Matias Hilden

Tiedonsiirron web-rajapinnan toteuttaminen Loihdin-teknologiaan

Metropolia Ammattikorkeakoulu
Insinööritutkinto
Tietotekniikka
Opinnäytetyö
27.11.2012

Metropolia

Abstract

Tekijä Otsikko Sivumäärä Aika	Matias Hilden Tiedonsiirron web-rajapinnan toteuttaminen Loihdin-teknologiaan 36 sivua 27.11.2012
Tutkinto	Insinöörityötutkinto
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	yliopettaja Auvo Häkkinen toimitusjohtaja Toni Halsti

Insinöörityössä toteuttiin tiedonsiirron rajapinta Loihdin-teknologiaan. Työ aloitettiin selvittämällä perusseikkoja Loihdin-teknologiasta. Loihdin-teknologian avulla voidaan tarjota pilvipalveluja, joiden tietosisältöä ja käyttöliittymiä voidaan hallita asetustiedostoilla. Tämän jälkeen kerrottiin myös aikaisemmista rajapinnoista ja niiden heikkouksista.

Seuraavaksi tutkittiin web-rajapintoihin liittyviä teknologioita. Loihdin-teknologia saneli paljon mitä teknologioita rajapinnassa käytettiin. Java Servletin käyttö oli välttämätöntä, jotta rajapinta voitiin helposti liittää osaksi Loihdin-teknologiaa. Tietokannassa olevan tiedon muokkaukseen käytettiin Loihdin-kirjastoja, joten siihen liittyviä teknologioita ei tarvinnut edes tutkia. Myös rajapintoihin liittyviä tietoturvariskejä tutkittiin, jolloin pääpaino oli sql-injektioiden tutkimisessa.

Tämän jälkeen käytiin läpi rajapinnan määrittely ja suunniteltiin rajapinnan toiminnot tutkimustulosten avulla. Rajapinnalla piti pystyä kirjautumaan, sekä hakemaan, lisäämään, muokkaamaan ja poistaamaan tietoa. Lisäksi osaan toiminnoista piti rakentaa Loihdin-luokat, joilla rajapinnan käyttö Loihdin-järjestelmissä on helppoa.

Lopuksi käytiin läpi itse rajapinnan toteutus. Pääsääntöisesti rajapinta koostui servletistä, autentikointiluokasta ja wrapper-luokasta. Kullekin rajapintatoiminnolle tehtiin omat luokkansa, joita wrapper-luokka kutsui. Näiden lisäksi rajapinta sisälsi kaksi apuluokkaa, jotka sisälsivät Loihdin- tiedonkäsittelyyn liittyviä apumetodeja. Lisäksi erillisenä rajapinnasta toteutettiin luokat, joilla Loihdin-järjestelmät pystyvät kutsumaan toistensa rajapintoja.

Rajapinnan toteutuksen aikana ongelmat liittyivät enimmiltä osin rajapinnan käyttäjälle palautettaviin viesteihin virhetilanteiden tapahtuessa ja helposti ymmärrettävien rajapintakuvausten tekemiseen. Rajapinta on tällä hetkellä käytössä yhdellä yhteityökumppanilla ja useissa yrityksen Loihdin-järjestelmissä.

Avainsanat	Web-rajapinta, Loihdin, dynaaminen tietorakenne

Metropolia

Abstract

Author Title Number of Pages Date	Matias Hilden Web interface for data transfer in Loihdin technology 36 pages 27.11.2012
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialization	Software Engineering
Instructors	Auvo Häkkinen, Principal lecturer Toni Halsti, CEO

The aim of the thesis was to create a web interface for data transfer to Loihdin technology. The thesis starts off with introducing the basics of Loihdin technology. With Loihdin one can provide cloud services and customize their data structure and user interfaces. After this, information is given about the old web interfaces and their weaknesses.

Next the thesis studies technologies associated with web interfaces. Building the web interface as a part of Loihdin dictated a lot on what technologies to use. The use of Java Servlet was necessary so that the interface could be easily attached to Loihdin. Loihdin libraries were used for writing and reading data so it was not necessary to study technologies associated with that. Security risks associated with web interfaces were also studied with the main focus on sql injections.

Based on the study results, the thesis defines and plans the web interface. Use cases are authentication, data search, add, edit and delete. In addition to this, Loihdin classes had to be built for some of these use cases to ease their use in Loihdin systems.

Finally the thesis goes through constructing the web interface. The interface has three major parts: Servlet, authentication and a wrapper class. Each use case has its own implementing class which the wrapper class calls. The interface also has two helper classes that have methods for handling data in Loihdin. Loihdin classes for calling the interface are separated from the other classes.

Problems that rose during the development of the interface were mostly related to responses in error cases and making the documentation easy to understand. At the moment the interface is used by one business partner and several Loihdin systems in our company, Adensy Oy.

Keywords	Web interface, Loihdin, dynamic data structure

Sisältö

1 Johdanto

Sovelluksille tarkoitetut web-rajapinnat ovat nykypäivänä oleellinen osa pilvipalveluja. Yrityksillä on harvoin käytössään vain yksi työkalu, joka hoitaisi kaikki tietotekniset tarpeet, jolloin ne haluavat työkalujensa toimivan automaattisesti yhdessä. Pilvipalvelut tarjoavat sovelluskehittäjille web-rajapinnan, jotta tämä automatiikka saadaan toimimaan.

Loihdin on Adensy Oy:n kehittämä teknologia, jolla voidaan tarjota pilvipalveluja pitkälti automatisoidusti. Adensy Oy on 2009 tammikuussa perustettu ohjelmistoalan yritys. Yrityksen asiakasmäärän ja pilvipalvelujen määrän kasvaessa on myös kysyntä web-rajapinnalle kasvanut. Tämän myötä yhtenäisen rajapinnan toteuttaminen Loihdin-järjestelmille tuli ajankohtaiseksi.

Tässä raportissa kerrotaan aluksi luvussa 2 Loihdin-teknologian toimintaperiaatteesta ja selvitetään Loihdin-teknologiaan liittyvää termistöä. Tämän jälkeen kerrotaan, miten aikaisemmat rajapintatarpeet on toteutettu ja miksi näiden tilalle tarvitaan uusi rajapinta. Luvussa 3 tutkitaan rajapinnan toteuttamiseen tarvittavat teknologiat ja perustellaan, miksi juuri ne valittiin rajapinnan toteuttamiseen. Tarkastellaan myös web-rajapintojen tietoturvaan liittyviä ongelmia ja niiden ratkaisuja.

Luvussa 4 käydään läpi rajapinnan toteutusta alkaen rajapinnan yleisestä määrittelystä. Tämän jälkeen suunnitellaan kunkin rajapintatoiminnon käyttö, jonka jälkeen siirrytään ohjelmiston rakenteeseen ja luokkakohtaisiin toteutuksiin. Lopuksi kerrotaan, miten rajapinnan toiminta todettiin ja miten rajapinta otettiin tuotantokäyttöön.

2 Loihdin

Tässä luvussa käydään läpi Loihdin-teknologian historiaa, mitä varten Loihdin on rakennettu ja mitä sillä on tarkoitus tehdä. Tämän jälkeen käydään läpi teknologian perustietoja, eritoten työn ymmärtämiselle oleelliset tietoluokat, joiden ympärille Loihtimen toiminta perustuu. Lopuksi käydään läpi Loihtimen tarjoamat työkalut sovelluskehityksessä, joista tarkastellaan eritoten web-rajapintaan liittyviä asioita.

2.1 Perustiedot Loihtimesta ja malliperusteisesta ohjelmistokehityksestä

Loihdin on malliperusteinen sovelluskehityksen työkalu. Loihdin mahdollistaa monimutkaisten yrityssovellusten kehittämisen ilman tarvetta perinteiseen ohjelmointiin. Loihtimen avulla voidaan luoda ilman teknistä osaamista yrityssovelluksia interaktiivisten mallien avulla. [1.] Tämä tarkoittaa sitä, että käyttöliittymät ja niihin liittyvät prosessit, kuten esimerkiksi muokatun tiedon tallentaminen, voidaan kehittää erilaisten editorien avulla. Esimerkiksi yrityksen tiedon muokkauksessa Loihtimen käyttäjä voisi valita, mitä kenttiä halutaan muokata ja mihin näkymään sovellus siirtyy eri toimenpiteiden (tallenna, peruuta) jälkeen. Nämä valinnat kirjautuvat sovelluksen asetustiedostoihin, joiden perustella Loihdin automaattisesti rakentaa halutun käyttöliittymän ja toiminnallisuuden.

Loihtimen tarkoitus on säästää huomattavasti ohjelmistokehitykseen kuluvia resursseja sekä pienentää mahdollisten virheiden määrää automatisoimalla suurimman osan järjestelmästä niin, että käyttöliittymä ja järjestelmän data ovat edelleen dynaamisina. Yritykset voivat myös räätälöidä Loihdin-järjestelmiä uusiin tarpeisiin sopivaksi olematta riippuvaisia palvelun tuottajasta. Loihdin-työkalu on saanut kehuja akateemisilta tahoilta, kuten muun muassa VTT:lta ja Helsingin yliopistolta sekä organisaatioilta Karl Fazer ja HUS. [1.]

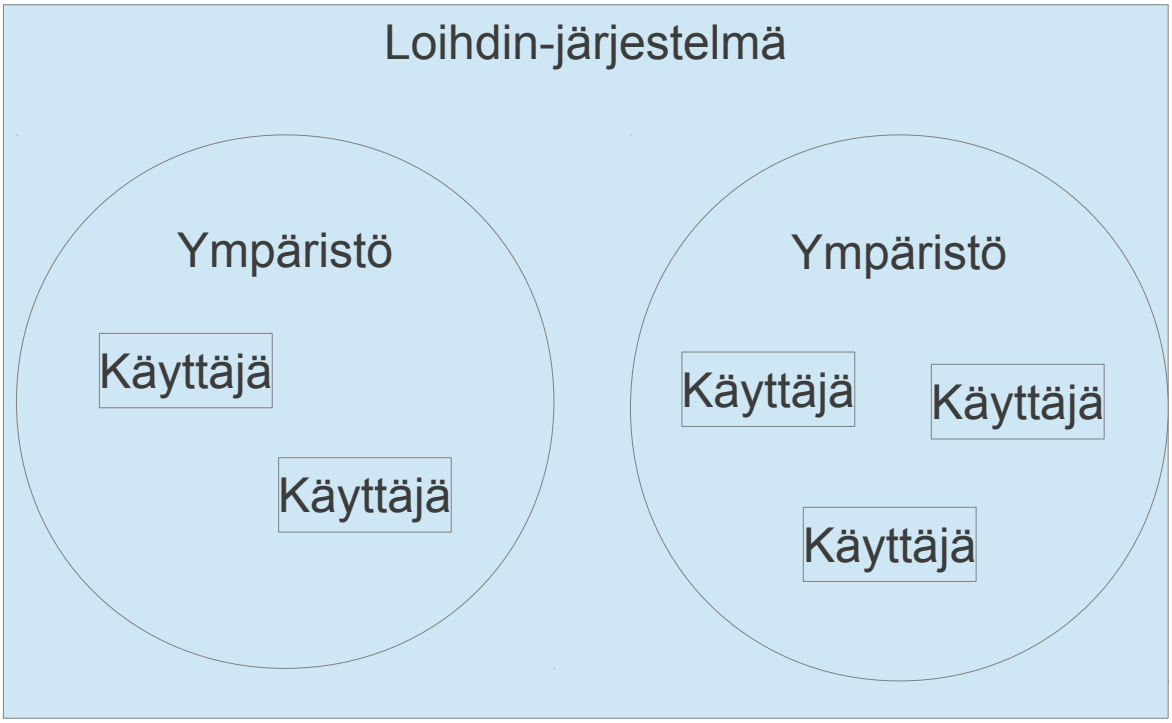
Loihtimen malliperusteisen ohjelmistokehityksen lähestymistapa eroaa tavanomaisesta ohjelmistokehityksestä myös ohjelmistoon liittyvällä laadunvarmistuksella. Tavanomaisessa ohjelmistokehitysmallissa laadunvarmistus tapahtuu yleensä ennen tai jälkeen ohjelmiston kehitystä käyttäen vaatimusanalyysiä (ennen) ja ohjelmiston testausta (jälkeen), jolloin näiden kahden välinen linkki on vain abstraktilla tasolla. Malliperusteisessa ohjelmistokehityksessä laadunvarmistus voidaan tehdä ohjelmiston kehityksen aikana. Mallin muuntuessa ohjelmistokoodiksi vältetään teknisiä virheitä, ja mallin validoinnilla vältetään kehittäjän tekemiä loogisia virheitä. [1.]

Prosessien ja käyttöliittymien muokkaus editoreilla ovat kuitenkin lähinnä Loihtimen tavoitteita, sillä teknologia ei ole vielä täysin valmis. Tällä hetkellä toimivat editorit puuttuvat, jolloin asetustiedostoja pitää muokata käsin. Tämän takia Loihdin-järjestelmien kehittäminen ei onnistu ilman Loihdin-teknologian laajaa tuntemusta. Teknologian perusaatteet ovat kuitenkin pohjalla, joten suuri osa järjestelmien toiminnasta ja käyttöliittymistä on automatisoituja. Tämä myös eliminoi kooditason virheitä, ja suurin osa virheistä liittyy asetustiedostojen oikeinkirjoitukseen, kuten yrityksen sisäisessä käytössä on huomattu.

Tällä hetkellä Loihdinta on käytetty lähinnä tuottamaan yrityksen omia järjestelmiä, eikä sitä ole tuotteena myyty asiakkaille. Tuotettuihin järjestelmiin kuuluu muun muassa Adensy CRM, joka sisältää asiakkuudenhallinnan työkalujen lisäksi työaikaseurantaa, varastohallintaa, laskutustyökalun ja muita yritystoiminnalle hyödyllisiä toimintoja. Tämän lisäksi Loihtimella on tuotettu tapahtumatietojen hallintatyökalu, jonka avulla tapahtumajärjestäjät voivat tarjota osallistujilleen järjestelmän omien esitetietojen muokkaamiseen ja muiden osallistujien seuraamiseen. Loihdin-teknologialla on myös tehty käyttäjäseurannan työkalu, joka seuraa Adensy CRM käyttöä ja on Adensyn sisäiseen käyttöön tarkoitettu.

2.1.1 Ympäristöt ja käyttäjät

Loihdin-järjestelmät koostuvat ympäristöistä ja käyttäjistä. Käyttäjällä tarkoitetaan henkilöä, joka käyttää järjestelmää. Tämä käyttäjä kuuluu aina johonkin ympäristöön, johon yleensä kuuluu myös muita käyttäjiä. Samaan ympäristöön kuuluvat käyttäjät näkevät järjestelmässä saman tiedon, ellei käyttäjäkohtaisilla oikeuksilla sitä ole jotenkin eroteltu. Esimerkkinä tästä voidaan käyttää Adensy CRM -palvelua, jossa ympäristö kuuluu yhdelle asiakasyritykselle ja tämän yrityksen työntekijät puolestaan ovat ympäristön käyttäjiä. Ympäristöjen ja käyttäjien suhdetta on kuvattu kuvassa 1.



Kuva 1: Ympäristöt ja käyttäjät

Loihtimessa voidaan määritellä ympäristökohtaisesti, mitä ominaisuuksia asiakkaalla on käytössä. Loihdin-prosesseissa voidaan ympäristökohtaisesti kysyä, onko ominaisuus päällä. Enimmäkseen ominaisuuden tarkistusta käytetään, kun rajataan pääsyä sovelluksen eri osiin, kuten vaikkapa laskutukseen tai varastonhallintaan. Tämän lisäksi kullekin käyttäjälle voidaan käyttäjäkohtaisesti määritellä käyttöoikeudet, joilla voidaan rajata käyttäjälle nä-

kyvää tietoa ja pääsyä eri sovelluksen osiin. Nämäkin käyttöoikeudet on tehty mahdollisimman dynaamisiksi, joten lähes millaiset vain voi rajata.

2.1.2 Tietoluokat ja niiden kentät

Loihdin-järjestelmän sisältämä tieto koostuu tietoluokista ja niiden kentistä. Nämä voidaan rinnastaa tietokannan tauluihin ja niiden sarakkeisiin. Esimerkiksi yrityksen tiedot voisivat löytyä company-tietoluokasta, jolla olisi kentistä yrityksen nimi, yhteystiedot, työntekijät ja yritykseen liittyvät sopimukset. Näistä työntekijät ja sopimukset olisivat muiden tietoluokkien jäseniä. Tietoluokkien ilmentymät sisältävät myös yksilöllisen rowid-tunnisteen ja tiedon siitä, mihin asiakasympäristöön ne kuuluvat.

Tietoluokat ovat dynaamisia sisältämänsä tiedon suhteen, joten niiden avulla voidaan käsitellä minkälaisia tietokokonaisuuksia tahansa. Tietoluokat määritellään kuvaustiedostoissa, joiden perusteella Loihdin muokkaa tietokannan sopivaksi tietoluokkien datalle. Tämän toiminnallisuuden ansiosta tietojärjestelmän dataa on mahdollista käsitellä ohjelmallisesti eräänlaisen Bean-olion kautta. Myös muutokset tietojärjestelmään onnistuvat tämän ansiosta yrityksen kokemuksen mukaan nopeasti ja vaivattomasti.

2.2 Loihdin ohjelmointikehyksenä

Tietoluokkien ansiosta Loihtimen päälle kehitettävät sovellukset voidaan pitää erillään käsiteltävästä tiedosta. Tämän ansiosta on helppoa toteuttaa sovellus, jonka käsittelemä tieto määritellään suoraan ohjelmalle tulevasta syötteestä. Samalla Loihdin tarjoaa myös oliomallin tietokannan käsittelyyn, jolloin tiedon käsittely helpottuu.

Loihtimen ansiosta ohjelmointiprosessissa säästetään aikaa, kun tietokan-

nan käsittelyä ei tarvitse itse toteuttaa. Samasta syystä voidaan myös luottaa siihen, ettei tietokannan käsittelyssä tule virheitä. Se vähentää koko ohjelman virhealttiutta. Loihdin tarjoaa omat työkalunsa myös käyttöliittymien rakentamiseen.

Kääntöpuolena Loihdin, kuten kaikki muutkin ohjelmistokehykset, haittaa ohjelman suorituskykyä. Tämä ei kuitenkaan yleensä muodostu ongelmaksi, sillä ohjelman prosesseja tehdään lähinnä käyttäjän pyynnöstä, eivätkä ne sisällä raskasta laskentaa.

2.3 Tarve tiedonsiirron rajapintaan

Loihdin-järjestelmiin voidaan toteuttaa rajapintakutsujen kaltaisia pyyntöjä. Kun tarve rajapinnalle on ilmentynyt, se on toteutettu käyttämällä Loihtimen omaa prosessikäytäntöä, jossa xml-kuvaustiedostojen perusteella muodostetaan suoritettavaksi haluttava prosessi. Tämä tekniikka ei kuitenkaan sovi rajapinnoille.

Loihdin-järjestelmissä on määritelty joukko http-osoitteita, joilla järjestelmään pääsee. Vanhat rajapinnat on lisätty tähän joukkoon. Ongelmana on kuitenkin se, että osoitteet on tarkoitettu selainkäyttöön, ja niitä kutsumalla järjestelmä palauttaa aina javascriptillä rakennetun käyttöliittymän. Tämän takia vanhoissa rajapinnoissa rajapinnan käyttäjä ei koskaan saanut viestiä operaation onnistumisesta, vaan hänelle palautettiin automaattisesti generoitu käyttöliittymä. Samasta syystä tiedon luku vanhojen rajapintojen kautta ei ollut mahdollista.

Loihtimen prosessikäytäntö johti myös toiseen ongelmaan, rajapinnan hitauteen. Vanha rajapinta generoi turhaan käyttöliittymän ja suoritti muitakin käyttöliittymään liittyviä asioita, vaikkei tähän ollut lainkaan tarvetta. Generoitu käyttöliittymä myös tallentui sovelluksen muistiin istunnon päättymi-

seen saakka (45 minuuttia).

Kolmanneksi jokaiselle rajapintatarpeelle piti rakentaa erikseen toteutus Loihtimen asetustiedostoilla. Tämä oli työlästä myös siksi, ettei asetustiedostojen muokkaukseen ollut editoria, vaan xml-tiedostoja piti muokata käsin. Loihdin siis tarvitsee rajapinnan, jossa ei ole näitä ongelmia.

3 Web-rajapinnat

Tässä luvussa käsitellään web-rajapintoihin liittyviä teknologioita ja tietoturvaongelmia. Tarkoituksena on löytää sopivat työkalut web-rajapinnan toteuttamiseen ja välttää web-rajapintoihin liittyvät tietoturvaongelmat.

3.1 Teknologiat

Web-rajapintojen teknologiat voidaan jakaa karkeasti kahteen joukkoon: siirrettävän tiedon teknologioihin ja palvelimen päässä sijaitseviin teknologioihin. Ensimmäisiin liittyen tutkitaan laajalle levinnyttä xml-formaattia (Extended Markup Language) ja vähemmän tunnettua JSON-formaattia (JavaScript Object Notation). Palvelinpään teknologioista tutkitaan Javan HttpServlet, jota muukin Loihdin-teknologia käyttää käyttöliittymän välittämisessä selaimelle.

3.1.1 XML

Xml:n levinneisyyden vuoksi valtaosa pilvipalveluiden web-rajapinnoista tukee sitä. Siksi onkin järkevää, että tämänkin projektin tuottama rajapinta tukee xml:ää.

Xml-aineiston käyttämisestä on useita hyötyjä. Ensinnäkin xml-aineisto on ihmiselle helposti luettavaa, joka auttaa rajapintaa vasten kehittämisessä. Xml-aineiston käyttämiseen löytyy myös erittäin laaja valikoima työkaluja, joiden avulla aineistojen tuottaminen ja tulkitseminen on tehokasta ja luotettavaa. Xml-datan siirtäminen http:n yli ei myöskään tuota ongelmia. Edellä mainittujen hyötyjen lisäksi tässä projektissa tärkein etu, xml-tiedon tyypittömyys, mahdollistaa dynaamisen ja automaattisesti laajentuvan rajapinnan muodostamisen. [2.]

Xml:ssä on myös haittapuolensa, jotka liittyvät xml-aineiston suureen kokoon. Objekteja kuvattaessa lopetustagit ovat usein redundantteja. Seuraavassa kappaleessa tarkastelemmekin yhtä teknologiaa, joka korjaa tämän ongelman.

3.1.2 JSON

JSON on toinen formaatti, jota käytetään tiedonsiirrossa Xml-formaatin lisäksi. Vaikka töissä kohdattujen muiden järjestelmien rajapinnat ovat tukeneet lähinnä Xml-formaattia, on joukosta löytynyt myös JSON:ää tukevia rajapintoja. Mobiilisovellusten lisääntyessä JSON:n käyttö on myös lisääntynyt, sillä mobiililaitteiden hitailla yhteyksillä Xml:ää tiiviimpi formaatti on tarpeellinen. Vaikkei Loihtimen rajapintaan ensimmäisessä vaiheessa rakenneta tukea JSON:lle, on se silti hyvä tutkia, mikäli tarvetta laajennukselle tulee.

JSON eli JavaScript Object Notation on vuonna 2006 standardoitu dataformaatti. Se noudattaa sisältämissään muuttujissa paljolti JavaScriptin ja Php:n syntaksia, muttei ole ohjelmointikieleen sidonnainen. JSON-aineiston luontiin ja tulkitsemiseen löytyvät työkalut useimmista ohjelmointikielistä. [3.]

Seuraavassa koodiesimerkissä vertaillaan JSON- ja Xml-aineiston eroja:

```
{
      "type": "company",
      "name": "Firma Oy",
      "priority": 1,
      "isactive": true,
      "contact": {
            "name": "Matti Meikäläinen"
            "email": "mm@firma.fi"
            "phonenumber": "040 123 4567"
      }
}

<company>
      <name>Firma Oy</name>
      <priority>1</priority>
      <isactive>true</isactive>
      <contact>
            <name>Matti Meikäläinen</name>
            <email>mm@firma.fi</email>
            <phonenumber>040 123 4567</phonenumber>
      </contact>
</company>
```

Koodiesimerkki 1: JSON:n ja XML:n vertailu

Ensimmäinen huomattava seikka on edellisessä kappaleessa mainittu syntaksi muuttujissa. Merkkijonot määritellään lainausmerkkejä käyttäen. Numerot ja boolean-muuttujat taas merkitään ilman lainausmerkkejä. JSON:ssä ei myöskään käytetä lopetustageja muuttujien perässä, joka pienentää siirrettävän aineiston kokoa huomattavasti, varsinkin suurien aineistojen tapauksessa. Haittapuolena JSON ei sisällä attribuutteja. Tosin tämä voidaan kiertää järkevällä objektien käytöllä.

JSON onkin hyvä formaatti tämänlaisen tiedon siirtoon. Kyselyidenkin teko onnistuu sitä käyttämällä, vaikkakin xml-elementtien attribuuttien korvaaminen saattaa muodostua ongelmaksi.

3.1.3 Java HttpServlet

Java Servletit ovat luokkia, jotka on tarkoitettu web-palvelimen käytettäväk-

si. Kaikki servletit periytyvät luokasta GenericServlet. Tässä työssä keskitytään sen aliluokkaan HttpServlet, joka sisältää lisätoimintoja http-kutsujen käsittelyyn.

Toisin kuin tavalliset Java-ohjelmat, servletit eivät sisällä main()-metodia. Tämän sijaan servletille toteutetaan ennalta nimetyt metodit, joita web-palvelin kutsuu servletin service()-metodin kautta. Kutsu saa parametrinaan request- ja response-oliot. Request sisältää http-kutsun tiedot ja responsea käytetään vastauksen palauttamiseen.

HttpSerlvetiä käytettäessä ylikirjoitetaan yleensä doGet()- ja doPost()-metodit, joita service()-metodi kutsuu riippuen siitä, onko http-kutsun metodiksi määritelty GET vai POST. HttpServletissä voidaan myös ylikirjoittaa metodit doPut(), doDelete(), doTrace() ja doOption(), vaikkakaan näitä metodeja ei yleensä käytetä [4.]. Tässä työssä tarvitaan vain doGet()- ja doPost()-metodeja.

HttpSerlvet-, HttpRequest- ja HttpResponce-luokkien lisäksi Javan servlet-paketti sisältää paljon apuluokkia. Näistä mainittavia ovat HttpSession ja Cookie -luokat, joiden avulla voidaan hallinnoida istuntotietoja sekä evästeitä. Tässä työssä näitä ei kuitenkaan tulla tarvitsemaan.

3.2 Tietoturva

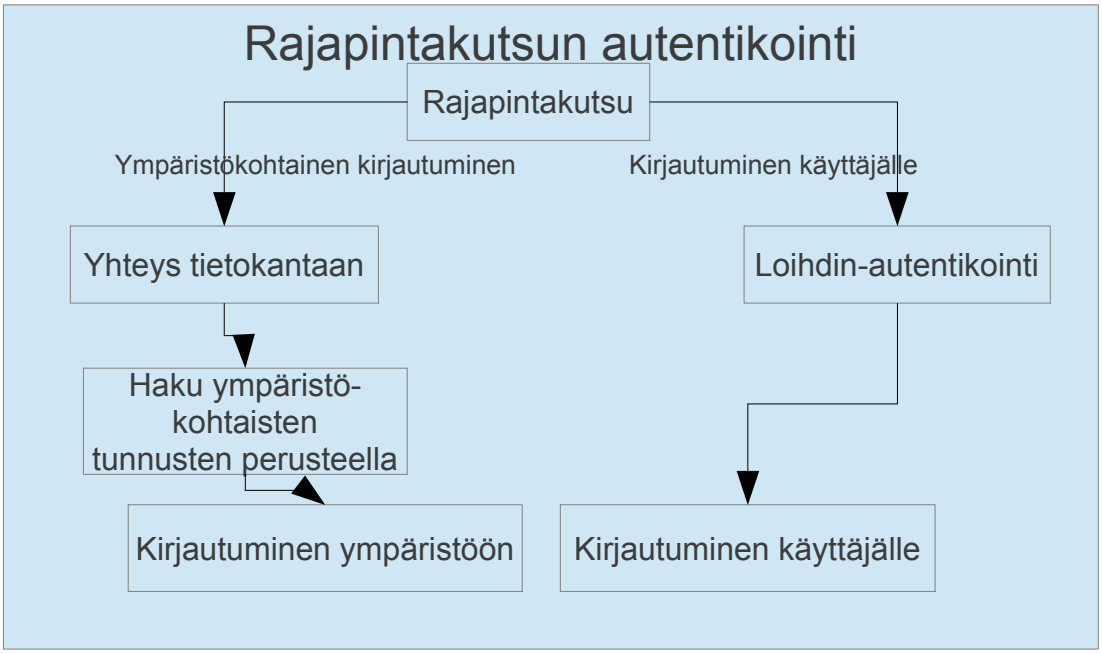
Web-rajapintaa suunnitellessa on päätettävä, millä tavoin rajapintakutsut autentikoidaan ja miten kutsujen sisältämää tietoa käsitellään. Tiedon käsittelyssä tulee ottaa huomioon samat asiat kuin esimerkiksi normaalissa web-lomakkeessa, joten SQL-injektiot ovat uhkana.

3.2.1 Rajapintakutsun autentikointi

Rajapintakutsun autentikointiin on karkeasti jaoteltuna kaksi vaihtoehtoa. Joko kirjautumistiedot talletetaan istuntoon ja haetaan sieltä kutsujen yhteydessä, tai kirjautumistiedot vaaditaan jokaisen kutsun yhteydessä. Vaikkakin kirjautumistietojen tallettamisesta istuntotietoihin on joitain hyötyjä (esimerkiksi voidaan rajoittaa kutsujen määrää tilanteissa, joissa käyttäjä haluaa kuormittaa tietokantaa), se on vaikeampi toteuttaa ja avaa mahdollisuuden istuntokaappaukselle. Näiden seikkojen takia autentikaatio toteutetaankin jokaisen rajapintakutsun yhteydessä.

Loihdin-järjestelmä sisältää oman tapansa kirjautua käyttäjätilille. Tätä valmista kirjautumistoimintoa ei voida kuitenkaan suoraan käyttää, sillä rajapinnan on toteutettava käyttäjäkohtaisen tunnistautumisen lisäksi myös ympäristökohtainen tunnistautuminen. Loihdinin kirjautumisjärjestelmä tukee tällä hetkellä ainoastaan käyttäjäkohtaista kirjautumista.

Ympäristökohtainen tunnistautuminen saadaan ratkaistua siten, että ympäristökohtainen kirjautuminen hoidetaan kehyksen ulkopuolella. Tämän jälkeen ympäristöstä valitaan pääkäyttäjä, jonka tunnuksilla kirjaudutaan Loihdin-järjestelmään. Autentikoinnissa verrataan http-kutsun parametreina annettuja tunnuksia tietokannasta löytyviin ympäristökohtaisiin tunnuksiin. Kuvassa 2 on kuvattu autentikoinnin eteneminen.



Kuva 2: Rajapintakutsun autentikointi

Kuvassa 2 näkyvällä rajapintakutsulla tulee olla parametrina joko apicode tai username. Tämän lisäksi kutsulla on parametri password. Mikäli parametrina on username, annetaan username ja password Loihdin-kehykselle, joka autentikoi ja kirjautuu käyttäjän tilille, jos tunnukset olivat oikein. Jos parametrina oli apicode, avataan yhteys Loihdin-järjestelmän tietokantaan. Tässä kannassa suoritetaan haku ympäristökohtaisten tunnusten (apicode ja password) perusteella perusteella tauluun, jossa sijaitsevat järjestelmän ympäristöt. Jos haku palauttaa yhden tuloksen, kirjaudutaan kyseisen ympäristön pääkäyttäjän tilille. Tämä tietokantahaku parametrien perusteella vaarantaa rajapinnan Sql-injektioille, joita käsitellään seuraavassa luvussa.

3.2.2 SQL-injektiot

Loihdin hoitaa tietoluokkiin liittyvän tietoturvan, mutta käyttäjän autentikaatio pitää suorittaa kehyksen ulkopuolella. Koska käyttäjätiedot löytyvät tietokannasta, SQL-injektiot ovat uhka, johon pitää varautua.

SQL-injektiot ovat aina vaarana, kun SQL-kyselyyn lisätään dataa, joka on peräisin järjestelmän käyttäjältä. Tästä esimerkkinä voidaan käyttää esimerkiksi yksinkertaista hakutoimintoa. Olkoon SQL-kysely vaikka rakenteeltaan seuraavan koodiesimerkin mukainen:

```
String sql = "SELECT * FROM contact WHERE firstname LIKE "
      + name + "%'";
```

Koodiesimerkki 2: SQL-haku etunimen perusteella

Koodiesimerkin 2 name-muuttuja saa arvonsa käyttäjän syötteestä. Jos tämä syöte sisältääkin SQL-erikoismerkkejä, voidaan hakulausetta muokata. Syötteen paikalle voidaan esimerkiksi laittaa merkkijono “'-- TRUNCATE user;'”, jolloin suoritettavan SQL-lauseeksi muodostuu seuraavan esimerkin mukainen:

```
sql = "SELECT * FROM contact WHERE firstname LIKE "
      + "''--TRUNCATE user;'%'";
```

Koodiesimerkki 3: SQL-injektio

Esimerkin 3 toinen heittomerkki lopettaa hakukentän merkkijonon, jonka jälkeen tuleva “--” peruuttaa select-lauseen suorittamisen. Tämän jälkeen hyökkääjä voi tehdä uuden SQL-lauseen, TRUNCATE user, joka poistaa kaikki rivit user-taulusta, mikäli sellainen on. Puolipisteen jälkeinen “'%'” aiheuttaa SQL-virheen, joka pahimmassa tapauksessa tulostuu käyttäjälle, jolloin tämä saa tietoa järjestelmästä.

Javan tietokantakirjastot sisältävät välineet SQL-injektioiden estoon. Tekemällä kaikki kantaan menevät kyselyt käyttäen PreparedStatement-olioita, syötteenä tulevasta tiedosta suodatetaan kaikki erikoismerkit pois. PreparedStatementin käyttö onnistuu seuraavan koodiesimerkin osoittamalla tavalla:

```
Class.forName("com.mysql.jdbc.Driver");
connection = DriverManager.getConnection(url, "user", "passwd");
PreparedStatement pStat = connection.prepareStatement("SELECT * FROM "
      + "SystemContainers WHERE apicode = ? AND apipassword = ?");
pStat.setString(1, apicode);
pStat.setString(2, password);
ResultSet result = pStat.executeQuery();
if (result.first()){
      return result.getInt("containercontextid");
}
```

Koodiesimerkki 4: PreparedStatement

Koodiesimerkissä 4 ladataan ensin ohjelman muistiin mysql:n jdbc-ajurit, jonka jälkeen otetaan otetaan yhteys järjestelmän tietokantaan. Tämän jälkeen luodaan PreparedStatement, jolle annetaan konstruktorissa SQL-kysely, jossa muuttuvat osat on korvattu '?'-merkillä. Tämän jälkeen näiden merkkien tilalle asetetaan arvot PreparedStatementin setString()-metodilla. Tässä tapauksessa asetetaan vain merkkijonoja, mutta luokasta löytyvät metodit myös muiden tietotyyppien asettamiseen. Kun kaikki arvot on asetettu, suoritetaan PreparedStatement execute()-metodilla ja käsitellään tulosjoukko halutulla tavalla. Tässä esimerkissä noudetaan asiakastilistä tunniste.

4 Rajapinnan toteutus

Tässä luvussa käydään läpi työn toteutus. Aluksi tarkastellaan toteutusta yleiseltä tasolta ja mitä kirjastoja käytetään. Tämän jälkeen käydään läpi servletin toteutus, jonka jälkeen tarkastellaan jokaisen toiminnon käyttäminen ja toteuttaminen erikseen.

4.1 Rajapinnan määrittely

Rajapinnan määrittelyssä viitataan järjestelmän sisältämään tietoon riveinä selkeyden vuoksi. Rajapinnan kautta tulee voida tehdä seuraavat asiat:

1. kirjautuminen
2. tiedon luku
3. tiedon lisäys
4. tiedon muokkaus
5. tiedon poisto
6. virhetilanteiden käsittely ja tulosten palauttaminen
7. kaikkien tietoluokkien tiedon muokkaaminen
8. Loihdin-luokat rajapinnan käyttöön.

Vaikka autentikointi on päätetty suorittaa jokaisen kutsun yhteydessä, kirjautumiselle määritellään toteutettavaksi erillinen metodi, joka ottaa vastaan ainoastaan autentikaatiotunnukset ja palauttaa käyttäjätiedot. Kirjautumisen lisäksi vaatimuksiin kuuluvat tiedon käsittelyyn liittyvät metodit. Vaikka virhetilanteiden käsittely ja tulosten palauttaminen ovat kaikessa ohjelmoinnissa oleellinen osa, on ne tässä tilanteessa erikseen vaadittu, sillä niitä ei vanhoissa rajapinnoissa juurikaan ollut.

Vaatimukset 1-6 olivat rajapinnan käyttäjälle näkyviä seikkoja. Jotta vältymme yrityksessä sisäisesti vanhan rajapinnan ongelmalta, jossa jokaiselle rajapintatarpeelle pitää tehdä oma toteutus, tulee uuden rajapinnan pystyä käsittelemään kaikkien Loihdinin tietoluokkia, vaikka niihin tulisi muutoksia tai uusia tietoluokkia lisättäisiin. Tämän lisäksi osana työtä pitää tuottaa Loihtimeen luokat rajapinnan kutsumiseen, jotta keskustelu eri Loihdin-järjestelmien välillä saadaan toimimaan ilman lisätyötä.

Yleisesti rajapinnan kutsumisesta määriteltiin seuraavat asiat: Rajapintaa käytetään http-kutsuilla, joissa metodeiksi kelpaavat GET- ja POST-metodit. Enkoodauksena käytetään UTF-8:aa, kuten web-liikenteessä on ohjeistettu [5]. UTF-8:aa käyttää myös suurin osa web-sivuista, tällä hetkellä arviolta 73,4 % [6]. Kussakin rajapintaoperaatiossa välitetään ohjelmalle kolme parametria: tunnus, salasana ja xmlquery-parametri, joka sisältää suoritettavan operaation määrittävän xml-aineiston. Tähän poikkeuksena on login-operaa-

tio, jolle välitetään vain tunnus ja salasana.

Tunnuksena ja salasanana voidaan rajapinnassa käyttää joko järjestelmän käyttäjän käyttäjätunnusta ja salasanaa tai ympäristökohtaista apicode/api-password-paria. Käyttäjätunnukset soveltuvat paremmin Adensy CRM:ää käyttäviin client-sovelluksiin, kun taas apicode/apipassword-paria voidaan käyttää esimerkiksi muiden tietojärjestelmien datan synkronointiin tiettyyn asiakasympäristöön. Login-operaatiossa ei tietenkään voi käyttää api-tunnuksia, sillä se kohdistuu yksittäiseen käyttäjään.

4.2 Rajapinnan käyttö

Seuraavaksi tarkastellaan rajapinnan käyttötapauksia. Tiedon haku tarkastellaan ennen kirjautumista, sillä hakuoperaation ja kirjautumisen paluuaineistot ovat saman muotoisia. Kirjautumisen yhteydessä palautettava materiaali sisältää aina kirjautuneen työntekijän tiedot, kun taas hakuoperaatiossa palautettava materiaali määritellään hakuaineistossa. Esimerkkien tietoluokat ja kentät vastaavat Adensy CRM -järjestelmän tietoluokkia ja kenttiä.

Kunkin operaation esittelyn yhteydessä ilmenevä operaation osoite vaihtuu riippuen siitä, kuka järjestelmän tarjoaa ja mihin Loihdin-järjestelmään otetaan yhteyttä. Koska rajapinta on toteutettu käyttäen Java-servlettiä, voidaan itse valita osoite, johon rajapinta liitetään. Rajapinta lukee osoitteestaan muutaman asian, jotka on syytä ottaa huomioon. Jos oletetaan, että servletille välitetään pyynnöt osoitteisiin, jotka vastaavat http://www.testi.fi/api/*- mukaisia osoitteita, tapahtuu esimerkiksi kirjautumisoperaatio tekemällä pyyntö osoitteeseen http://www.testi.fi/api/loihdin/production/login. Tässä osoitteessa “loihdin”-osa määrää järjestelmän, jonka tietoa käsitellään, “production”-määrää, käytetäänkö tuotanto- vai testausympäristöä ja “login” määrittelee käytettävän rajapintaoperaation. Pyynnöt, joita ei kohdisteta tämän kaavan osoitteisiin, estetään.

Selvyyden vuoksi jatkossa käytetään Adensy CRM yhteydessä olevan rajapinnan osoitetta. Rajapinnan käyttö vaatii lisäksi dokumentaation käsiteltävän järjestelmän tietoluokista.

4.2.1 Tiedon haku

Tiedon haussa xmlquery-parametri sisältää xml-muotoisen kuvauksen haettavasta tiedosta. Xml-aineisto voi olla esimerkiksi seuraavanlainen:

```
<dataclass name="company" limit="50">
      <condition>
            <field>name</field>
            <operator>LIKE</operator>
            <parameter>%ABB%</parameter>
      </condition>
      <condition>
            <field>owner</field>
            <operator>=</operator>
            <parameter>1</parameter>
      </condition>
</dataclass>
```

Koodiesimerkki 5: Tiedon haku

Aineiston juurielementtinä oleva dataclass-elementti määrittää tietoluokan, josta tietoa haetaan. Tämä tapahtuu name-attribuutin mukaan. Haluttaessa voidaan haulle määrittää myös limit-attribuutti, joka rajaa paluuaineiston halutuun määrään rivejä.

Juurielementin alle määritellään hakuehdot, jotka koostuvat verrattavasta kentästä (field-elementti), vertailuoperaattorista (operator-elementti) ja verrattavasta arvosta (parameter-elementti). Field-elementin arvoksi voidaan antaa minkä tahansa tietoluokan kentän nimi tai tietoluokan rowid. Operator-elementin arvoksi voidaan antaa mikä tahansa SQL-vertailuoperaattori. Parameter-operaattorin arvoksi annetaan arvo, johon field-elementtiä halutaan verrata. Hakuehtoja voi olla mielivaltainen määrä, myös hakuehtojen

pois jättäminen on mahdollista, jolloin kysely palauttaa kaikki kyseisen tietoluokan rivit. Muussa tapauksessa rajapinta palauttaa hakuehtoja vastaavat rivit. Koodiesimerkissä 5 esitellyn mukainen haku palauttaa seuraavan tuloksen:

```
<return>
    <company rowid="7" titlevalue="ABB Oy">
        <name>ABB Oy</name>
        <description>Ajax-kumppani</description>
        <type listid="5">kumppani</type>
        <status listid="1">aktiivinen</status>
        <faxnumber/>
        <wwwadress/>
        <phonenumber/>
        <phonenumber2/>
        <picture>0</picture>
        <officalCode/>
        <industry/>
        <owner rowid="1">Matias Hilden</owner>
        <emaildomainend/>
        <number>0</number>
        <parentcompany/>
        <revenue>0</revenue>
        <turnoverclass/>
        <personel>0</personel>
        <employeesizeclassification/>
        <companyformclass/>
        <industryclassification/>
        <shippingstreet/>
        <shippingpostalcode/>
        <shippingcity/>
        <shippingstate/>
        <shippingcountry/>
        <billingstreet/>
        <billingpostalcode/>
        <billingcity/>
        <billingstate/>
        <billingcountry/>
        <billingcurrency/>
        <billingpaymentterms/>
        <billingcontact/>
        <invoicesendtype/>
        <evoiceaddress/>
        <evoiceint/>
        <invoiceemail/>
        <rootFolder/>
        <externalDScompanyid/>
        <lastactivity/>
        <overdueinvoicecount type="int">0</overdueinvoicecount>
        <invoicestatus listid="0">Ei avoimia laskuja</invoicestatus>
    </company>
</return>
```

Koodiesimerkki 6: Hakuoperaation palautusaineisto

Koodiesimerkissä 6 näkyy, kuinka rivin tiedot palautuvat rajapinnan käyttäjälle. Return-elementin lapsiksi listautuvat hakuehtoihin täsmänneet yritykset company-elementeiksi. Kullakin company-elementillä on attribuutteina rowid, joka sisältää arvonaan rivin tunnisteen, ja titlevalue, joka on rivin ni-

mi. Rivin nimen muodostuminen voidaan määritellä Loihdin-järjestelmän tietoluokan asetuksissa. Company-elementtien lapsiksi listautuvat tietoluokan kentät ja niiden arvot.

Suurin osa kentistä sisältää vain kentän arvon tai ei mitään, mikäli kentälle ei ole annettu arvoa. Joukosta kuitenkin poikkeavat type, status, owner, overdueinvoicecount ja invoicestatus. Näistä kentistä type, status ja invoicestatus ovat listakenttiä, jotka käyttöliittymän puolella näkyvät vakioarvoisina pudotusvalikkoina. Näille kentille annetaan tekstiarvon lisäksi listid-attribuutti, jolla on arvona kyseisen valinnan tunniste. Lisäys- ja päivitysoperaatioissa listakentän arvo määritellään tätä tunnistetta käyttämällä, joten se kannattaa hakuoperaatiossakin palauttaa selvyyden vuoksi. Owner-kenttä on objektikenttä eli linkitys toiseen riviin, joka voi osoittaa joko samaan tai toiseen tietoluokkaan. Tämäkin määritellään Loihdin-järjestelmän tietoluokan asetuksissa. Rivin nimen lisäksi palautetaan rowid-attribuutti, joka kertoo linkitetyn rivin tunnisteen. Tätä tunnistetta voidaan puolestaan käyttää kyseisen rivin tietojen hakuun. Viimeisenä erikoisuutena on overdueinvoicecount, joka on laskukaavakenttä. Tälle kentälle voidaan asetuksissa määritellä laskukaava, jonka perusteella kentän arvo päivittyy aina, kun riviä päivitetään. Type-attribuutti määrittelee laskukaavakentän tyypin, joka voi olla joko int, double tai date.

4.2.2 Käyttäjän kirjautuminen

Kirjautumisoperaatio saa parametreiksi vain käyttäjätunnuksen ja salasanan. Olettaen, että käyttäjätunnus ja salasana täsmäävät, palauttaa rajapinta seuraavanlaisen vastauksen:

```
<return>
      <person rowid="13" titlevalue="Matias Hilden">
            <firstname>Matias</firstname>
            <lastname>Hilden</lastname>
            <phonenumber>0401231223</phonenumber>
            <mobilenumber></mobilenumber>
            <fax></fax>
            <OrganizationUnit rowid="20">Adensy
Oy</OrganizationUnit>
            <status listid="1">active</status>
            <email>matias.hilden@adensy.com</email>
      </person>
</return>
```

Koodiesimerkki 7: Kirjautumisen paluuaineisto

Koodiesimerkin 7 paluuaineisto sisältää tietoluokka-elementin, joka kirjautumisen tapauksessa on aina “person”-niminen. Tietoluokka-elementin attribuutteina ovat rowid, joka on tietoluokan ilmentymän yksilöllinen tunniste, ja titlevalue, joka koostuu kuvaustiedostossa määritellyn kaavan mukaisesti tietoluokan kentistä.

Tietoluokka-tagin lapsina löytyvät kenttien tagit. Tagien nimeäminen tapahtuu tietoluokan kenttien mukaan ja niiden arvoina on kyseisen kentän arvo. Kentillä, joiden tietotyyppi on lista tai linkitys toiseen tietoluokkaan, on paluuaineistossa lisätietoja xml-elementtien attribuutteina. OrganizationUnit elementillä saadaan rowid, joka määrittää linkitetyn tietoluokan rivin. Listalla puolestaan on listid-attribuutti, joka määrittelee listan arvon tunnisteen. Näitä tietoja voidaan käyttää hyväksi esimerkiksi linkitetyn tietoluokan rivin hakemiseen.

4.2.3 Tiedon lisäys

Tiedon lisäyksessä xmlquery-parametri sisältää xml-muotoisen kuvauksen lisättävistä riveistä. Dokumentin juurielementtinä on xmldata-elementti, jonka alla on tietoluokka-elementtejä. Yrityksiä lisättäessä hakuaineisto voisi

olla esimerkiksi seuraavan muotoinen:

```
<xmldata>
      <dataclass name="company">
            <name>Yritys Oy</name>
            <officalCode>1-123456</officalCode>
            <phonenumber>123 456 7788</phonenumber>
            <email>asiakaspalvelu@yritys.fi</email>
            <streetaddress>Mannerheimintie 101</streetaddress>
            <city>Helsinki</city>
            <postalcode>00430</postalcode>
            <country>Finland<country>
            <owner>13</owner>
      </dataclass>
      <dataclass name="company">
            <name>Firma Oy</name>
            <officalCode>1-554321</officalCode>
            <phonenumber>123 654 2233</phonenumber>
            <email>asiakaspalvelu@firma.fi</email>
            <streetaddress>Mannerheimintie 102</streetaddress>
            <city>Helsinki</city>
            <postalcode>00430</postalcode>
            <country>Finland<country>
            <owner field="lastname">Meikäläinen</owner>
      </dataclass>
</xmldata>
```

Koodiesimerkki 8: Tiedon lisäys

Kuten aineistosta huomataan, tietoluokka-elementit ja niiden lapsina olevat kenttä-elementit käyttävät samaa nimeämiskäytäntöä kuin hakuoperaation palautusaineisto, jota koodiesimerkki 6 käsitteli. Lisäysoperaation yhteydessä voidaan lisätä usea rivi samaan aikaan lisäämällä dataclass-elementtejä. Esimerkissä huomioitavaa on owner-kenttä, joka on linkitys toiseen tietoluokkaan, tässä tapauksessa person-tietoluokkaan. Näissä linkityskentissä voidaan elementille asettaa halutessa attribuutteja, joiden avulla voidaan muuttaa linkityksen tekoa.

Tällä hetkellä attribuutteja on kaksi: “useLinkId” ja “field”. Ilman kummankaan määritystä rajapinta hakee rivin, jonka nimi vastaa kenttä-attribuutin mukaista arvoa. useLinkId-attribuutti määrittelee käyttääkö rajapinta linkityksen luontiin rowid:tä vai rivin nimeä (koodiesimerkissä 15 näkyvä titlevalue-attribuutti). Attribuutille voi antaa arvon “true” tai “false”. field-attribuut-

ti puolestaan määrittelee linkitetyn tietoluokan kentän, jota verrataan kenttä-elementin arvoon. Ainoastaan toinen näistä attribuuteista voidaan antaa kenttä-elementille kerrallaan.

Paluuaineistona aikaisemmasta lisäyspyynnöstä rajapinta palauttaa xml-aineistona tiedot onnistuneesti lisätyistä riveistä koodiesimerkin 9 mukaisesti.

```
<return>
      <result index="0" success="true">1234</result>
      <result index="1" success="true">1235</result>
</return>
```

Koodiesimerkki 9: Lisäysoperaation paluuaineisto

Paluuaineistossa index-attribuutit viittaavat kutsun dataclass-elementteihin. Ensimmäinen dataclass-elementti saa arvokseen index=”0”, seuraava saa index=”1” ja niin edelleen. Success-attribuutit sisältävät totuusarvon operaation onnistumisesta. Mikäli tämä arvo on true eli lisäys onnnistui, sisältää result-elementti arvonaan lisätyn rivin rowid-tunnisteen.

4.2.4 Tiedon muokkaus

Tiedon muokkauksen xmlquery-parametrin xml-aineisto on lähes identtinen lisäysaineiston kanssa, joten sen kuvaukseen voidaan käyttää koodiesimerkkiä 5. Ainoa ero päivitysoperaatiossa on päivitettävän kentän määrittely, joka tapahtuu dataclass-elementissä.

Päivitettävän kentän määrittelyssä on useita vaihtoehtoja, jotka toimivat samaan tapaan kuin linkitettyjen kenttien määrittely. Dataclass-elementille voidaan antaa rowid-attribuutti, jolloin päivitys kohdistuu riviin, jonka rowid-tunniste vastaa annettua arvoa. Vaihtoehtoisesti voidaan määritellä myös attribuutit “field” ja “value”, jolloin päivitys kohdistetaan ensimmäiseen ri-

viin, jonka field-attribuutin määräämä kenttä sisältää value-attribuutin arvon.

Paluuaineisto on päivitysoperaatiossa identtinen lisäysoperaation paluuaineiston kanssa, joka kuvattiin koodiesimerkissä 6. Result-elementtien tekstiarvona rowid-tunniste ei tosin viittaa lisättyyn riviin vaan päivitettyyn riviin.

4.2.5 Tiedon poisto

Tiedon poistossa määritellään ainoastaan poistettavan rivin rowid-tunniste koodiesimerkin 10 mukaisesti:

```
<dataclass name="company">
      <rowid>135</rowid>
</dataclass>
```

Koodiesimerkki 10: Tiedon poisto

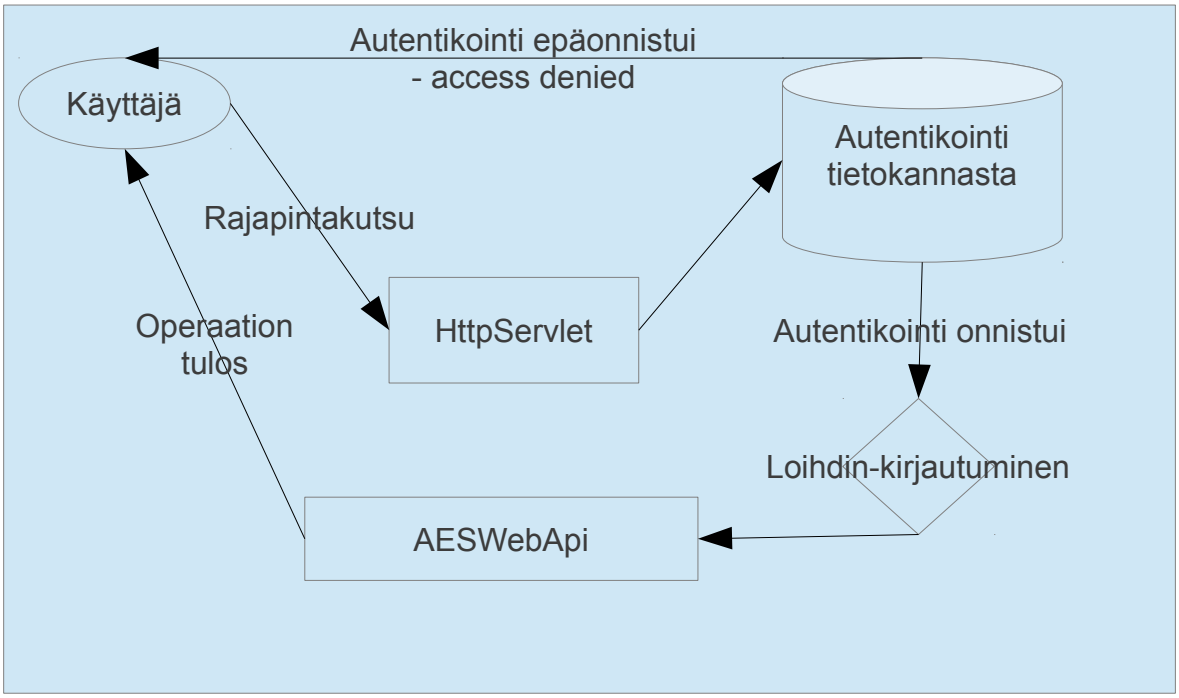
Poisto-operaatiossa voidaan poistaa tällä hetkellä vain yksi rivi kerrallaan, tämä rajoite johtuu operaation peruuttamattomuudesta. Paluuaineistona operaatio palauttaa vastauksena koodiesimerkki 6 kaltaisen vastauksen, sisältäen yhden result-elementin operaation onnistumisesta.

4.3 Rajapinnan toteuttaminen

Työ toteutettiin tekemällä Java-servlet, sillä sen liittäminen osaksi muuta järjestelmää oli vaivatonta. Servletin tehtävänä oli dekoodata UTF-8-muotoinen kutsu, välittää saadut parametrit ohjelmalle ja palauttaa kutsujalle ohjelman muodostama vastaus. Tämän lisäksi servletin tuli estää kutsut rajapintaan kuulumattomiin osoitteisiin ja suodattaa mahdolliset SQL-injektioyritykset.

Xml-aineistojen muodostamiseen käytetään Javan peruskirjastoista löytyviä w3schools xml-kirjastoja sekä Xercesin xml-kirjastoja. Näitä kirjastoja käytetään muuallakin järjestelmässä, joten niiden valinta tukee järjestelmän yhdenmukaisuutta.

Rajapintakutsun eteneminen on kuvattu kuvassa 3:

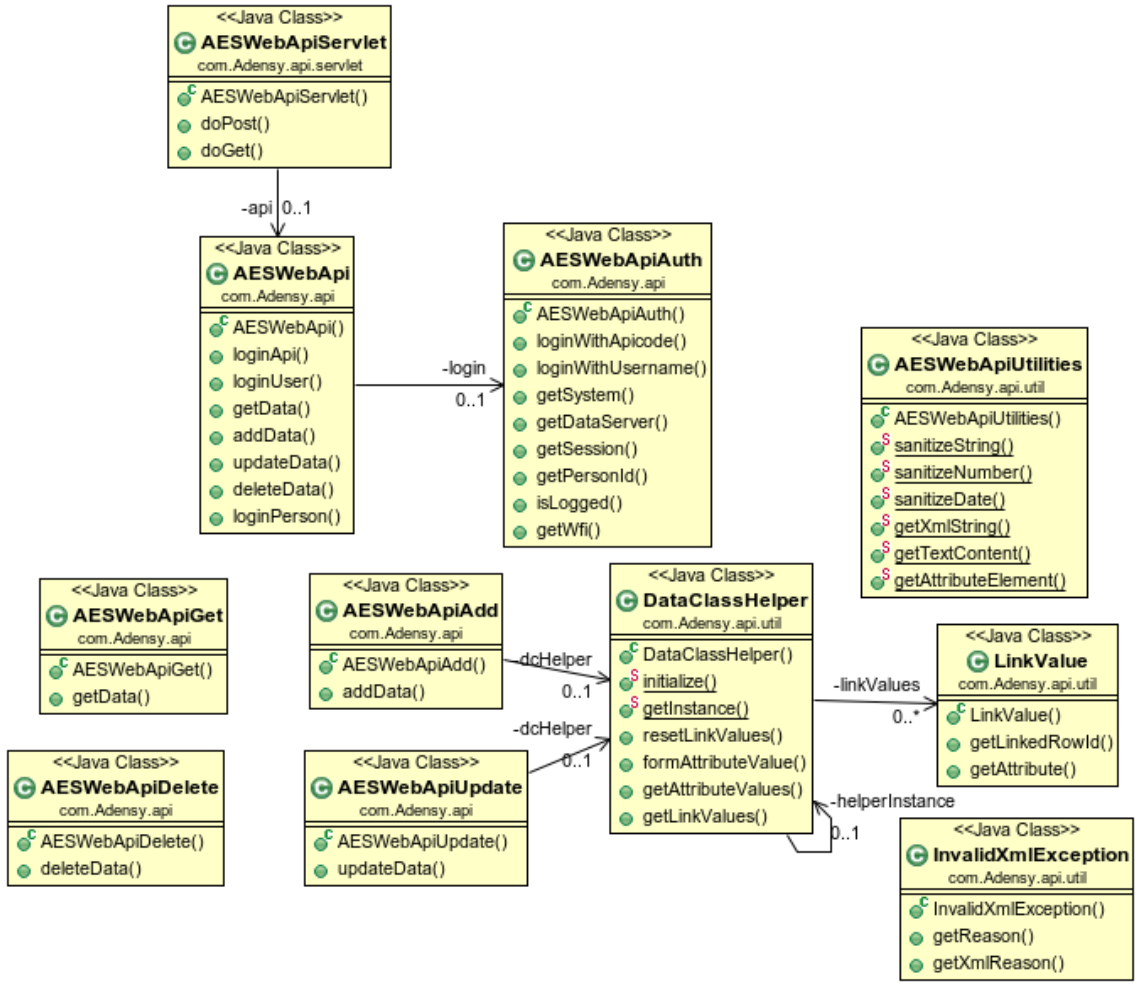


Kuva 3: Rajapintakutsun eteneminen

HttpServlet saa rajapintakutsun käyttäjältä. Servlet välittää kutsun yhteydessä tulleet käyttäjätunnukset autentikaatioluokalle, joka tarkistaa, ovatko ne oikein. Mikäli tunnukset eivät täsmää, palautetaan käyttäjälle tästä virheilmoitus. Mikäli taas tunnukset ovat oikein, autentikaatioluokka kirjaa käyttäjän Loihdin-järjestelmään, ja ohjelman suoritus ohjautuu pyydetyn operaation suorittavalle luokalle, joka on kuvassa nimetty “AESWebApi”. Operaation tulos palautetaan takaisin käyttäjälle.

4.3.1 Ohjelman rakenne

Ohjelman perusrakenne on melko yksinkertainen: Servlet ottaa vastaan kutsuja ja välittää ne AESWebApi-luokalle, joka toimii wrapperina rajapinnan eri toiminnoille. Kullekin rajapinnan toiminnolle on oma luokkansa, joiden yhteiset koodikokonaisuudet on sijoitettu apuluokkiin. Näin voidaan välttää saman koodin toistumista, ja mahdollistetaan uusien rajapintatoimintojen vaivaton lisääminen. Kuvassa 4 on näkyvissä rajapinnan luokkakaavio.



Kuva 4: Luokkakaavio

Palvelin ohjaa rajapintakutsut kuvan 4 yläosassa näkyvällä AESWebApiServlet-luokalle kutsumalla sen doGet()- tai doPost()-metodia, joka kutsuu AES-

WebApi-luokan loginApi()- tai loginUser()-metodia. Mikäli tämä kirjautuminen onnistuu, kutsutaan tämän jälkeen rajapintatoimintoa vastaavaa metodia: getData() tiedon haulle, addData() tiedon lisäykselle, updateData() tiedon päivitykselle, deleteData() rivin poistolle ja loginPerson() kirjautumiselle. Metodin kutsu johtaa vastaavan luokan luontiin (AESWebApiGet, AESWebApiAdd, AESWebApiUpdate, AESWebApiDelete) ja tämän luokan ainoan metodin kutsuun. Tässä metodissa tehdään pyynnön vaatimat toimenpiteet ja muodostetaan paluuaineisto toimenpiteiden onnistumisen perusteella. Tarvittaessa luokat käyttävät DataClassHelper-luokan apufunktioita.

AESWebApiServlet lukee rajapintakutsusta käytettävän Loihdin-järjestelmän, kirjautumistunnukset ja xml-aineiston operaatiota varten. Vastaavasti se myös palauttaa kutsujalle rajapinnan muodostaman xml-aineiston tai virheellisen autentikaation tapauksessa Http 403 -virheen.

AESWebApiAuth-luokka hoitaa käyttäjän autentikoinnin lisäksi yhteyden muodostamisen käytettävään Loihdin-järjestelmään. Käyttäjän autentikoinnin yhteydessä on SQL-injektion vaara, joka estetään käyttämällä Javan tietokantatoteutuksen tarjoamaa PreparedStatement-kyselyä, jota käytiin läpi kappaleen 3.2.2 koodiesimerkissä 4. AESWebApiAuth-luokka tarjoaa myös Loihdinin käytölle pakollisien luokkien get-metodit.

DataClassHelper-luokka on singleton-mallinen luokka, joka sisältää toimintoja Loihdinin tietoluokkien käsittelyyn. Käytännössä luokka toteuttaa muunnokset xml-elementtien arvoista tietoluokan arvoon ja pitää kirjaa linkityskenttien arvoista. Näitä arvoja varten on tehty LinkValue-luokka.

AESWebApiUtilities-luokka sisältää muita yleisesti käytettyjä toimintoja, kuten käyttäjäsyötteiden sanitointia SQL-injektioiden varalta. Näitä sanoitointimetodeja käytetään, kun tietokantaan otetaan yhteys kehyksen kautta eikä PreparedStatement-luokkaa voida käyttää. Luokka sisältää myös xml-aineiston lukuun liittyviä toimintoja, joita käytetään kaikkialla ohjelmassa. Inva-

lidXmlException-luokka on Javan Exception-luokasta periytetty virheluokka.

AESWebApi-luokka toimii wrapperina rajapinnan eri toiminnoille, kuten luvun alussa kerrottiin. Tällä hetkellä se sisältää luokkinaan AESWebApiGet, AESWebApiLogin, AESWebApiUpdate ja AESWebApiDelete luokat. Näiden luokkien tehtävänä on ottaa vastaan rajapintakysely xml-aineistona ja palauttaa tiedon sisältävä xml-aineisto takaisin AESWebApi-luokalle, joka välittää sen edelleen AESWebApiServlet-luokalle.

4.3.2 HttpServlet

Servletin toteutus voidaan jakaa kolmeen osaan: kutsun autentikointiin, kutsun välittämiseen oikealle rajapintaluokalle ja vastauksen palauttamiseen käyttäjälle. Koodiesimerkissä 11 tarkastellaan autentikointia:

```
private AESWebApi api;
private String systemName;
private boolean isDesign;

public void doGet(HttpServletRequest request,
      HttpServletResponse response) {

      request.setCharacterEncoding("UTF-8");
      response.setCharacterEncoding("UTF-8");

      if (!analyzeUrl(request.getRequestURI())){
            response.sendError(HttpServletResponse.SC_NOT_FOUND);
            return;
      }

      api = new AESWebApi(systemName, isDesign, "server.xml");

      if (request.getParameterMap().containsKey("apicode") &&
            request.getParameterMap().containsKey("password")){
            if (!api.loginApi(request.getParameter("apicode"),
                  request.getParameter("password"))) {

      response.sendError(HttpServletResponse.SC_FORBIDDEN);
                  return;
            }
      }else if (request.getParameterMap().containsKey("username") &&
            request.getParameterMap().containsKey("password")){
            if (!api.loginUser(request.getParameter("username"),
                  request.getParameter("password"))) {
                  response.sendError(HttpServletResponse.SC_FORBIDDEN);
                  return;
            }
      }
```

Koodiesimerkki 11: Autentikointi servletissä

Esimerkissä asetetaan aluksi tulevan ja lähtevän aineiston enkoodaukseksi UTF-8. Tämän jälkeen välitetään pyynnön URI-osoite analyzeUrl-metodille, jonka tarkoituksena on suodattaa virheellisiin osoitteisiin tulevat pyynnöt pois, jolloin palautetaan käyttäjälle Http 404 -virhe. Metodi myös lukee luokkamuuttujiin systemName- ja isDesign-arvot osoitteesta, joilla api-kutsu kohdistetaan oikeaan Loihdin-järjestelmään. Tämän jälkeen luetaan http-kutsun parametreista rajapintatunnukset ja välitetään AESWebApi-luokan loginApi()- tai loginUser()-metodille. Jos kirjautuminen epäonnistuu, palautetaan käyttäjälle http 403 -virhe. Autentikoinnin jälkeen rajapintakutsu ohjataan eteenpäin koodiesimerkin 12 mukaisesti:

```
String requestUrl = request.getRequestURI();
String xmlQuery = request.getParameter("xmlquery");
if (xmlQuery != null){
      xmlQuery = URLDecoder.decode(xmlQuery, "UTF-8");
}
Document responseXml = null;
if (requestUrl.endsWith("/get")) {
      try {
            responseXml = api.getData(xmlQuery);
      } catch (InvalidXmlException e) {
            responseXml = e.getXmlReason();
      }
}else if(requestUrl.endsWith("/add")){
      try {
            responseXml = api.addData(xmlQuery);
      } catch (InvalidXmlException e) {
            responseXml = e.getXmlReason();
      }
}else if (requestUrl.endsWith("/update")){
      try{
            responseXml = api.updateData(xmlQuery);
      }catch (InvalidXmlException e){
            responseXml = e.getXmlReason();
      }
}else if (requestUrl.endsWith("/delete")){
      try{
            responseXml = api.deleteData(xmlQuery);
      }catch (InvalidXmlException e){
            responseXml = e.getXmlReason();
      }
}else if (requestUrl.endsWith("/login")){
      try{
            responseXml =
                  api.loginPerson(request.getParameter("username"),
                  request.getParameter("password"));
      }catch (InvalidXmlException e){
            responseXml = e.getXmlReason();
      }
}
```

Koodiesimerkki 12: Rajapintakutsun ohjaaminen oikealle luokalle

Koodiesimerkissä luetaan rajapintakutsun osoitteen loppuosasta operaatio ja kutsutaan AESWebApi-luokasta sille varattua metodia. Tämä metodi puolestaan ohjaa pyynnön oikealle luokalle. Kukin metodi palauttaa paluuaineiston, joka sijoitetaan responceXml-muuttujaan. Tämän paluuaineiston käsittely on näkyvillä seuraavassa koodiesimerkissä:

```
if (responseXml != null){
      XMLSerializer xmlSer = new XMLSerializer();
      OutputFormat out = new OutputFormat(responseXml);
      out.setEncoding("UTF-8");
      out.setVersion("1.0");
      out.setOmitXMLDeclaration(false);
      out.setOmitDocumentType(true);
      out.setIndenting(true);
      out.setIndent(4);
      xmlSer.setOutputFormat(out);
      xmlSer.setOutputCharStream(response.getWriter());
      xmlSer.serialize(responseXml);
      response.getWriter().close();
}
```

Koodiesimerkki 13: Aineiston palauttaminen

Koodiesimerkissä 13 otetaan esimerkissä 12 muodostettu paluuaineisto ja ajetaan sen XMLSerializer-luokan läpi käyttäjälle. Tässä määritellään formaatti ihmiselle luettavaksi. Tosin mikäli tarve ilmenee, on formaatti helppo ehdottaa myös mahdollisimman vähän tilaa vieväksi. Serialisoijalle määritellään ulostuloksi HttpResponce-luokan ulostulo, jolloin serialisointi kirjoitetaan suoraan rajapinnan käyttäjälle.

4.3.3 Tiedon haku ja muokkaus Loihdin-teknologian avulla

Kun AESWebApi-luokka ohjaa rajapintapyynnön haun tai muokkauksen suorittavalle luokalle, xml-aineistosta muodostetaan tietoluokan ilmentymä. Xml-aineistossa määritellyn tietoluokan nimen mukaan järjestelmästä haetaan oikea tietoluokka, josta ilmentymä muodostetaan.

Muokkauksessa ja lisäyksessä tälle lisätään ne kentät ja arvot, jotka on xml-aineistossa määritelty, olettaen, että samat kentät löytyvät myös järjestelmän tietoluokasta ja arvo on hyväksyttävä. Virhetilanteessa palautetaan rajapinnan käyttäjälle viesti siitä, mikä kenttä aiheutti virheen. Poistossa asetetaan ilmentymälle vain rivin tunniste. Ilmentymä välitetään tämän jälkeen Loihdinin metodeille, jotka suorittavat loput toimenpiteet. Tiedon haussa

xml-aineistosta muodostetaan hakuehdot, jotka välitetään Loihdinin metodille. Loihdin palauttaa hakutulokset, joista muodostetaan käyttäjälle palautettava aineisto.

4.4 Loihdin-luokkien toteuttaminen

Rajapintaa käyttävien Loihdin-luokkien tehtävänä on mahdollistaa eri Loihdin-järjestelmien muokata toistensa tietoja. Näiden luokkien toiminnallisuutta tulee voida ohjata Loihdinin asetustiedostoilla ja Loihdin tarjoaa tätä varten yläluokan. Yläluokan perimällä luokka saa konstruktorissaan xml Element -olion, joka sisältää kaikki luokalle määritellyt asetukset.

Tämän työn osana toteutetaan luokat seuraaville rajapinnan käyttötarkoituksille: tiedon lisäys, tiedon muokkaus ja tiedon poisto. Tiedon haku ja kirjautuminen jätetään tässä vaiheessa toteuttamatta, sillä niille ei ole tässä vaiheessa tarvetta.

Kuten luvussa 4.2 nähtiin, tiedon lisäys ja muokkaus eivät xml-aineistoltaan juurikaan eroa toisistaan. Muokkauksessa vain määritellään mitä riviä halutaan muokata, kun taas lisäyksessä vain lisätään rivi. Tiedon poistossa ei määritellä muuta kuin rivi, joka halutaan poistaa, ja kunkin toiminnon paluuaineistot ovat muodoltaan samanlaisia. Oleellisena erona tiedon poisto ei tue muuta kuin yhden rivin poistoa kerrallaan. Tässä vaiheessa ei lisäyksessä eikä muokkauksessa tarvitse tukea monen rivin lisäystä eikä päivitystä.

Aikaisemmista seikoista johtuen luokissa voidaan käyttää pitkälti samanlaista toimintaperiaatetta: Muodostetaan asetustiedoston perusteella lähetettävä aineisto, avataan http-yhteys rajapintaan ja luetaan tieto operaation onnistumisesta. Tämän takia riittääkin, että käydään läpi vain lisäysoperaatio ja tarkastellaan sen yhteydessä muokkaus- ja poisto-operaatioiden erot.

Luodaan aluksi ohjelmallisesti xmldata- ja dataclass-elementit luvun 4.2.3 koodiesimerkin 5 mukaisesti. Dataclass-elementille asetetaan name-attribuutin arvoksi asetuksista saatu kohdejärjestelmän tietoluokka. Muokkaus- ja poisto-operaatioissa tässä kohtaa lisätään myös dataclass-elementille rowid-attribuutti, joka luetaan asetustiedostosta. Tämän jälkeen käydään läpi asetuksista saadut parametrit, joista saadaan muodostettua dataclass-elementin lapsielementit. Koodiesimerkissä 14 käsitellään xml-aineiston ohjelmallinen muodostus.

```
Document document = docBuilder.newDocument();
Element docEle = document.createElement("xmldata");

Element dataclassElement = document.createElement("dataclass");
dataclassElement.setAttribute("name", datasource);

for (FunctionParameter parameter : getParameters()) {
      String value;
      value = parameter.getMatchwith();
      String elementName = parameter.getMatchfor();

      if (value != null && !value.equals("")){
            Element attributeElement =
AESWebApiUtilities.getAttributeElement(
                  document, elementName);
            attributeElement.appendChild(
                  document.createTextNode(value));
            dataclassElement.appendChild(attributeElement);
      }
}

docEle.appendChild(dataclassElement);
document.appendChild(docEle);
```

Koodiesimerkki 14: Xml-aineiston muodostus ohjelmallisesti

Esimerkissä FunctionParameter-olio on yksittäinen asetuksista saatu parametri, jolta otetaan getMatchfor()- ja getMatchwith()-metodeilla rajapintakutsun kenttä ja sen arvo. Xml-elementtien muodostuksessa käytetään jo aikaisemmin tehtyjä apuluokkia. Tyhjiä arvoja ei lisätä aineistoon. Poisto-operaatiossa ei näitä parametreja käydä läpi, sillä sen rajapintakutsuun ei sisälly muita tietoja kuin dataclass-elementin rowid- ja name-attribuutit.

Seuraavassa esimerkissä näkyy xml-aineiston lähettäminen rajapinnalle:

```
URL url = addParameters(path, wfi, URLEncoder.encode(AESWebApiUtili-
ties.getXmlString(docEle), "UTF-8"));
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setConnectTimeout(3000);
conn.setDoOutput(true);

StringBuilder data = new StringBuilder();
BufferedReader rd = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
String line = rd.readLine();
while (line!=null){
      data.append(line.trim());
      line = rd.readLine();
}

Document responseDocument = null;
try {
      responseDocument = docBuilder.parse(new InputSource(new
StringReader(data.toString())));
} catch (Exception e) {
      e.printStackTrace();
      throw new InvalidXmlException("Unparseable xml.");
}

Element returnElement = (Element) responseDocument.getFirstChild();
Element resultElement = (Element) returnElement.getFirstChild();

if (targetVariableName != null){
      try{
            int rowid =
Integer.parseInt(AESWebApiUtilities.getTextContent(resultElement));
            Variable target = wfi.findVariable(targetVariableName);
            TextVariable idVar = new TextVariable("rowid");
            idVar.setValue(rowid);
            target.Synchronize(idVar);
      }catch (Exception e){}
}
```

Koodiesimerkki 15: Rajapintakutsun lähettäminen

Koodiesimerkissä 15 suoritetaan rajapinnan kutsuminen ja paluuaineiston tulkitseminen. Yhteyden muodostamiseen käytetään Javan peruskirjastoista löytyvää HttpURLConnection-luokkaa. Tämän luokan ilmentymälle asetetaan metodiksi GET, katkaisuajaksi 3 sekuntia ja doOutput(true), sillä halutaan lähettää dataa. Lähetettävä xml-aineisto enkoodataan UTF-8 muotoon, jonka

jälkeen avataan yhteys rajapintaan getInputStream()-metodilla. Metodin kutsun yhteydessä HttpURLConnection lähettää kaikki annetut parametrit rajapinnalle ja InputStream palautuu siinä vaiheessa, kun rajapinta palauttaa vastauksen. Paluuaineistona tuleva xml-aineisto parsitaan Document-olioksi, josta käydään lukemassa lisätyn rivin tunniste, mikäli asetuksissa on määritelty tunniste luettavaksi, ja tieto operaation onnistumisesta. Päivitys- ja poisto-operaatioissa luetaan vain tieto operaation onnistumisesta.

4.5 Rajapinnan toiminnan todentaminen

Rajapintaa testattiin aluksi käsin lähettämällä selaimen kautta rajapintakutsuja. Kun virheet oli saatu seulottua pois, siirryttiin testauksessa rajapintaa käyttävien Loihdin-luokkien pariin. Nämä luokat oltiin otettu käyttöön yrityksen sisäisessä Android-projektissa. Luokkiin jouduttiin tekemään jonkin verran muutoksia tässä vaiheessa, sillä Androidin xml-kirjastoissa ei ollut kaikkia samoja ominaisuuksia kuin pc-puolen Javan kirjastoissa. Suurimpana ongelmana oli Element-luokan getTextValue()-metodin puuttuminen, jonka käyttö jouduttiin kiertämään.

Kun Android-projektissa rajapinnan käyttö sujui ongelmitta, siirryttiin rajapinnan testauksessa käyttämään pilottiasiakasta, joka alkoi tekemään rajapintakutsuja omilla tunnuksillaan. Tässä vaiheessa ongelmaksi muodostui lähinnä se, että täysin dynaamisesta rajapinnasta oli vaikea muodostaa helposti ymmärrettävää käyttöohjetta. Jotain muitakin pieniä viilauksia tehtiin muun muassa virhetilanteissa rajapinnan käyttäjälle palautuvaan aineistoon, jotta käyttäjä saisi helpommin selville, mitä oli kirjoittanut väärin.

Pilottiasiakkaan jälkeen voitiin todeta, että rajapinta toimii kuten pitää ja että se voidaan ottaa käyttöön muihinkin projekteihin.

5 Päätelmät

Opinnäytetyössä tutkittiin web-rajapintojen toteuttamista Loihdin-teknologian avulla. Lisäksi toteutettiin tiedonsiirron rajapinta Loihdin-järjestelmiin.

Rajapintateknologioita tutkittaessa löydettiin vaihtoehtoinen tietoformaatti vanhalle xml-formaatille, jota voidaan tulevaisuudessa hyödyntää, mikäli tulee tarve vähentää siirrettävän tiedon kokoa. Opittiin myös tietoturvariskeistä, jotka piti ottaa huomioon rajapinnan toteutuksessa.

Itse rajapinnan toteuttaminen sujui hyvin, eikä ensimmäisen version ohjelmointiin mennyt ajallisesti viikkoa kauempaa. Myöhemmissä versioissa parannettiin muun muassa virhetilanteiden käsittelyä ja rajapinnan käyttäjälle palautettavia viestejä. Joillekin hakuehdoille lisättiin myös lisätoiminnallisuuksia.

Rajapintaa kehittäessä suurimmaksi haasteeksi muodostui uusien versioiden taaksepäin yhteensopivuus. Myös rajapintadokumentaation laatiminen oli hankalaa. Koska rajapinta oli tiedon suhteen dynaaminen, dokumentaatiosta tuli melko yleisellä tasolla asiat kertova ja vaikeasti ymmärrettävä. Tämän takia päätettiin tehdä kullekin rajapintatarpeelle tapauskohtainen ohjeistus sitä myöten miten tarpeita ilmenee.

Rajapinnan jatkokehityksessä on tarpeellista selventää virhetilanteissa käyttäjälle palautuvaa viestiä entisestään. Rajapinnan käytön laajetessa tulee myös harkita vaihtoehtoisten tietoformaattien, kuten JSON, tukemista. Rajapintaa voisi myös laajentaa tukemaan muutakin kuin tiedonsiirtoa, kuten vaikkapa käyttäjätilien ja asiakasympäristöjen hallintaa.

Lähteet

1. Toni Halsti, Toimitusjohtaja, Adensy Oy, Keskustelu Tekes-hakemuksesta, 2012.
2. Frank P. Coyle, Xml, Web Services, and the Data Revolution, 2002.
3. Nicholas C. Zakas, Professional JavaScript for Web Developers, 2011.
4. Jason Hunter, William Crawford, Java Servlet Programming, 2001.
5. Buildwith Trends, UTF-8 Usage Statistics, 2012.
6. W3Techs, Usage of character encodings for websites, 2012.