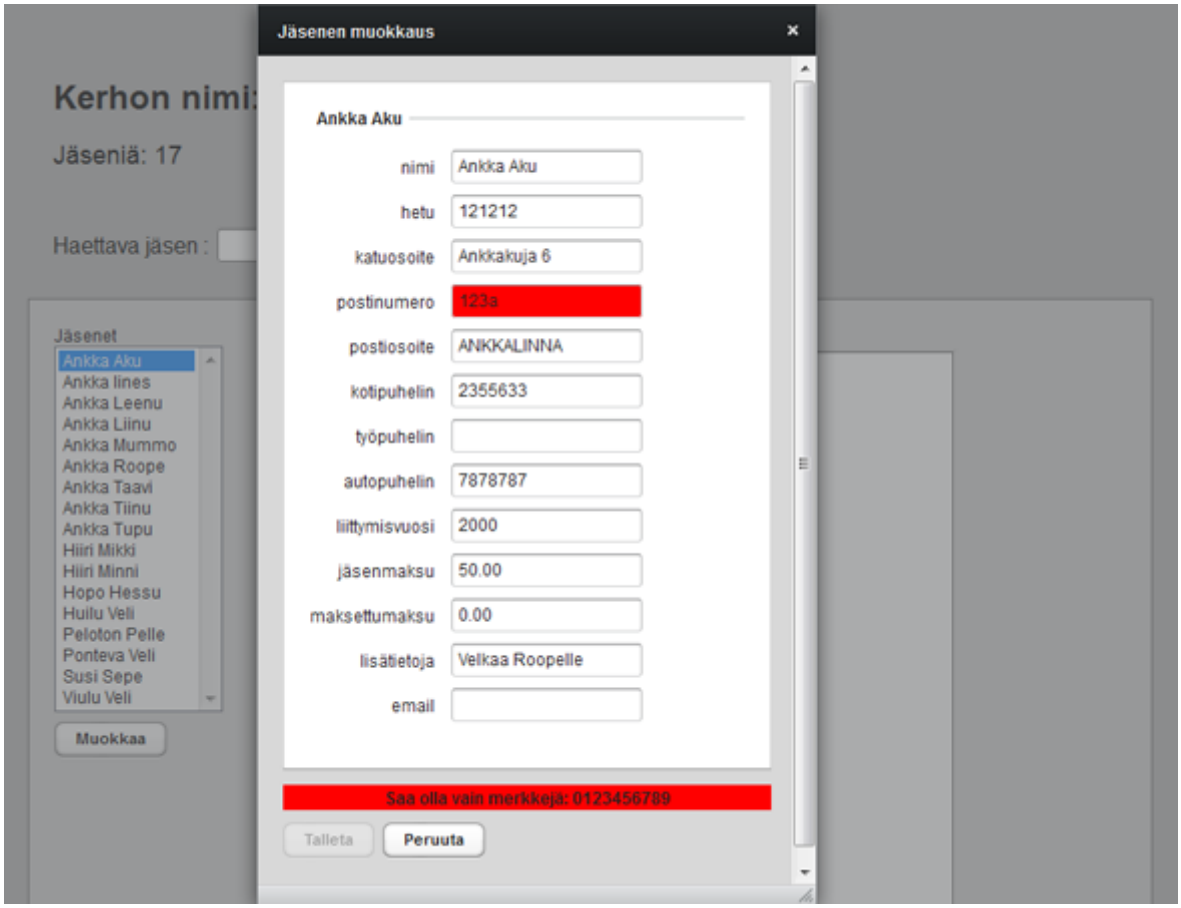
Tässä on Ohjelmointi 2 -kurssin esimerkkiharjoitustyö Kerho tehtynä Vaatimella WWW-sovellukseksi. Tämän tekemiseen tarvitut Kerhon koodit sekä KerhoBean löytyvät svn:stä:

Kerho ja kanta: https://svn.cc.jyu.fi/srv/svn/ohj2/vesal11/trunk/src/

KerhoBean: https://svn.cc.jyu.fi/srv/svn/ohj2/vesalJsp/trunk/src/jspkerho/



Kuva C.1: Kerho Vaatimella tehtynä



Kuva C.2: Kerhon jäsenenmuokkausikkuna

C.1 VaadinkerhoApplication.java

```
package com.example.vaadinkerho;

import com.vaadin.Application;
import com.vaadin.ui.*;

/**
 * Kerho Vaatimella toteutettuna
 * @author Joni Korkalainen
 * @version 30.05.2012
 */
public class VaadinkerhoApplication extends Application {
  @Override
  public void init() {
    Window main = new Window("Vaadinkerho");
    setMainWindow(main);

    // Laitetaan ikkunan sisällöksi Vaadinkerho
    main.setContent(new Vaadinkerho(main));

    setTheme("kerhoteema");
  }
}
```

C.2 Vaadinkerho.java

```
package com.example.vaadinkerho;

import java.util.Collection;
import jspkerho.KerhoBean;
import kerho.Jasen;
import kerho.Kerho;
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
```

```
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.AbstractSelect;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;
import com.vaadin.ui.ComboBox;
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.ListSelect;
import com.vaadin.ui.Panel;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Window;
import com.vaadin.ui.Window.CloseEvent;
import com.vaadin.ui.Window.CloseListener;

/**
 * Kerho Vaatimella toteutettuna
 * @author Joni Korkalainen
 * @version 30.05.2012
 */
public class Vaadinkerho extends CustomComponent {
    @AutoGenerated
    private AbsoluteLayout mainLayout;
    @AutoGenerated
    private VerticalLayout kerhoAlue;
    @AutoGenerated
    private HorizontalLayout jasenetAlue;
    @AutoGenerated
    private Panel panelJasen;
    @AutoGenerated
    private VerticalLayout jasenTiedot;
    @AutoGenerated
    private Label labelJasentiedot;
    @AutoGenerated
```

```
private VerticalLayout jasenetValinta;
@AutoGenerated
private Button buttonMuokkaa;
@AutoGenerated
private ListSelect listJasenet;
@AutoGenerated
private HorizontalLayout hakuLayout;
@AutoGenerated
private Button buttonHaku;
@AutoGenerated
private ComboBox cbHaku;
@AutoGenerated
private TextField editHaku;
@AutoGenerated
private Label labelHaku;
@AutoGenerated
private VerticalLayout otsikkoLayout;
@AutoGenerated
private Label labelJasenLkm;
@AutoGenerated
private Label labelOtsikko;

private Kerho kerho;
private Jasen jasenkohdalla;
private Jasen apujasen = new Jasen();
private Window mainwindow;
private Window mywindow;

/**
 * Oletusmuodostaja
 */
public Vaadinkerho() {

}

/**
 * The constructor should first build the main layout,
```

```
     * set the composition root and then do any custom
     * initialization.
     *
     * The constructor will not be automatically
     * regenerated by the visual editor.
     * @param main ikkuna johon tämä sovellus laitetaan
     */
    public Vaadinkerho(Window main) {
        mainwindow = main;
        buildMainLayout();

        setCompositionRoot(mainLayout);

        kerho = KerhoBean.getKerho();

        String nimi = kerho.getNimi();
        int jasenia = kerho.getJasenia();

        // Asetetaan labeleihin kerhon nimi ja jäsenmäärä.
        labelOtsikko.setValue("<h1>Kerhon nimi: "
                              + nimi
                              + "</h1>");
        labelJasenLkm.setValue("Jäseniä: " + jasenia);

        // Asetetaan, että valintalistan alkion otsikko on
        // aina asetettava erikseen. Näin saadaan alkion
        // otsikoksi vain jäsenen nimi, vaikka alkiona on
        // objekti Jasen.
        listJasenet.setItemCaptionMode(
          AbstractSelect.ITEM_CAPTION_MODE_EXPLICIT);
        // Asetetaan valintalistan nimien määrä samaksi
        // kuin jäsenien määrä.
        listJasenet.setRows(jasenia);
        // Poistetaan valintalistan alusta yksi tyhjä rivi,
        // eli ei sallita Null valintaa.
        listJasenet.setNullSelectionAllowed(false);
        // Laitetaan valinta muuttumaan heti kun valitaan
```

```
// uusi kohta.
listJasenet.setImmediate(true);

// Luodaan hakukentät
int cbIndex = 0;
for (int i=apujasen.ekaKentta();
  i<apujasen.getKenttia();i++) {
    // Asetetaan kentän arvoksi kokonaisluku ja
    // otsikoksi kysymys. Näin saadaan myöhemmin
    // getValue()-metodilla selville valittu
    // "indeksi".
    cbHaku.addItem(cbIndex);
    cbHaku.setItemCaption(cbIndex,
      apujasen.getKysymys(i));
    cbIndex++;
}

// Asetetaan ettei null-valinta ole mahdollinen.
// Näin saadaan pois comboboxin alusta yksi tyhjä
// rivi.
cbHaku.setNullSelectionAllowed(false);
cbHaku.setValue(0);

// Haetaan kerhon jäsenet valintalistaan.
hae(0);

// Asetetaan muokkausnapille kuuntelija.
buttonMuokkaa.addListener(new ClickListener() {
    @Override
    public void buttonClick(ClickEvent event) {
        jasenkohdalla = (Jasen) listJasenet.getValue();
        // Aukaistaan uusi ikkuna, jolle viedään
        // parametrina nykyinen ikkuna ja valittu
        // jäsen.
        mywindow = Vaadinjasen.addWindow(mainwindow,
          jasenkohdalla);
```

```
            final int valittu = ((Jasen)
              listJasenet.getValue()).getTunnusnro();

            // Kuunnellaan avatun ikkunan sulkemista,
            // jotta saadaan päivitettyä ikkunassa
            // muutetut tiedot.
            mywindow.addListener(new CloseListener() {
                @Override
                public void windowClose(CloseEvent e) {
                  hae(-1);
                  // Asetetaan valintalistaan valituksi
                  // sama jäsen kuin oli ikkunaa
                  // aukaistaessa.
                  listJasenet.setValue(kerho.annaJasenId(
                    valittu));
                  // Asetetaan muokkausnappi toimivaksi.
                  buttonMuokkaa.setEnabled(true);
                }
            });
        }
});

// Asetetaan hakunapille kuuntelija.
buttonHaku.addListener(new ClickListener() {
    @Override
    public void buttonClick(ClickEvent event) {
        hae(0);
    }
});

// Asetetaan jäsenlistalle kuuntelija, joka
// kuuntelee valinnan muuttumista.
listJasenet.addListener(new
  Property.ValueChangeListener() {
    private static final long serialVersionUID = 1L;
    @Override
    public void valueChange(ValueChangeEvent event) {
```

```
            jasenkohdalla = (Jasen) listJasenet.getValue();
            tulosta();
        }
    });

}


/**
 * Hakee jäsenet hakuehtojen mukaan ja näyttää ne
 * valintalistassa.
 * @param nro jos 0 niin asetetaan valituksi listan
 * ensimmäinen jäsen
 */
public void hae(int nro) {
    buttonMuokkaa.setEnabled(true);
    // Poistetaan listasta kaikki jäsenet.
    listJasenet.removeAllItems();
    int hakukentta = apujasen.ekaKentta()
      + Integer.parseInt(cbHaku.getValue().toString());
    Collection<Jasen> loytyneet = kerho.etsi(
      "*"+editHaku.getValue().toString() +"*",hakukentta);

    // Luodaan valintalistan alkiot.
    int j = nro;
    for (Jasen jasen : loytyneet) {
        // Laitetaan alkioksi jasen.
        listJasenet.addItem(jasen);
        // Laitetaan alkion otsikoksi jäsenen nimi.
        listJasenet.setItemCaption(jasen,
          jasen.getNimi());

        if (j == 0) {
            // Asetetaan listalle oletusarvo.
            listJasenet.setValue(jasen);
            jasenkohdalla = jasen;
            tulosta();
```

```
            }
            j = 1;
        }
        // Jos haussa ei löytynyt yhtään jäsentä,
        // niin asetetaan muokkausnappi pois
        // käytöstä.
        if (jasenkohdalla == null)
            buttonMuokkaa.setEnabled(false);
    }


    /**
     * Tulostaa jäsenen tiedot labeliin
     */
    private void tulosta() {
        if (jasenkohdalla == null) {
            panelJasen.setCaption("Jäsen: ");
            // Asetetaan labelille korkeus, että
            // tulostusalue jää näkyviin.
            labelJasentiedot.setHeight(400,UNITS_PIXELS);
            labelJasentiedot.setValue("");
            return;
        }
        // Asetetaan labelin koko määrittelemättömäksi,
        // jos sitä on muutettu.
        labelJasentiedot.setSizeUndefined();

        // Asetetaan panelille otsikko, johon laitetaan
        // valitun jäsenen nimi.
        panelJasen.setCaption("Jäsen: "
            + jasenkohdalla.getNimi());
        String tulostus = "";
        for (int i = jasenkohdalla.ekaKentta();
          i < jasenkohdalla.getKenttia(); i++) {
            tulostus = tulostus + "<p><b>"
            + jasenkohdalla.getKysymys(i) + ":</b> "
            + jasenkohdalla.anna(i) + "</p>";
```

```
    }
    // Laitetaan labeliin tulostettava teksti.
    labelJasentiedot.setValue(tulostus);
}


/**
 * @param jasen jäsen joka asetetaan kohdalla olevaksi
 */
public void setJasenkohdalla(Jasen jasen) {
    jasenkohdalla = jasen;
}


@AutoGenerated
private AbsoluteLayout buildMainLayout() {
    // common part: create layout
    mainLayout = new AbsoluteLayout();
    mainLayout.setStyleName("tausta");
    mainLayout.setImmediate(false);
    mainLayout.setWidth("100%");
    mainLayout.setHeight("800px");
    mainLayout.setMargin(false);

    // top-level component properties
    setWidth("100.0%");
    setHeight("800px");

    // kerhoAlue
    kerhoAlue = buildKerhoAlue();
    mainLayout.addComponent(kerhoAlue,
      "top:0.0px;left:0.0px;");

    return mainLayout;
}
```

```
@AutoGenerated
private VerticalLayout buildKerhoAlue() {
    // common part: create layout
    kerhoAlue = new VerticalLayout();
    kerhoAlue.setImmediate(false);
    kerhoAlue.setWidth("-1px");
    kerhoAlue.setHeight("-1px");
    kerhoAlue.setMargin(true);
    kerhoAlue.setSpacing(true);

    // otsikkoLayout
    otsikkoLayout = buildOtsikkoLayout();
    kerhoAlue.addComponent(otsikkoLayout);

    // hakuLayout
    hakuLayout = buildHakuLayout();
    kerhoAlue.addComponent(hakuLayout);

    // jasenetAlue
    jasenetAlue = buildJasenetAlue();
    kerhoAlue.addComponent(jasenetAlue);

    return kerhoAlue;
}


@AutoGenerated
private VerticalLayout buildOtsikkoLayout() {
    // common part: create layout
    otsikkoLayout = new VerticalLayout();
    otsikkoLayout.setImmediate(false);
    otsikkoLayout.setWidth("-1px");
    otsikkoLayout.setHeight("-1px");
    otsikkoLayout.setMargin(true);

    // labelOtsikko
    labelOtsikko = new Label();
```

```
        labelOtsikko.setImmediate(false);
        labelOtsikko.setWidth("-1px");
        labelOtsikko.setHeight("-1px");
        labelOtsikko.setValue("Otsikko");
        labelOtsikko.setContentMode(3);
        otsikkoLayout.addComponent(labelOtsikko);

        // labelJasenLkm
        labelJasenLkm = new Label();
        labelJasenLkm.setStyleName("lkmText");
        labelJasenLkm.setImmediate(false);
        labelJasenLkm.setWidth("-1px");
        labelJasenLkm.setHeight("-1px");
        labelJasenLkm.setValue("Jaseniä: ");
        otsikkoLayout.addComponent(labelJasenLkm);

        return otsikkoLayout;
    }


    @AutoGenerated
    private HorizontalLayout buildHakuLayout() {
        // common part: create layout
        hakuLayout = new HorizontalLayout();
        hakuLayout.setImmediate(false);
        hakuLayout.setWidth("-1px");
        hakuLayout.setHeight("-1px");
        hakuLayout.setMargin(true);
        hakuLayout.setSpacing(true);

        // labelHaku
        labelHaku = new Label();
        labelHaku.setStyleName("hakuText");
        labelHaku.setImmediate(false);
        labelHaku.setWidth("-1px");
        labelHaku.setHeight("-1px");
        labelHaku.setValue("Haettava jäsen : ");
```

```
        hakuLayout.addComponent(labelHaku);
        hakuLayout.setComponentAlignment(labelHaku,
          new Alignment(33));

        // editHaku
        editHaku = new TextField();
        editHaku.setStyleName("raja");
        editHaku.setImmediate(false);
        editHaku.setWidth("9.0em");
        editHaku.setHeight("-1px");
        editHaku.setSecret(false);
        hakuLayout.addComponent(editHaku);
        hakuLayout.setComponentAlignment(editHaku,
          new Alignment(33));

        // cbHaku
        cbHaku = new ComboBox();
        cbHaku.setImmediate(false);
        cbHaku.setWidth("10.0em");
        cbHaku.setHeight("-1px");
        hakuLayout.addComponent(cbHaku);
        hakuLayout.setComponentAlignment(cbHaku,
          new Alignment(33));

        // buttonHaku
        buttonHaku = new Button();
        buttonHaku.setCaption("Hae");
        buttonHaku.setImmediate(true);
        buttonHaku.setWidth("-1px");
        buttonHaku.setHeight("-1px");
        hakuLayout.addComponent(buttonHaku);
        hakuLayout.setComponentAlignment(buttonHaku,
          new Alignment(33));

        return hakuLayout;
    }
```

```
@AutoGenerated
private HorizontalLayout buildJasenetAlue() {
    // common part: create layout
    jasenetAlue = new HorizontalLayout();
    jasenetAlue.setStyleName("jasenetTausta");
    jasenetAlue.setImmediate(false);
    jasenetAlue.setWidth("800px");
    jasenetAlue.setHeight("-1px");
    jasenetAlue.setMargin(true);

    // jasenetValinta
    jasenetValinta = buildJasenetValinta();
    jasenetAlue.addComponent(jasenetValinta);

    // panelJasen
    panelJasen = buildPanelJasen();
    jasenetAlue.addComponent(panelJasen);

    return jasenetAlue;
}


@AutoGenerated
private VerticalLayout buildJasenetValinta() {
    // common part: create layout
    jasenetValinta = new VerticalLayout();
    jasenetValinta.setImmediate(false);
    jasenetValinta.setWidth("-1px");
    jasenetValinta.setHeight("-1px");
    jasenetValinta.setMargin(false);
    jasenetValinta.setSpacing(true);

    // listJasenet
    listJasenet = new ListSelect();
    listJasenet.setCaption("Jäsenet");
    listJasenet.setImmediate(false);
```

```
        listJasenet.setWidth("10.0em");
        listJasenet.setHeight("-1px");
        jasenetValinta.addComponent(listJasenet);

        // buttonMuokkaa
        buttonMuokkaa = new Button();
        buttonMuokkaa.setCaption("Muokkaa");
        buttonMuokkaa.setImmediate(true);
        buttonMuokkaa.setWidth("-1px");
        buttonMuokkaa.setHeight("-1px");
        jasenetValinta.addComponent(buttonMuokkaa);

        return jasenetValinta;
    }


    @AutoGenerated
    private Panel buildPanelJasen() {
        // common part: create layout
        panelJasen = new Panel();
        panelJasen.setCaption("Jäsen");
        panelJasen.setImmediate(false);
        panelJasen.setWidth("400px");
        panelJasen.setHeight("-1px");

        // jasenTiedot
        jasenTiedot = buildJasenTiedot();
        panelJasen.setContent(jasenTiedot);

        return panelJasen;
    }


    @AutoGenerated
    private VerticalLayout buildJasenTiedot() {
        // common part: create layout
        jasenTiedot = new VerticalLayout();
```

```
        jasenTiedot.setStyleName("raja");
        jasenTiedot.setImmediate(false);
        jasenTiedot.setWidth("395px");
        jasenTiedot.setHeight("100.0%");
        jasenTiedot.setMargin(true);
        jasenTiedot.setSpacing(true);

        // labelJasentiedot
        labelJasentiedot = new Label();
        labelJasentiedot.setImmediate(false);
        labelJasentiedot.setWidth("-1px");
        labelJasentiedot.setHeight("-1px");
        labelJasentiedot.setValue("Jäsen");
        labelJasentiedot.setContentMode(3);
        jasenTiedot.addComponent(labelJasentiedot);

        return jasenTiedot;
    }
}
```

C.3 Vaadinjasen.java

```
package com.example.vaadinkerho;

import jspkerho.KerhoBean;
import kerho.Jasen;
import kerho.Kerho;
import kerho.SailoException;
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.event.FieldEvents.TextChangeEvent;
import com.vaadin.event.FieldEvents.TextChangeListener;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.AbstractTextField.TextChangeEventMode;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;
```

```
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.Form;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.Panel;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Window;

/**
 * Näyttää valitun jäsenen tiedot
 * @author Joni Korkalainen
 * @version 13.06.2012
 */
public class Vaadinjasen extends CustomComponent {
    @AutoGenerated
    private AbsoluteLayout mainLayout;
    @AutoGenerated
    private VerticalLayout lomakeAlue;
    @AutoGenerated
    private HorizontalLayout painikeAlue;
    @AutoGenerated
    private Button buttonPeruuta;
    @AutoGenerated
    private Button buttonTalleta;
    @AutoGenerated
    private Label labelVirhe;
    @AutoGenerated
    private Panel lomakePanel;
    @AutoGenerated
    private VerticalLayout kenttaAlue;

    private TextField[] tekstiKentat = null;
    private Jasen apujasen;
    private Jasen muutettujasen = new Jasen();
    private Kerho kerho = KerhoBean.getKerho();
    private Form form;
```

```
private Window mywindow;
private Window mainwindow;
private Vaadinkerho vaadinkerho = new Vaadinkerho();

/**
 * The constructor should first build the main layout,
 * set the composition root and then do any custom
 * initialization.
 *
 * The constructor will not be automatically
 * regenerated by the visual editor.
 * @param main ikkuna jonka päälle uusi ikkuna laitetaan.
 */
public Vaadinjasen(Window main) {
    buildMainLayout();
    setCompositionRoot(mainLayout);

    // Luodaan uusi ikkuna.
    mywindow = new Window("Jäsenen muokkaus");
    mywindow.setPositionX(200);
    mywindow.setPositionY(0);
    mywindow.setHeight("800px");
    mywindow.setWidth("400px");

    // Asetetaan ikkuna modaaliseksi.
    // Näin käyttäjän täytyy sulkea ikkuna
    // ennen kuin voi jatkaa pääikkunassa.
    mywindow.setModal(true);

    main.addWindow(mywindow);
}


/**
 * @param main ikkuna jonka päälle uusi ikkuna laitetaan
 * @param jasen jäsen jonka tiedot näytetään
 * @return avattu ikkuna
```

```
 */
public static Window addWindow(Window main, Jasen jasen)
{
    Vaadinjasen vaadinjasen = new Vaadinjasen(main);
    vaadinjasen.mainwindow = main;

    vaadinjasen.luoLomake(jasen);

    // Asetetaan ikkunan sisältö
    vaadinjasen.mywindow.setContent(vaadinjasen);
    // Asetetaan sisältö näkyväksi
    vaadinjasen.setVisible(true);

    return vaadinjasen.mywindow;
}


/**
 * Luo lomakkeen ja asettaa painikkeille kuuntelijat
 * @param jasen jäsen jonka tiedot näytetään
 */
public void luoLomake(Jasen jasen) {
    apujasen = jasen;

    // Luodaan uusi lomake.
    form = new Form();
    // Asetetaan lomakkeen otsikoksi valitun
    // jäsenen nimi.
    form.setCaption(jasen.getNimi());
    // Asetetaan lomakkeen kentät.
    asetaKentat(jasen);
    // Lisätään lomake paneliin.
    lomakePanel.addComponent(form);

    // Asetetaan napeille kuuntelijat.
    buttonTalleta.addListener(new ClickListener() {
        @Override
```

```
            public void buttonClick(ClickEvent event) {
                // Jos lomakkeen tietoja ei ole syötetty
                // oikein.
                if (!tarkista()) return;
                // Asetetaan jäseneen muutetut tiedot
                // ja talletetaan.
                asetaMuutokset();
                talleta(apujasen);
                // Suljetaan ikkuna.
                mainwindow.removeWindow(mywindow);
            }
        });

        buttonPeruuta.addListener(new ClickListener() {
            @Override
            public void buttonClick(ClickEvent event) {
                mainwindow.removeWindow(mywindow);
            }
        });

    }


    /**
     * Asettaa lomakkeen kentät ja niihin valitun jäsenen
     * tiedot.
     * @param jasen jäsen jonka tiedot näytetään
     */
    public void asetaKentat(Jasen jasen) {
        labelVirhe.setVisible(false);
        // Poistetaan lomakkeelta kaikki kentät.
        form.removeAllProperties();
        int n =  jasen.getKenttia() - jasen.ekaKentta();
        tekstiKentat = new TextField[n];

        // Luodaan lomakkeen kentät.
        for (int i = 0, k = jasen.ekaKentta();
```

```
        k < jasen.getKenttia(); i++, k++) {
          final int kentta = k;
          String oletus = jasen.anna(k);
          String kysymys = jasen.getKysymys(k);
          muutettujasen.aseta(k, oletus);

          // Luodaan uusi tekstikenttä.
          TextField tfJasen = new TextField(kysymys);
          // Asetetaan kentälle oletusarvo.
          tfJasen.setValue(oletus);
          // Asetetaan muutokset kentässä välittymään
          // palvelimelle heti kun tekstiä on syötetty.
          tfJasen.setTextChangeEventMode(
            TextChangeEventMode.EAGER);
          // Lisätään kenttä lomakkeelle.
          form.addField(kysymys, tfJasen);
          tekstiKentat[i] = tfJasen;
          // Asetetaan kentälle kuuntelija, joka tarkistaa
          // onko syötetty oikeaa tietoa aina kun teksti
          // muuttuu.
          tfJasen.addListener(new TextChangeListener() {
              @Override
              public void textChange(TextChangeEvent event)
              {
                  // Otetaan kenttään syötetty teksti ja
                  // tarkistetaan sen oikeellisuus.
                  String syotto = event.getText();
                  tarkistaKentta(
                    (TextField)event.getComponent(),
                      syotto, kentta);
              }
          });

    }
}
```

```
/**
 * Tarkistaa onko tekstikenttään syötetty
 * oikeanlainen arvo.
 * @param text tekstikentta joka tarkistetaan
 * @param syotto kenttään syötetty teksti
 * @param kentta mikä jäsenen kenttä tarkistetaan
 */
public void tarkistaKentta(TextField text, String syotto,
  int kentta) {
    String virhe = muutettujasen.aseta(kentta, syotto);
    if (virhe == null) {
        // Muutetaan kentän tyylinimi, jotta kenttä
        // tulee normaalinväriseksi.
        text.setStyleName("normaaliEdit");
        // Asetetaan labelVirhe tyhjäksi.
        labelVirhe.setValue("");
        // Asetetaan labelVirhe pois näkyvistä.
        labelVirhe.setVisible(false);
        // Asetetaan talletusnappi toimivaksi.
        buttonTalleta.setEnabled(true);
        return;
    }
    // Asetetaan kentän tyylinimi, jotta saadaan
    // sille virheväri.
    text.setStyleName("virheEdit");
    // Asetetaan labelVirhe näkyviin.
    labelVirhe.setVisible(true);
    // Asetetaan labelVirheeseen virheteksti.
    labelVirhe.setValue(" " + virhe);
    // Asetetaan talletusnappi pois käytöstä.
    buttonTalleta.setEnabled(false);
}


/**
 * Tarkistaa onko kaikki kentät täytetty oikein
 * @return true jos kaikki kentät on syötetty oikein
```

```
 */
public boolean tarkista() {
    boolean result = true;
    for (TextField text : tekstiKentat) {
        // Jos tekstikentän tyylinimi on "virheEdit",
        // niin lomakkeella on virheitä.
        if (text.getStyleName().equals("virheEdit"))
          result = false;
    }
    return result;
}


/**
 * Asettaa jäseneen muutetut tiedot
 */
public void asetaMuutokset() {
    for (int k = apujasen.ekaKentta();
      k < apujasen.getKenttia(); k++) {
        apujasen.aseta(k, muutettujasen.anna(k));
    }
}


/**
 * @param jasen jäsen joka talletetaan
 */
public void talleta(Jasen jasen) {
    try {
        kerho.korvaaTaiLisaa(jasen);
        kerho.talleta();
    } catch (SailoException e) {
            e.printStackTrace();
    }
    // Asetetaan kohdalla olevaksi jäseneksi
    // talletettu jäsen.
    vaadinkerho.setJasenkohdalla(jasen);
```

```
}


@AutoGenerated
private AbsoluteLayout buildMainLayout() {
    // common part: create layout
    mainLayout = new AbsoluteLayout();
    mainLayout.setStyleName("jasenTausta");
    mainLayout.setImmediate(false);
    mainLayout.setWidth("100%");
    mainLayout.setHeight("600px");
    mainLayout.setMargin(true);

    // top-level component properties
    setWidth("100.0%");
    setHeight("600px");

    // lomakeAlue
    lomakeAlue = buildLomakeAlue();
    mainLayout.addComponent(lomakeAlue,
      "top:0.0px;right:-3.0px;bottom:0.0px;left:0.0px;");
    return mainLayout;
}


@AutoGenerated
private VerticalLayout buildLomakeAlue() {
    // common part: create layout
    lomakeAlue = new VerticalLayout();
    lomakeAlue.setStyleName("jasenTausta");
    lomakeAlue.setImmediate(false);
    lomakeAlue.setWidth("100.0%");
    lomakeAlue.setHeight("100.0%");
    lomakeAlue.setMargin(true);

    // lomakePanel
    lomakePanel = buildLomakePanel();
```

```
        lomakeAlue.addComponent(lomakePanel);

        // labelVirhe
        labelVirhe = new Label();
        labelVirhe.setStyleName("virheLabel");
        labelVirhe.setImmediate(false);
        labelVirhe.setWidth("100.0%");
        labelVirhe.setHeight("-1px");
        labelVirhe.setValue("Label");
        lomakeAlue.addComponent(labelVirhe);

        // painikeAlue
        painikeAlue = buildPainikeAlue();
        lomakeAlue.addComponent(painikeAlue);

        return lomakeAlue;
    }


    @AutoGenerated
    private Panel buildLomakePanel() {
        // common part: create layout
        lomakePanel = new Panel();
        lomakePanel.setImmediate(false);
        lomakePanel.setWidth("100.0%");
        lomakePanel.setHeight("-1px");

        // kenttaAlue
        kenttaAlue = new VerticalLayout();
        kenttaAlue.setImmediate(false);
        kenttaAlue.setWidth("100.0%");
        kenttaAlue.setHeight("100.0%");
        kenttaAlue.setMargin(true);
        lomakePanel.setContent(kenttaAlue);

        return lomakePanel;
    }
```

```
    @AutoGenerated
    private HorizontalLayout buildPainikeAlue() {
        // common part: create layout
        painikeAlue = new HorizontalLayout();
        painikeAlue.setImmediate(false);
        painikeAlue.setWidth("-1px");
        painikeAlue.setHeight("-1px");
        painikeAlue.setMargin(false);
        painikeAlue.setSpacing(true);

        // buttonTalleta
        buttonTalleta = new Button();
        buttonTalleta.setCaption("Talleta");
        buttonTalleta.setImmediate(true);
        buttonTalleta.setWidth("-1px");
        buttonTalleta.setHeight("-1px");
        painikeAlue.addComponent(buttonTalleta);

        // buttonPeruuta
        buttonPeruuta = new Button();
        buttonPeruuta.setCaption("Peruuta");
        buttonPeruuta.setImmediate(true);
        buttonPeruuta.setWidth("-1px");
        buttonPeruuta.setHeight("-1px");
        painikeAlue.addComponent(buttonPeruuta);

        return painikeAlue;
    }
}
```

C.4 styles.css

```
@import url(../reindeer/styles.css);
```

```
.tausta {
    background: #C0C0C0;
}

.jasenetTausta {
    background: #D8D8D8;
    border: 1px solid #888888;
}

.jasenTausta {
    background: #D8D8D8;
}

.raja {
    border: 1px solid #888888;
}

.lkmText {
    font-size: 1.5em;
}

.hakuText {
    font-size: 1.3em;
}

.virheLabel {
    background: red;
    text-align: center;
    font-weight: bold;
}

.virheEdit {
    background: red;
}

.normaaliEdit {
    background: white;
```

```
}
```