Opinnäytetyö (AMK)

Tietojenkäsittely

Tietojärjestelmät

2013

Ari Mäkeläinen

KONFIGURAATIOPALVELIMEN TOTEUTUS WISE-PROJEKTILLE

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Tietojärjestelmät

Elokuu 2013 | 36 sivua

Ohjaajat: Anne Jumppanen

Ari Mäkeläinen

KONFIGURAATIOPALVELIMEN TOTEUTUS WISE PROJEKTILLE

Opinnäytetyön tavoitteena on toteuttaa konsepti konfiguraatiopalvelimesta WISE-projektille. Projekti tutkii mahdollisuuksia käyttää TV-taajuuksien vapaata kaistaa tiedonsiirtoon kognitiiviradioille. Palvelimen tulee kyetä vastaanottamaan http-pyyntöjä ja vastaamaan niihin.

Ongelmaa on lähestytty pääosin ohjelmoijan näkökulmasta koodin ollessa pääaiheena. Tällaisessa työssä on tarpeen selvittää myös alustan ja työkalujen merkitystä. Työtä varten on tutkittu olemassa olevia järjestelmiä ja pystytetty toimiva kehitysympäristö. Työssä tutustutaan myös digitaaliseen allekirjoitukseen ja XML-kieleen.

Opinnäytetyössä toteutettu palvelu saatiin toimimaan määrittelyjen mukaan. Koska päätelaitteita ei ollut olemassa, palvelun testaamista varten kehitettiin myös prototyyppi asiakasohjelmasta. Prototyypin avulla pystyttiin varmistamaan palvelun toimivan oikein.

Halutunlaisen palvelun toteuttaminen todettiin mahdolliseksi.

ASIASANAT:

Java, Ohjelmointiympäristö, Ohjelmointi, Apache, J2EE, Palvelin

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Information Systems

August 2013 | 36 pages

Instructors: Anne Jumppanen

Ari Mäkeläinen

IMPLEMENTATION OF CONFIGURATION SERVER TO PROJECT WISE

The target of this thesis is to develop a working concept of a configuration server to WISE-project. The project studies the usage of cognitive radios using an unused space of TV-frequencies for data transfer. A server has to receive http-requests as well as send responses to them.

The study is mainly concerned with the problem of a programmer's viewport, the main focus being on the code. While the code is important, some background information for development tools and environment is needed at work like this. For this thesis existing other similar systems were studied and a working development environment was set up. Also digital signatures and XML-language are discussed in the thesis.

The server developed for the thesis was made to work way according to the specifications demanded. Since there was not a client device, a developing prototype of client software was needed for testing. With the client prototype the server could be tested like in live environment.

It was ascertained that the implementation of a desired server is possible.

KEYWORDS:

Java, Programming, Apache, J2EE, Programming environment, Server

SISÄLTÖ

KUVAT

SANASTO

WSD	White Space Device – laite, joka hyödyntää TV:n taajuusaluetta
IDE	Integrated Development Enviroment - kehitys-/ohjelmointiympäristö
Palvelin	Fyysinen tai virtuaalinen laite, jossa ajetaan palvelusovelluksia. Joskus myös yksittäisiin palvelusovelluksiin viitataan kyseisellä termillä.
Brute Force attack	Hyökkäystapa, jossa koitetaan joko arvaamalla tai kirjastoja käyttäen murtaa salasana tai muu vastaava suojaus.
Servlet	Sovelluspalvelimella suoritettava Java-luokka, joka vastaanottaa pyyntöjä ja vastaa niihin.
Tehdasluokka	Luokka, jonka avulla voidaan luoda ilmentymiä toisesta luokasta.

1 JOHDANTO

Opinnäytetyö liittyy WISE-projektiin, joka tähtää käyttämättömien TV-taajuuksien hyödyntämiseen kognitiivisen radion ja tietokantojen avulla (Wise 2013). Langattomia tiedonsiirtoteitä on nykyään käytössä useita, esimerkiksi WLAN, jota käytetään laajalti. WLAN:n ongelmana on se, että kantomatka on vaatimaton. Lisäksi tukiasemat samalla alueella häiritsevät toisiaan, mikäli ne kuuluvat eri verkkoihin. Edellinen johtuu siitä, että WLAN:a on käytössä vain muutamia kanavia, joiden taajuudet menevät osittain päällekkäin ja tämä johtaa verkon paikoittaiseen tehottomuuteen. Taajuuksia on käytössä vähän, koska niiden rajallisen määrän vuoksi ne ovat varsin hintavia.

Tarve uudenlaiselle tavalle siirtää dataa langattomasti on siis perusteltu. Jotta vältettäisiin vanhojen tapojen pullonkaulat, käytetyistä taajuuksista tulisi pitää kirjaa. TV-taajuusaluetta käyttävät TV-lähetinten lisäksi myös langattomat radiomikrofonit. Käyttöä taajuuksilla siis on. WISE-projektissa kehitetään testausympäristöä, jonka avulla pyritään näiden taajuuksien tehokkaaseen käyttöön. Ympäristöön kuuluu geolokaatiotietokanta, johon kerätään paikkatietoon perustuen tietoa käytössä olevista taajuuksista, ja mikäli tätä tietoa päivitetään käytön mukaan, saadaan taajuuksista irti maksimaalinen hyöty ilman, että häiritään toisia käyttäjiä.

Opinnäytetyössä toteutettiin prototyyppi Fairspectrum Oy:n ideoimasta mahdollisuudesta käyttää konfiguraatiopalvelinta osana aiemmin toteutettua geolokaatiotietokanta-kokonaisuutta. Ideana on, että WSD-laite hakee konfiguraation, minkä jälkeen se ottaa yhteyttä taajuuspalvelimeen, joka varmistaa, että laitteella on toimiva konfiguraatio ja tämän jälkeen antaa käytettävät taajuudet laitteelle. Mikäli konfiguraatio puuttuu tai on virheellinen, taajuuspalvelin pyytää laitetta hakemaan konfiguraation uudelleen.

Opinnäytetyössä lähdettiin kehittämään järjestelmää jonka tarve tunnistettiin, mutta toteutuksesta ei ollut lyöty juuri mitään lukkoon. Ensiksi tuli selvittää mitä eri toteutusvaihtoehtoja on. Saaduista vaihtoehdoista valitaan sellainen joka

vaikuttaa toimivimmalta. Kaikkia vaihtoehtoja ei kannata hylätä vaikka niitä ei alkuun näyttäisi tarvitsevan. Myöhemmin jokin valituista voi osoittautua soveltumattomaksi. Silloin on hyvä olla valmiiksi mahdollisia lähestymistapoja. Tutkimusmenetelmänä käytettiin konstruktiivista tutkimusta.

Järjestelmävaatimusten rinnalla kartoitettiin konfiguraation koostumusta. Mitä sen tulee sisältää ja mitä ei. Konfiguraation muotoa ja rakennetta, sekä eri tapoja ehkäistä mahdollisia tietoturvauhkia tutkittiin. Järjestelmää ryhdyttiin rakentamaan jo ennen, kuin kaikkia määrityksiä oli saatu valmiiksi. Samanaikaisella kehittämisellä ja tutkimuksella oli toki miinuspuolensa. Ominaisuuden valmistuessa sitä ei tarvita lainkaan, tai sen käyttötarkoitusta joudutaan muuttamaan uusien määritysten mukaiseksi.

Konfiguraatio sisältää tiedot laitteesta, taajuuksista, lähetystehosta ja muista hyödyllisistä osista. Jotta konfiguraation voidaan vahvistaa tulleen oikeasta paikasta, se allekirjoitetaan sertifikaatilla. Lähtökohtana oli luoda turvallinen tapa välittää konfiguraatiotiedot asiakaslaitteille. Konfiguraatiotietojen siirtomuotona käytetään XML:ää ja ne siirretään https-protokollaa käyttäen.

2 TOTEUTUSVALINNAT

2.1 Käyttöjärjestelmä

Jotta palvelu saataisiin julkaistua, tarvitaan palvelin. Palvelimen käyttöjärjestelmää valittaessa tulisi kiinnittää huomiota siihen, miten hyvin sen hallinta on toteutettu ja kuinka aktiivisesti siihen toimitetaan tietoturvapäivityksiä. Hallinnalla tässä yhteydessä on useampia merkityksiä. Ensimmäinen asia, joka hallinnassa korostuu, on järjestelmän käyttöympäristö. Hallitaanko palvelinta fyysisesti paikan päällä, lokaalista nopeasta lähiverkosta vai etäyhteydellä internetin yli. Nopeasta lähiverkosta tai paikan päällä hallittaessa toimivat kaikki hallintamuodot graafisesta liittymästä komentoriviin. Internetin yli hallinnoitaessa yhteyden nopeus ja vakaus jättävät jäljelle vain selain- ja komentorivipohjaisen hallinnointimuodon, koska graafisen liittymän käytettävyys rampautuu.

Hallintaan liittyy myös palveluiden ja ohjelmistojen asennus, poisto ja saatavuus. Aina tulisi selvittää, haluttujen palvelujen saatavuus kyseiselle alustalle sekä niiden asennuksen ja konfiguroinnin sujuvuus. Poisto on tärkeää siksi, että yleensä palvelin hankitaan suorittamaan tiettyä tehtävää eikä ylimääräisiä palveluita tai ohjelmistoja tarvita. Ylimääräiset palvelut ja ohjelmistot voivat aiheuttaa ennakoimattomia tietoturvariskejä, mikäli niiden päivityksistä ei huolehdita. Tämä taas aiheuttaa ylimääräistä työtä ylläpitäjille.

Tietoturvapäivitykset ovat tärkeitä, koska täydellisesti suojattua palvelua ei pystytä luomaan muuta kuin lokaalisti yksinään toimivalle koneelle. Tietoturvapäivitysten lisäksi tulee vahtia myös muita päivityksiä. Ohjelmistojen päivitykset voivat myös rikkoa aiempiin versioihin tehdyt muokkaukset, jolloin niiden muutoksiin ja ominaisuuksiin pitää tutustua ennen asennusta. Päivitysten toimivuus on helpointa testata, mikäli käytössä on identtinen järjestelmä, johon ne voidaan ensin asentaa ja sitten testata, syntyykö ongelmia.

Palvelimille on useita käyttöjärjestelmävaihtoehtoja: Microsoft Windowsin ja Apple OSX:n serveriversiot, Oraclen Solaris, BSD-järjestelmät ja lukuisat Linuxin

jakelut. Tämän projektin osalta ei tarvinnut käyttää aikaa järjestelmän valintaan, koska sellainen oli jo esiasennettuna. Käyttöjärjestelmä Amazonin EC2 palvelimella oli Linuxin jakeluversio Ubuntu.

Ubuntu palvelinalustana poikkeaa yleisemmin käytössä olevista Debian- CentOS- tai SuSe-jakeluista siinä, että se on suunnattu peruskäyttäjille. Ubuntun hakemistopuu poikkeaa yleisen käytännön määrittämästä sekä konfiguraatio- että työhakemistojen osalta. Tämä aiheutti aluksi hieman päänvaivaa, koska itse käytän kehitysympäristönä CentOS:ää, joka on RedHatin lähdekoodeista käännetty ilmainen Enterprise-tason käyttöjärjestelmä. Helppo ratkaisu konfigurointipulmaan olisi kääntää halutut palvelut itse oletusasetuksilla, mutta tällöin niiden ylläpito ja päivittäminen olisi jatkossa vaikeaa ja epäkäytännöllistä. Ratkaisin ongelman luomalla symboliset linkit Ubuntun omista hakemistosijainneista yleensä käytössä oleviin sijainteihin, jolloin ylläpitäjän vaihtuessa kaikki löytyy sieltä mistä kuuluukin.

2.2 Sovelluspalvelin

Sovelluspalvelin huolehtii kirjoitetun ohjelman suorituksesta palvelimella. Tähän tarkoitukseen on olemassa useita sekä vapaan lähdekoodin että suljettuja ratkaisuja. Suljettujen järjestelmien vahvuuksia ovat laajat tukimuodot ja yleensä nopeasti reagoiva asiakaspalvelu. Haittapuolena kaupallisissa järjestelmissä on korkea hankintahinta, toistuvat lisenssimaksut tai molemmat. Vapaan lähdekoodin ratkaisuissa hankinta on yleensä yksinkertaista, hinta halpa tai olematon. Joihinkin ratkaisuihin on saatavilla nopeaa asiantuntija-apua, mutta tämä on yleensä maksullista. Verkossa olevilta keskustelupalstoilta ja postilistoilta löytää ongelmiin ratkaisuja ilman suuria rahallisia investointeja. Tosin ylläpitäjänkään työaika ei ole ilmaista, joten ratkaisut palvelimen hankkimiseksi kannattaa miettiä tarkoin.

Vapaan lähdekoodin sovelluspalvelimia ovat esimerkiksi Tomcat ja jBoss (Apache Software Foundation 2013; JBossWeb, 2013). Molemmat kykenevät toimimaan omillaan ilman erillisen esikäsittelijän käyttöä. Vaikka esikäsittelijän

käyttö ei ole välttämätöntä, se on silti suotavaa. Yleisin esikäsittelijä, jota käytetään, on Apache http-palvelin (FAQ Httpd Wiki 2013). Apachen avulla on mahdollista ohjata liikennettä käyttäen virtuaalisia osoitteita palveluihin. Virtuaalisten osoitteiden käyttö on suotavaa, koska oletuksena Java web-sovellukset käyttävät oletuspolkua host:8080/projekti/sovelma, joka on epäkäytännöllinen. Virtuaalinen osoite taasen on muotoa host/<sovelman alias>, joka on lyhyempi ja helpompi muistaa.

2.3 Tiedon siirtomuodot

Tietoa pystytään verkossa siirtämään monin tavoin: XML, JSON, puhdas teksti, konekielinen data näin muutaman mainitakseni. Puhtaasti tekstimuotoinen tieto on ihmiselle helpoin tapa, mutta ohjelmista joutuu tekemään tarpeettoman monimutkaisia, mikäli niiden pitää ymmärtää selkokielistä kirjoitusta. Toisessa ääripäässä tietoa liikutellaan puhtaasti konekielisenä, tällöin ihmisillä on vaikeuksia varmistaa tiedon oikeellisuus.

Ratkaisuksi ääripäiden välille on kehitetty useita muotoja, joista laajasti käytettyihin kuuluu XML. Se on kuvannuskieli, jonka pääkomponentit ovat tunniste- ja datakenttä (Introduction to XML 2013). Tunnistekenttä sisältää datakentän ”nimen” ja datakenttä varsinaisen arvon. Esimerkiksi <foo>bar</foo> esittelee muuttujan nimeltä foo, jonka arvona on bar. Huomioitavaa on, että kun datakenttä on avattu, se on myös suljettava.

XML on pitkälti standardoitu tapa esittää tietoa. Dokumenttien yhdenmukaisuuden varmistamiseksi voi käyttää DTD:ä tai schemaa (DTD Tutorial 2013; Quin 2013). Kummatkin näistä ovat karkeasti kuvattuina oikeinkirjoitusoppaita itse XML-dokumentille. Schema on näistä uudempi, joten päädyin käyttämään sitä.

Schemassa kerrotaan, miten dokumentti on järjestetty, minkä muotoista dataa kentät sisältävät ja tarvittaessa esimerkiksi arvojen vaihteluvälin. Schemaa voi käyttää ohjeena siitä, miten XML-dokumentti muodostuu. Ohjelmoijalle se tarjoaa mahdollisuuden käyttää sopivaa rajapintaa, joka generoi tarvittavat luokat dokumentin tekoon.

2.4 Projektin hallinta

Työkaluohjelmaksi valittiin Eclipse, joka sen lisäksi, että se oli tuttu, on myös eräs käytetyimmistä Java-IDE:istä. Sen plugin-kirjastoista löytyy huomattava määrä erilaisia lisäosia, joita laskettiin projektissa tarvittavan.

Projektissa käytetään kirjastoja, joita Java ei oletuksena tarjoa. Niiden hallintaan on löydettävä jokin tapa, joka mahdollistaa koodin tehokkaan kirjoittamisen kirjastoriippuvuudet huomioiden. Ratkaisuksi löytyi Apache Maven, joka on Apache-säätiön alun perin Java-ohjelmointiin kehittämä työkalu. Se on saavuttanut aseman ohjelmistoteollisuuden standardina erityisesti Java-projekteissa (Smart 2012). Nykyään Mavenin ohjelmointikielien tuki on laajentunut Javan lisäksi myös muihin kieliin. Sen avulla pystyy hallitsemaan riippuvuuksia, automatisoimaan testausta, luomaan dokumentaatiota ja tarjoamaan parhaita käytäntöjä ohjelmoijille.

Maven tarjoaa ohjelmistoprojektille projektirungon (Maven:Archetype), yhtenäisen kansiorakenteen, joka ei vaihdu, vaikka vaihtaisi IDEä sekä raportointirajapinnan. Tämä helpottaa projektien hallintaa, koska pääosin projektihenkilöstö voi käyttää omien mieltymystensä mukaisia työkaluja. Projektissa voi siis käyttää esimerkiksi sekä Eclipseä että Netbeanssia. Maven tukee suurinta osaa yleisesti käytetyistä versionhallintaohjelmistoista SCM Pluginin avulla (SCM Matrix 2013).

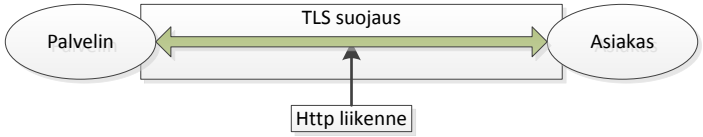
Versionhallinta on osa projektin hallintaa. Kun kirjoittaa koodia, ei voi välttyä virheiltä ja välillä on tarpeen palata jonkin verran taaksepäin kehityksessä, jotta saa korjattua ohjelman toimintaa. Lisäksi versionhallinta auttaa tehtäessä töitä eri paikoista, kun esimerkiksi kotona tehdyt muutokset voi hakea suoraan päätyökoneelle tai kannettavalle. Versionhallintaan on tarjolla useita työkaluja, joista nykyään käytetyimmät ovat Git ja Subversion (SVN). Markkinoilla on myös jokunen kaupallinen järjestelmä, mutta niiden hinnoittelusta joko ei saa selvää kuvaa, tai sitten hinta on projektin kokoon nähden korkea (AccuRev 2013; BitKeepper 2013; Perforce 2013).

Edellisessä alaluvussa mainittuun luokkien generointiin tarvitaan myös työkalu. Sellainen on JAXB, jolla pystytään luomaan XML-schemoista Javan luokkia sekä myös toisinpäin schemoja luokista. Paketti tarjoaa myös kirjastot, joiden avulla XML:n käsittely ohjelman sisällä voidaan suorittaa. Koska JAXB hoitaa XML-tiedostojen oikean muotoilun, ohjelmoijan ei tarvitse käyttää omia resurssejaan siihen, vaan hän voi keskittyä itse ohjelmiston tekemiseen. (GlassFish>>Metro>>Project JAXB 2013.)

3 TIETOTURVA

3.1 Tiedonsiirto

Tieto liikkuu verkon yli pääasiassa https:ää käyttäen. Https itsessään ei ole protokolla OSI-mallissa esitettyjen tasojen mukaan, vaan on yhdistelmä http- ja SSL/TLS-protokollista. Näistä http kuuluu sovelluskerrokseen, kun taas SSL/TLS sijoitetaan istuntokerrokseen. (ISO/IEC 7498-1:1994.)



Kuva 1. Https:n toiminnallisuus

Kuvassa 1 on yksinkertainen esitys https-liikenteestä. Palvelimella on käytössä sertifikaatti, jonka se lähettää asiakkaalle kättelyn yhteydessä. Asiakas vastaa kättelyyn muodostamalla avaimen, joka lähetetään takaisin salattuna palvelimen omalla sertifikaatilla. Palvelin käyttää asiakkaan avainta muodostaakseen pääavaimen ja lähettää sen asiakkaalle. Kun molemmilla osapuolilla on käytössä pääavain uusia yhteyksiä varten, ne käyttävät sitä muodostaessaan yhteyskohtaisia avaimia, joilla lähetettävä tieto salataan. (Trustwave 2013.)

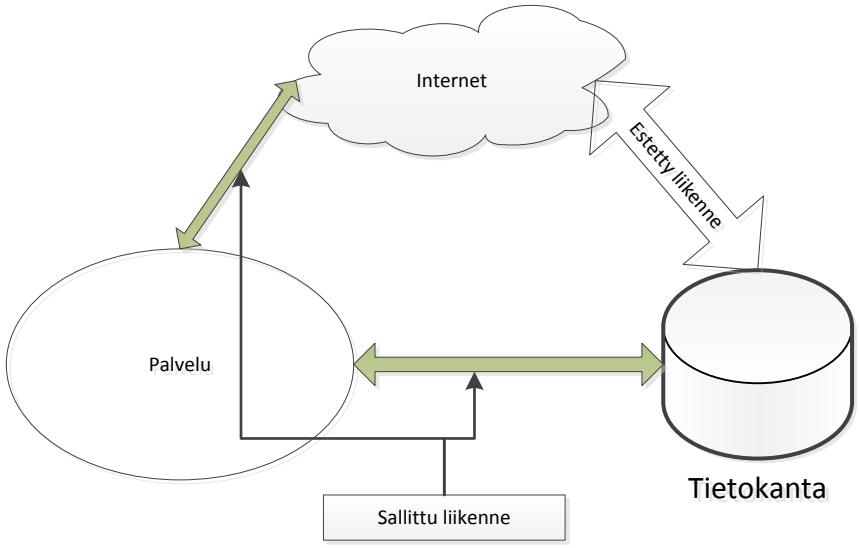
Tietoa siirretään julkisen verkon yli, joten se on suojattava. Suositeltavaa olisi salata liikenne tehokkaasti, lisäämällä asiakaspuolelle oma sertifikaatti, joka tunnetaan palvelinpäässä. Tämä kuitenkin johtaisi laitteiden teho- ja kustannusvaatimusten kasvamiseen. Data, jota liikutetaan, on tärkeää järjestelmän kannalta, joten sen muuttumattomuus on vähintäänkin vahvistettava. Tähän palataan luvun 4 alaluvussa 4.

3.2 Tiedon käsittely

Järjestelmä käsittelee dataa, jonka voi sanoa olevan identifioivaa. Tällöin tiedon saatavuus järjestelmän sisällä on suojattava, samalla kun estetään asiaankuulumaton käyttö. Siihen ei saa päästä käsiksi ulkopuolelta muuten kuin ennalta määrätyillä tavoilla. Tietoa liikkuu järjestelmässä sekä sisään- että ulospäin. Järjestelmään syötteenä tulevasta tiedosta on tarkistettava, onko se vaadituntlaista. Tähän asiaan palataan luvussa 4.

3.3 Tiedon säilytys

Tietoa säilytetään tietokannassa, jonka tulee samaan aikaan olla saavutettavissa ja vahvasti suojattu.



Kuva 2. Tietokannan liikenne

Saatavuus tässä yhteydessä tarkoittaa, että palvelun on päästävä tietoon käsiksi aina tarvittaessa. Tietoon kiinnipääseminen on toteutettu siten, että palvelu pääsee tekemään kyselyitä ja lisäyksiä tietokantaan. Muilta tietokannan käyttö on estetty. Tämä on esitetty kuvassa 2. Toteutus vaatii sekä palvelimen että tietokannan asetusten muokkaamista siten, että tietokanta on turvassa.

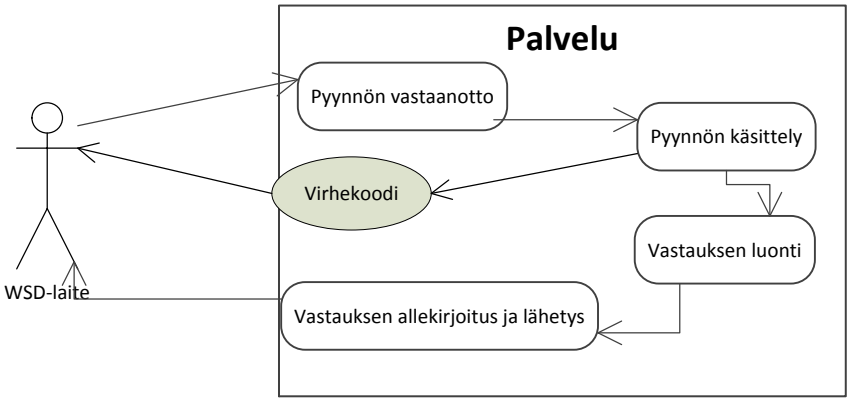
Palvelimen päässä asia voidaan ratkaista estämällä kaikki ulkopuolinen liikenne tietokantaan. Kyseinen ratkaisumalli toimii, mikäli sekä palvelu että tietokanta sijaitsevat samassa järjestelmässä. Mikäli palvelu on hajautettu useampaan järjestelmään saavutettavuuden parantamiseksi, edellä mainittu tapa edellyttäisi kaikkien palvelua tarjoavien koneiden liittämistä VPN-putkella kantaa ylläpitävään koneeseen. Tässä tapauksessa parempi tapa on sallia https-liikenne tunnistuksella, jolloin sertifikaatteja käyttämällä pystytään varmentamaan kumpikin osapuoli.

Tietokannan asetuksia lähdetään muokkaamaan ensiksi vaihtamalla oletuskäyttäjänimet ja salasanat. Edellä mainittu toimi estää tiedon vaarantumisen, mikäli tietokantaan pyritään käsiksi Brute force menetelmällä (Toponce 2013). Seuraava askel on muokata tietokannan sisäiset käyttöoikeudet halutulle tasolle. Se, miten kyseinen toiminto suoritetaan, riippuu siitä, mitä tietokantajärjestelmää käytetään. Tähän kannattaa käyttää apuna kunkin järjestelmän dokumentaatiota ja mikäli mahdollista, soveltaa parhaita käytäntöjä niiden ollessa saatavilla. Viimeiseksi luodaan palvelua varten oma käyttäjä ja tarvittavat taulut.

Käyttäjälle ei kannata antaa oikeuksia muuhun kuin niihin toimiin, joita palvelu tarvitsee. Kirjoitusoikeus niihin tauluihin, joihin tarvitsee kirjoittaa ja lukuoikeus niihin, joista haetaan tietoa. Tämä estää suuremmat vahingot mikäli kirjautumistunnisteet ja sertifikaatti päätyvät vääriin käsiin.

4 PALVELU

Tässä luvussa käydään läpi konfiguraatiopalvelun toiminta. Lähtökohtana on WSD-laitteelta saapuva pyyntö, johon palvelu vastaa.



Kuva 3. Tiedon kulku palvelussa

Kuvassa 3 esitetään, miten tieto siirtyy palvelun sisällä ja lopuksi palautetaan laitteelle valmiina konfiguraationa tai virhekoodina.

4.1 Pyyntö

Pyyntö käsittää kolme elementtiä: tunnistetiedon ja paikan GPS-arvoparina. Tunnistetietona toimii String-muotoinen UUID (Rfc4122). UUID:n käyttö identifioivana tietona on kannattavaa, koska sen arvaaminen tai generointi on hankalaa. GPS-arvopari koostuu kahdesta desimaaliarvosta, joiden tarkkuus on 10 desimaalia.

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

    String wsdid = request.getParameter("id");
    String longi = request.getParameter("lo");
    String lati = request.getParameter("la");
```

Kuva 4. Parametrien vastaanotto

Pyyntö saapuu palvelimelle HttpRequestina, josta luetaan kuva 4 mukaisesti parametrit String-muotoisiin muuttujiin. Stringin käyttö tietomuotona tässä vaiheessa on tarpeen, koska Javan perustietotyyppeihin ei kuulu desimaalilukua. Floatin käyttö tilanteessa, jossa ei ole tiedossa, tullaanko kyseistä lukua tulevaisuudessa käyttämään laskutoimituksiin, ei ole järkevää sen pyöristysongelmien vuoksi (Goldberg 1991). String saa sisältää mitä tahansa merkkejä, jolloin palvelu ei kaadu vahingossa tai tarkoituksella lähetettyyn virheelliseen pyyntöön.

4.2 Pyynnön käsittely

Ennen kuin pyyntöä voi alkaa käsitellä, on varmistettava, että se on oikean muotoinen ja sisältää kaikki vaaditut arvot.

```
74          if (wsdid.trim().length() == 0 | longi.trim().length() == 0
75                  | lati.trim().length() == 0) {
76              response.sendError(HttpServletResponse.SC_NO_CONTENT);
77              return;
78          }
79          if (!wsdid
80                  .matches("[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}")
81                  | !test(longi) | !test(lati)) {
82              response.sendError(HttpServletResponse.SC_UNSUPPORTED_MEDIA_TYPE);
83              return;
84          }
```

Kuva 5. Parametrien tarkistus

Pyynnön saapuessa parametrien sisältö tarkastetaan. Kaikille parametreille tehdään ensiksi tarkistus, onko arvoa asetettu lainkaan. Tämä tapahtuu kuvassa 5 riveillä 74-75, joilla verrataan saadun parametrin merkkien lukumäärää nollaan. Mikäli tarkastuksessa yksikin parametri on tyhjä, lähetetään vastauksena rivillä 76 http-virhekoodi 204, joka kertoo lähettäjälle, ettei pyyntö ollut riittävä. UUID-arvoa verrataan säännölliseen lausekkeeseen kuvan 5 rivillä 80, jolla pyritään varmistamaan, että se on oikean muotoinen.

GPS-arvopari tarkistetaan myös, tosin tämän tarkastusta varten piti asentaa apche.commons.lang kirjasto, koska Java ei tue suoraan muuttujien tietotyyppien automaattista tunnistamista. Kuvassa 6 on esitelty metodi, joka ottaa vastaan lähetetyn tiedon ja palauttaa tiedon siitä, onko annettu merkkijono numeerinen vai ei.

```
234    private boolean test(String item) {
235        return org.apache.commons.lang3.math.NumberUtils.isNumber(item);
236    }
```

Kuva 6. Metodi syötteen tarkistukseen

Kuvassa 5 rivillä 82 lähetetään http-virhekoodi 415, joka ilmaisee, ettei jokin tai jotkut annetuista parametreista ole oikeassa muodossa. Kun tarkistukset on läpäisty, siirrytään luomaan vastausta.

4.3 Vastauksen luonti

Kun pyyntö on tarkastettu ja hyväksytty, se välitetään palvelun suorittavalle osalle, joka on luokka nimeltä Datatest.

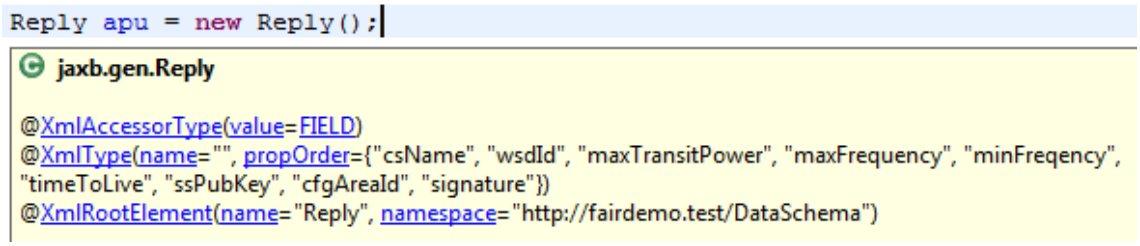
```
101         Datatest dt2 = new Datatest();
102         try {
103             dt2 = new Datatest(wsdid, longi, lati);
104         } catch (NullPointerException e1) {
105             e1.printStackTrace();
106         }
```

Kuva 7. Tietojen lähetys

Kuvassa 7 luodaan olio dt2 ensin tyhjänä rivillä 101 ja uudelleen try-catch lohkon sisällä rivillä 103 parametreilla varustettuna. Syy siihen, ettei dt2:a luoda suoraan parametrisoituna on välttää ongelmia kääntäjän kanssa. Kun olio tarvitsee parametreja, se on sijoitettava try-catch:n sisään, jotta mahdollinen tyhjä syöte ei kaataisi sovellusta. Näin on siitäkin huolimatta, että jo aiemmin on tarkistettu syötteiden olemassaolo. Olion esittelyä ei voi kumminkaan suorittaa

lohkon sisällä, koska silloin kääntäjä ei voi olla varma onko kyseistä oliota luotu ja prosessi kaatuu.

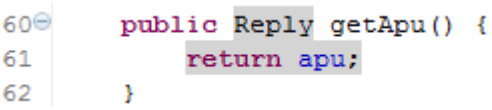
Datatestissä luodaan apu-olio JAXBilla generoidusta Reply-luokasta, johon on määritelty XML-dokumentti. Kuvassa 8 on karkea esitys kyseisestä luokasta.



Kuva 8. Reply-luokan kuvaus

Kohdassa @XmlAccessorType kerrotaan, että XML-käsittelijän tulee kohdella tietoa luokan sisällä ensisijaisesti kenttinä. @XmlType:n tyhjä name-kenttä kertoo, että kyseessä ei ole tietotyyppi. Elementti propOrder esittää tietokenttien oikean järjestyksen valmiissa dokumentissa. Kentässä @XmlRootElement name kerrotaan, mikä on dokumentin juurikenttä. Namespacen jälkeen löytyy käytettävän scheman sijainti ja nimi.

Luotua oliota ei ole tarkoitus hyödyntää Datatest-luokan sisällä, vaan ainoastaan lisätä siihen tarvittavat arvot. Tällöin tarvitaan palautusmetodi, joka on esitetty kuvassa 9.



Kuva 9. Palautusmetodi apu-oliolle

Metodin luonti public näkyvyysmääreellä rivillä 60 ei olisi ollut välttämätöntä. Ohjelman molemmat luokat kuuluvat samaan pakettiin, jolloin protected määre olisi ollut riittävä. Vaihtoehtoisesti näkyvyysmääreen olisi voinut jättää kokonaan pois, jolloin Java olisi käsitellyt metodia oletusarvoisesti package protectedina. Tässä kumminkin ennakoitiin tulevaa. Myöhemmin tehtävän tietokantaintegraa-

tion toteuttavat luokat sijaitsevat eri paketissa. Tästä huolimatta myös sen on kyettävä noutamaan data.

Koska konfiguraatiolla on oltava voimassaoloaika, oli tarpeellista luoda menetelmä sen asetukseen. Kuvassa 10 on esiteltynä, miten Javan GregorianCalendar-luokkaa käyttäen saadaan luotua XML-aikaobjekti.

```
25      Calendar cal = new GregorianCalendar();
26      Date tetday = new Date();
36          cal.setTime(tetday);
37          cal.add(Calendar.YEAR, 1);
38          Format sdf = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss");
39          Date test = cal.getTime();
40          String cal2 = sdf.format(test);
41          XMLGregorianCalendar ttl = null;
42          try {
43              ttl = DatatypeFactory.newInstance().newXMLGregorianCalendar(cal2),
44          } catch (DatatypeConfigurationException e) {
45              e.printStackTrace();
46          }
```

Kuva 10. Päiväyksen asetus

Aluksi riveillä 25 ja 26 luodaan kalenteri- ja päivä-objektit, joita hyödyntämällä rivillä 36 asetetaan kalenterin ajaksi tämänhetkinen aika. Konfiguraation voimassaoloajaksi asetetaan vuosi rivillä 37. Koska gregoriaanisen kalenterin luokka ei tarjoa päiväystä XML:n vaatimalla tavalla, tarvitsee aikaleimalle muodostaa oikeanlainen esitystapa käyttäen SimpleDateFormat-metodia rivillä 38. Rivillä 39 Date-objektiin tallennetaan aiemmin luotu aikaleima, joka muutetaan rivillä 40 Stringiksi käyttäen vaadittua muotoilua. Rivillä 40 luodaan objekti ttl XML:n kalenteriformaatissa, jonka sisällöksi asetetaan rivillä 43 try-catch lohkon sisällä muotoiltu aikaleima.

Viimeinen vaihe vastauksen luonnissa on niputtaa tarvittavat tiedot apu-olioon, jotta sitä voidaan käyttää.

```
47          apu.setCfgAreaId("europe");
48          apu.setCsName("CsEuropeFi");
49          apu.setMaxFrequency(bd1);
50          apu.setMaxTransitPower(bd3);
51          apu.setMinFreqency(bd2);
52          apu.setSsPubKey("AAAAB3NzaC1yc2EAAAABIwAAAIEA1on8gxCGJJWSRT4uOrR13mUaUk0hRf4RzxSZ"
53                  + "1zRbYYFw8pfGesIFoEuVth4HKyF8k1y4mRUnYHP1XNMNMJl1JcEArC2asV8sHf6zSPVffozZ5TT4"
54                  + "SfsUu/iKy91UcCfXzwre4WWZSXXcPff+EHtWshahu3WzBdnGxm5Xoi89zcE=");
55          apu.setTimeToLive(ttl);
56          apu.setWsdId(uuid);
```

Kuva 11. Tietojen sijoitus apu-olioon

Kuvassa 11 on esitetty miten olioon asetetaan vaaditut arvot. Tässä käytetään pääosin staattisia arvoja, koska tarvittavia tietokantoja ei ollut työtä tehdessä. Riveillä 47 ja 48 asetetaan konfiguraatiolle alueid ja palvelimen nimi. Muuttuja kertoo millä alueella konfiguraatio on validi. Konfiguraatiopalvelimen nimi tarvitaan, jotta virhetilanteissa pystytään jäljittämään, miltä palvelimelta virheellinen data on peräisin. Riveillä 49-51 asetetaan arvot taajuuksille ja lähetysteholle. Arvot ovat desimaalilukuja jotka on asetettu Javan BigDecimal muuttujiin. Tässäkään ei käytetä Float-tyyppiä numeerisiin arvoihin aiemmin esittämäni pyöristysvirheen mahdollisuuden vuoksi.

Rivit 52-54 voitaisiin kirjoittaa yhdelle riville, mutta rivin pituus kasvaisi niin pitkäksi, että lukeminen hankaloituisi. Kyseisillä riveillä annetaan taajuuspalvelimen julkinen avain, jotta palvelin voi varmentaa taajuuspyynnön tulevan oikeasta paikasta. Seuraavaksi rivillä 55 asetetaan aiemmin luotu voimassaoloaika ttl-muuttujasta. Lopuksi annetaan rivillä 56 WSD-laitteen identifioiva UUID-tunnus, joka tuli pyynnön mukana pääluokalta.

4.4 Vastauksen allekirjoitus

Vastaus on sisällöltään valmis, mutta vielä tarvitsee allekirjoittaa se, jotta taajuuspalvelin tietää sen tulleen oikealta taholta.

```
108         Reply apurep = dt2.getApu();
109         PrintWriter writer = response.getWriter();
110         JAXBContext context;
111         try {
112             context = JAXBContext.newInstance(Reply.class);
113             Marshaller m = context.createMarshaller();
114             DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
115             dbf.setNamespaceAware(true);
116             DocumentBuilder db = dbf.newDocumentBuilder();
117             Document doc = db.newDocument();
118             m.marshal(apurep, doc);
119             writer.println(signaa(doc));
```

Kuva 12. Reply-luokan konvertointi ja allekirjoitus

Allekirjoitusta varten kuvassa 12 rivillä 108 noudetaan luvussa 4.3 luotu apu-olio allekirjoitusta varten. Rivillä 109 alustetaan vastauksen lähettämistä varten PrintWriter-olio writer, joka noudetaan response-luokasta. Seuraavaksi alustetaan JAXBContext-luokka. Aluksi luokka esitellään rivillä 110. Rivillä 112 try-catch lohkon sisällä kääntäjävirheiden välttämiseksi, siihen noudetaan Reply-luokan ominaisuudet. Luokasta noudetaan rivillä 113 Marshalleri m, jolla voidaan luoda Reply-luokasta XML-dokumentin vaatimat tunniste- ja datakenttien tiedot.

Rivillä 114 luodaan DocumentBuilderFactory-tehdasluokasta olio dbf, joka sisältää tarvittavat metodit XML-dokumentin rakentamiseen. Seuraavaksi rivillä 115 dbf pakotetaan seuraamaan nimiavaruuksia, koska luotava dokumentti sisältää niitä enemmän kuin yhden. Pakotuksella vältetään valmiiseen dokumenttiin mahdollisesti tulevat ongelmat, jos molemmilla nimiavaruuksilla on saman tunnisteen omaava tietojäsen. Mikäli dokumentilla on vain yksi nimiavaruus käytössä, edellisen rivin voi joko jättää pois tai korvata sen arvon false:a. Tehtaan säädön jälkeen siitä tuotetaan dokumentinkirjoittaja db rivillä 116. Lopulta rivillä 117 luodaan itse dokumentti, joka on vielä tyhjä. Rivillä 118 marshalleri m saa

parametreikseen apurep-olion tarjoaman tietosisällön, joka sijoitetaan dokumenttiin doc.

JAXB ei sisällä allekirjoitukseen tarvittavia metodeja, joten allekirjoitus tulee mahdolliseksi vasta kun luokasta on luotu dokumentti.

```
165         DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
166         dbf.setNamespaceAware(true);
167         KeyStore ks;
168         KeyPair kp = null;
169         try {
170             ks = KeyStore.getInstance("JKS", "SUN");
171             char[] pass = "changeit".toCharArray();
172             ks.load(new FileInputStream(
173                     "/home/amakela/resttest/apu/mykeystore.jks"), pass);
174             kp = getPrivateKey(ks, "mycert", pass);
```

Kuva 13. Avainvaraston KeyStore käyttö

Dokumentin allekirjoitusta varten tarvitaan yksityinen avain. Avaimia ei säilytetä sellaisenaan vaan tarvittaessa luodaan sertifikaatin avulla. Sertifikaatit säilytetään suojatuissa varastoissa, joista tässä käytän Javakeystore-formaattia. Rivillä 173 kuvassa 13 esitetään polku mykeystore.jks tiedostolle, johon on tallennettu työtä varten luotu sertifikaatti. Avainten tietojen siirto ohjelmalle alkaa rivillä 167, jossa alustetaan sisäinen keystore ks. Seuraavalla rivillä tehdään sama avainparille. Try-catch-lohkossa rivillä 170 objektille ks kerrotaan, että avainvarasto on Sunin määritysten mukainen ja tyypiltään jks. Sertifikaattivarastoilla tulee aina olla salasana, jotta kuka tahansa ei pääse niihin käsiksi. Salasana pass luodaan rivillä 171 muuttamalla changeit Stringistä chararrayksi. Muutos on tarpeen, sillä metodit, jotka salasanaa käyttävät vaativat sen olevan kyseisen muotoinen.

Rivillä 172 keystoreen ladataan tiedostojärjestelmästä avainsäiliö antamalla tiedostosijainti ja salasana. Absoluuttisen polun käyttö voi johtaa ongelmiin. Mikäli palvelimella, jolle ohjelma asennetaan, luettava tiedosto ei sijaitse tarkalleen saman polun päässä on seurauksena virhetilanne eikä ohjelman suorittaminen jatku. Koodia on siis muistettava muokata ennen käännöstä vaihtamalla sinne kohdejärjestelmän polku. Ongelma voidaan välttää käyttämällä property-tiedostoa, johon tiedostosijainti voidaan päivittää ilman koodin muokkaamista.

Yksityinen avain noudetaan rivillä 173 metodilla getPrivateKey, jolle annetaan parametreiksi avainvarasto, halutun sertifikaatin alias ja avainvaraston salasana.

```
208    private KeyPair getPrivateKey(KeyStore keystore, String alias,
209            char[] password) {
210        try {
211            Key key = keystore.getKey(alias, password);
212            if (key instanceof PrivateKey) {
213                java.security.cert.Certificate cert = keystore
214                        .getCertificate(alias);
215                PublicKey publicKey = cert.getPublicKey();
216                return new KeyPair(publicKey, (PrivateKey) key);
```

Kuva 14. Yksityisen avaimen nouto

Kuvassa 14 rivillä 211 haetaan avainvarastosta avain, joka täsmää tarjottuun alias ja salasanayhdistelmään. Seuraavalla rivillä IF-lausekkeessa tarkistetaan onko saatu avain tyyppiä yksityinen avain. Mikäli se ei ole niin metodi palauttaa tyhjän arvon. Avaimen ollessa halutunlainen riveillä 213–214 noudetaan nimetty sertifikaatti avainvarastosta. Julkinen avain noudetaan sertifikaatista rivillä 215 käyttäen Javan Certificate-luokan metodia getPublicKey. Lopuksi avainpari luodaan ja palautetaan kutsujalle viimeisellä rivillä.

```
181        DOMSignContext dsg = new DOMSignContext(kp.getPrivate(),
182                koje.getDocumentElement());
183        XMLSignatureFactory fak = XMLSignatureFactory.getInstance("DOM");
184        Reference ref = fak.newReference("", fak.newDigestMethod(
185                DigestMethod.SHA1, null), Collections.singletonList(fak
186                .newTransform(Transform.ENVELOPED,
187                        (TransformParameterSpec) null)), null, null);
188        SignedInfo si = fak
189                .newSignedInfo(fak.newCanonicalizationMethod(
190                        CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
191                        (C14NMethodParameterSpec) null), fak
192                        .newSignatureMethod(SignatureMethod.RSA_SHA1, null),
193                        Collections.singletonList(ref));
194        dsg.setDefaultNamespacePrefix("ns2");
```

Kuva 15. Allekirjoituksen valmistelu

Allekirjoituksen valmistelu alkaa tarjoamalla DOMSignContext-luokalle allekirjoitukseen käytettävä avain, joka noudetaan äsken luodusta avainparista getPrivate-metodilla kuvan 15 rivillä 181, ja luodun koje dokumentin juurielementti rivillä 182. Seuraavaksi luodaan XMLSignatureFactory luokasta olio fak, jolle kerro-

taan luotavan allekirjoituksen käyttävän Document Object Model-mallia rivillä 183. Riveillä 184–187 lisätään allekirjoituksen referenssiosioon sen käyttämä algoritmi, joka tässä on SHA1, sekä tieto siitä, että allekirjoitusta itseään ei käytetä varmistuksen laskemiseen. SignedInfo-luokalle ilmoitetaan käytettävä kanonisointimetodi, joka on inklusiivinen eli allekirjoitus liitetään osaksi dokumenttia. Luokalle kerrotaan myös allekirjoitusmetodin olevan RSA ja käytettävä salausalgoritmi. Sille annetaan myös aiemmin luotu referenssi. Lopuksi rivillä 194 kerrotaan oliolle dsg, että allekirjoituksen nimiavavaruuden nimenä käytetään ns2:a

Seuraavaksi luodaan itse allekirjoitus sigi. Tätä varten kuvassa 16 rivillä 195 kutsutaan tehdasoliota fak, ja muodostetaan sillä allekirjoitus edellisessä kappaleessa luodusta SignedInfo luokan ilmentymästä si.

```
195         XMLSignature sigi = fak.newXMLSignature(si, null);
196         sigi.sign(dsg);
197         TransformerFactory tf = TransformerFactory.newInstance();
198         Transformer trans = tf.newTransformer();
199         trans.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
200         StringWriter writer2 = new StringWriter();
201         trans.transform(new DOMSource(koje), new StreamResult(writer2));
202         String reply = writer2.getBuffer().toString();
```

Kuva 16. Allekirjoitus

Rivillä 196 allekirjoitus lisätään dsg-olioon. Dokumentti on allekirjoitettu, mutta vielä oliomuodossa. Riveillä 197 ja 198 luodaan transformeri TransformerFactoryä hyödyntäen. Transformerille trans annetaan rivillä 199 määräys sisällyttää XML-standardin mukainen esittely luotavaan dokumenttiin. Stringwriter writer2 luodaan rivillä 200, jotta transformerilta saadaan helposti käsiteltävää dataa ulos. Rivillä 201 transformeri muodostaa koje-oliosta dokumentin, jonka se tulostaa suoratoistona oliolle writer2. Rivillä 202 writer2:n sisältö luetaan bufferista, ja muutetaan String-muotoisena muuttujaan reply.

Ennen vastauksen lähettämistä siitä tarvitsee poistaa ylimääräiset merkit.

```
203         reply.replaceAll("\r", "");
204         reply.replaceAll("\n", "");
205         return reply;
```

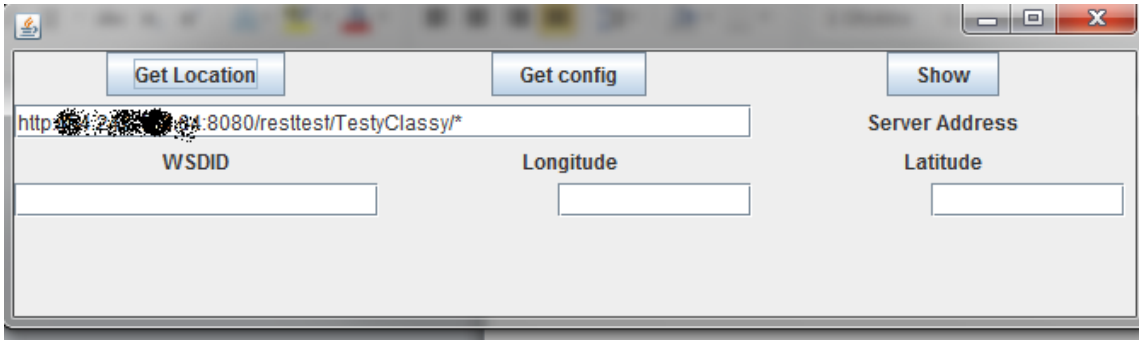
Kuva 17. Vastauksen viimeistely

Kuvassa 17 riveillä 203-204 korvataan kaikki muuttujasta reply löytyvät rivinvaihtomerkit tyhjällä. Tämä on tarpeellista, jotta dokumentin allekirjoitus säilyisi ehjänä. Mikäli vastaanottaja saa kyseisillä merkeillä varustetun vastauksen, se ei pysty vahvistamaan allekirjoitusta. Rivillä 205 valmis dokumentti palautetaan kutsujalle, joka on kuvan 12 rivillä 119 oleva response-luokasta noudettu tulostusolio.

5 ASIAKASOHJELMAN PROTOTYYPPI

Koska opinnäytetyön tekoaikaan ei ollut tarjolla varsinaista laitetta, oli perusteltua tehdä ohjelmallinen versio testausta varten. Teoriassa oikein toimivat metodit saattavat käyttäytyä aivan eri tavalla käytännössä. Asiakasohjelman prototyypin avulla palvelulle pystyy luomaan simuloidun toimintaympäristön. Tällöin vältetään heikosti tai väärin toimivan palvelun julkaisu. Tosin, jopa fyysisissä laitteissa dataa käsitellään ohjelmallisesti, jolloin prototyyppi ei poikkea niin sanotusta oikeasta ympäristöstä kuin toteutusalustansa osalta.

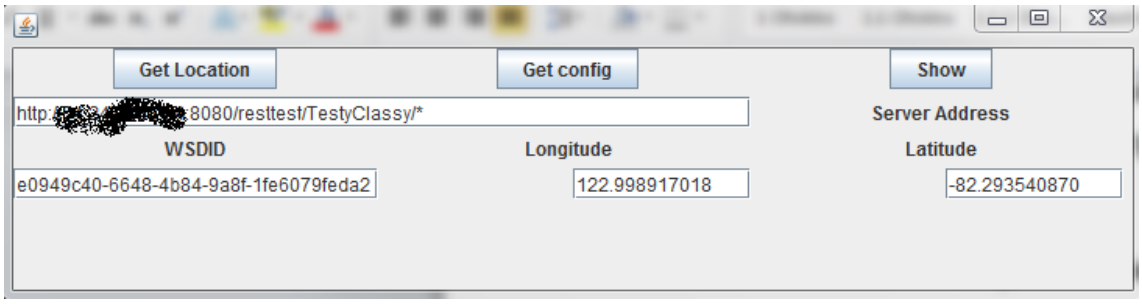
Asiakasohjelmalle tuli luoda mekanismit, joilla se pystyy lähettämään pyyntöjä ja ottamaan vastaan vastauksen sekä tutkimaan sen. Pyyntöjä pitää pystyä lähettämään sekä oikeilla että väärillä parametreilla. Väärien parametrien lähetysmahdollisuus auttaa tutkimaan, miten palvelu käyttäytyy virhetilanteissa. Asiakkaan tuli myös esittää saamansa vastaus ja tarkistaa allekirjoituksen paikkansapitävyys. Lisäksi asiakasta piti pystyä käyttämään eri alustoilla, jolloin Java oli pätevä vaihtoehto toteutusta varten.



Kuva 18. Asiakasohjelman käyttöliittymä

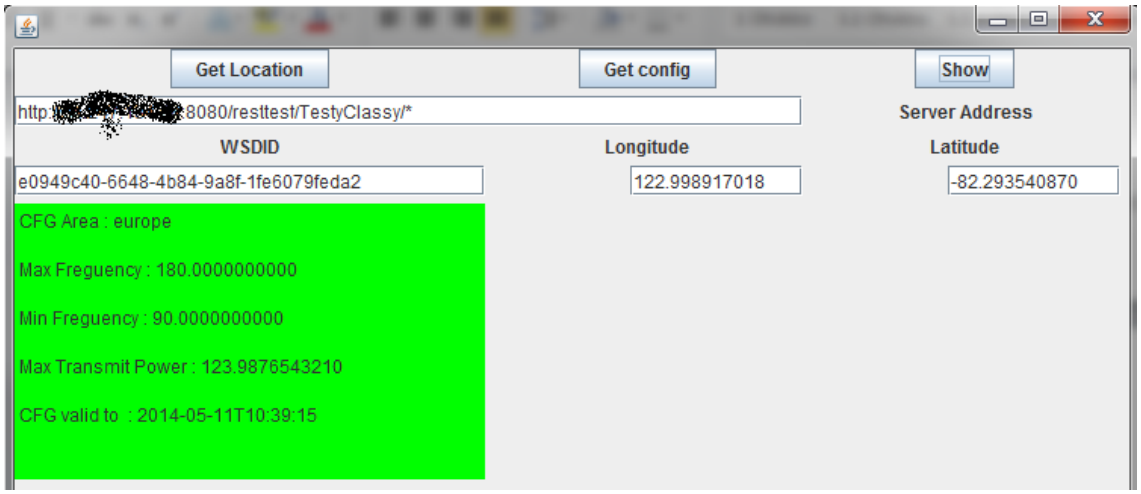
Kuten kuvasta 18 näkee, asiakkaan ulkonäkö jäi karkeaksi, koska sillä ei ollut sinänsä merkitystä. Pääasia oli, että se toimisi siinä roolissa, johon se tehtiin. Käytön helpottamiseksi asiakkaassa on mahdollisuus generoida pyyntöön tarvittavat arvot, samoin kuin mahdollisuus syöttää ne käsin. Generointi suoritetaan painamalla nappia Get Location, joka täyttää vaaditut arvot asianmukaisiin kent-

tiin. Kuvassa 19 kentät on täytetty generoimalla, mutta tämä onnistuu myös manuaalisesti syöttämällä.



Kuva 19. Generoinnin tulokset

Konfiguraatio noudetaan painonapilla Get config, ja tulokset saa näkyviin napilla Show.



Kuva 20. Tulokset näkyvissä

Kuvassa 20 on tehty kaikki edeltävät toimet. Wsdid:n syöttökentän alla on vihreä tausta, joka merkitsee, että allekirjoitus pystyttiin tarkistamaan. UI näyttää joitakin lisäarvoja, joista pystyy tarkistamaan niiden olevan noudettu palvelimelta. Mikäli allekirjoituksen tarkistus ei olisi onnistunut, tekstikentän tausta olisi punainen.

Jotta konfiguraatio voidaan noutaa, on käyttöliittymään generoidut tai kirjoitetut tiedot välitettävä palvelimelle.

```
16    public GetXml(String id, String la, String lo, String uri) {
17        if (uri.isEmpty()) {
18            uri ="http:/[redacted]:8080/resttest/TestyClassy/*";
19        }
20        String query = "?id=" + id + "&la=" + la + "&lo=" + lo + "&opt=k";
```

Kuva 21. Pyynnön muodostus

Kuvassa 21 on esitetty pyynnön muodostus. Get config-napin painallus lukee arvot tekstikentistä ja lähettää ne GetXml-metodille. Riveillä 17-19 tarkastetaan onko mukaan toimitettu palvelimen osoite. Mikäli osoitetta ei ole, arvoksi asetetaan testipalvelimen osoite. Rivillä 20 muodostetaan itse kysely omaksi String-muotoiseksi muuttujaksi.

Ennen lähetystä osoite jaetaan osiin. Kuvassa 22 riveillä 22-27 osoitteesta eritellään palvelin, tiedostosijainti ja portti. Mikäli porttia ei ole osoitteessa mainittu, käytetään http-liikenteen oletusporttia 80.

```
22            URL url = new URL(uri);
23            String host = url.getHost();
24            String filename = url.getFile();
25            int port = url.getPort();
26            if (url.getPort() == -1)
27                port = 80;
28            Socket soc = new Socket(host, port);
29            InputStream whatiget = soc.getInputStream();
30            PrintWriter to_host = new PrintWriter(new OutputStreamWriter(
31                    soc.getOutputStream()));
32            to_host.print("GET " + filename + query + "\r\n\r\n");
33            to_host.flush();
```

Kuva 22. Pyynnön lähetys

Rivillä 28 luodaan socket soc, joka tarvitsee palvelimen osoitteen ja portin. Näiden tietojen avulla voidaan lähettää ja vastaanottaa tietoa. Rivillä 29 socketista luodaan kuuntelija, joka vastaanottaa vastauksen. Kuuntelija on luotava ennen kyselyn lähettämistä, koska muuten vastaus katoaa. Rivillä 30 valmistellaan pyynnön lähetys luomalla tulostaja to_host. Tulostajalle kerrotaan rivillä 32, että

kyselyn metodi on GET, tiedostosijainti ja itse kysely. Kyselyn lähetyksen jälkeen rivillä 33 lähetyspuskuri tyhjennetään.

Vastaus on saapuessaan tallennettu InputStream tyyppiseen muuttujaan whatiget. Kyseisessä muodossa sitä ei pystytä hyödyntämään, vaan se on muutettava sopivampaan muotoon. Muunnos esitetään kuvassa 23. Rivillä 53 käytetään Javan Scanner-luokan metodia useDelimemiter. Säännöllisen lausekkeen \\A tarkoitus on aloittaa vastauksen lukeminen alusta. Metodi testing siis lukee koko streamin ja palauttaa sen kutsujalle String-muotoisena.

```
51    private String testing(java.io.InputStream is) {
52        try {
53            return new java.util.Scanner(is).useDelimiter("\\A").next();
```

Kuva 23. Vastauksen luku

Edellisessä kappaleessa mainittu metodin kutsu suoritetaan kuvassa 24 rivillä 34.

```
34              xml = testing(whatiget);
35              xml = xml.replace("\n", "");
36              xml = xml.replace("\r", "");
37              new ValidateSign(xml);
38              soc.close();
```

Kuva 24. Vastauksen käsittely

Riveillä 35-36 suoritetaan rivinvaihtojen poisto kuten kuvattiin luvun 4 lopussa. Vastauksena saatu XML-tiedosto lähetetään rivillä 37 allekirjoituksen validoivalle luokalle. Lopuksi rivillä 38 aiemmin luotu socketti suljetaan, jotta se ei olisi kuluttamassa resursseja turhaan.

Allekirjoituksen validointia varten tarvitaan sen muodostaneen yksityisen avaimen julkinen vastapari.

```
34    public ValidateSign(String file) {
35        try {
36            Document doc;
37            InputStream certfile = ClassLoader.getSystemResourceAsStream("apu/open.cer");
38            CertificateFactory cf = CertificateFactory.getInstance("X.509");
39            X509Certificate cert = (X509Certificate) cf.generateCertificate(certfile);
40            certfile.close();
```

Kuva 25. Sertifikaatin nouto

Asiakassovellusta tulee voida suorittaa useammalla koneella, minkä takia sertifikaatti on tallennettu samaan pakettiin kuin itse ohjelma. Tämän takia kuvassa 25 rivillä 37 oleva lukukomento ei sisällä absoluuttista polkua, vaan viittaa paketin sisäiseen hakemistopuuhun, toisin kuin palvelinsovelluksessa. Sertifikaatti tallennetaan cert-nimiseen muuttujaan rivillä 39. Tämän jälkeen luotu lukija suljetaan.

Kun sertifikaatti on saatu luettua, siitä noudetaan julkinen avain kuvan 26 rivillä 41.

```
41            pub = (RSAPublicKey) cert.getPublicKey();
42            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
43            dbf.setNamespaceAware(true);
44            doc = dbf.newDocumentBuilder().parse(
45                    new InputSource(new StringReader(file)));
46            NodeList nl = doc.getElementsByTagNameNS(XMLSignature.XMLNS,
47                    "Signature");
48            if (nl.getLength() == 0) {
49                throw new Exception("Cannot find Signature element");
50            }
51            XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
52            XMLValidateContext valCont = new DOMValidateContext(pub, nl.item(0));
53            valCont.setProperty("javax.xml.crypto.dsig.cacheReference",
54                    Boolean.TRUE);
55            XMLSignature sigi = fac.unmarshalXMLSignature(valCont);
56            coreValid = sigi.validate(valCont);
57            new ShowInfo(doc, coreValid);
```

Kuva 26. Validointi

Dokumentista etsitään allekirjoituselementti rivillä 46. Mikäli kyseistä elementtiä ei löydy, ilmoittaa sovellus virheestä rivillä 49. Rivillä 52 muodostetaan validointiin tarvittava konteksti julkisesta avaimesta ja dokumentin allekirjoituselementistä. Seuraavalla rivillä kontekstin ominaisuuksia muokataan muuttamalla ja-

vax.xml.crypto.dsig.cacheReference arvoon TRUE. Rivillä 55 allekirjoitus luetaan muuttujaan sigi. Itse tarkistus suoritetaan rivillä 56 ja sen tulos tallennetaan muuttujaan coreValid. Validoinnin tulos ja dokumentti lähetetään ShowInfo luokalle.

Dokumentin lukuun käytetään JAXBia, samoin kuin sen luontiin käytettiin palvelimella. ShowInfo-luokassa kuvassa 27 valmistellaan dokumentti rivillä 17 lukemalla se Reply-luokasta luotuun olioon rep.

```
12 public class ShowInfo {
13     public ShowInfo(Document dok, boolean valid) throws JAXBException {
14         Reply rep;
15         JAXBContext co;
16         co = JAXBContext.newInstance(Reply.class);
17         rep = (Reply) co.createUnmarshaller().unmarshal(dok);
18          tre = ("CFG Area : "+rep.getCfgAreaId()+ "\n\r"+
19                 "Max Freguency : "+rep.getMaxFrequency()+ "\n\r"+
20                 "Min Freguency : "+rep.getMinFreqency()+ "\n\r"+
21                 "Max Transmit Power : "+rep.getMaxTransitPower()+ "\n\r"+
22                 "CFG valid to  : "+rep.getTimeToLive()+ "\n\r"
23                 );
24     }
25     static String tre="Please get config first";
26     public static String getTre() {
27         return tre;
```

Kuva 27. Dokumentin luku

Tietojen nouto oliosta rep menee lähes samoin kuin niiden asetus palvelinpäässä. Tiedettäessä mitä arvoa ollaan noutamassa, kutsutaan sen hakumetodia, kuten riveillä 18-22 on tehty. Rivillä 25 asetetaan oletusarvo stringille tre sellaisten tapausten varalta, että tulosta yritetään noutaa ennen kuin dokumentti on haettu.

6 POHDINTA

Palvelu on lopuksi yksinkertaisempi toteutukseltaan, kuin mitä oletin aloittaessa. Tarkoituksena oli luoda luokka toimintoa kohti, jolloin uusien toimintojen lisääminen voidaan hoitaa luomalla ja testaamalla uusia luokkia. Poikkeuksen tästä tekee pää servletti, johon oli pyyntöjen kuuntelijan ja vastaajan lisäksi implementoitava myös allekirjoittaja. Tällöin allekirjoitus tulee aina luotua täydelliseen dokumenttiin eivätkä tulevaisuudessa tehtävät muokkaukset itse sisältöön vaikuta allekirjoituksen luontiin.

Asiakasprototyyppi osoitti hyödyllisyytensä paljastaessaan, että javan XML-allekirjoitukseen ilmaantui ylimääräisiä rivinvaihtoja. Toimivaksi oletettu palvelu ei antanutkaan sellaista allekirjoitusta, jonka olisi pystynyt varmistamaan. Tämä johti pitkään vianmetsästykseen, jonka aikana sekä palvelua että asiakasta koodattiin uusiksi useampaan kertaan. Ja kuten yleensä, omia virheitä ei itse huomaa, vaan kyseisen ongelman löytämiseen tarvittiin ulkopuolista apua.

Projektin myötä tuli huomattua miten tärkeää on käyttää versionhallintaa. Ennen versionhallinnan käyttöönottoa projektin kansio onnistui korruptoitumaan, mistä johtuen menetettiin muutaman päivän kehitystyö. Menetys olisi helpolla vältetty, mikäli olisi huolehdittu varmuuskopioinnista tai käytetty versionhallintaa.

Vaatimusmäärittely ei alkuun ollut kovin tarkka. Vaikka olisikin mukavaa tehdä asioita vapaasti, tulisi ennen työhön ryhtymistä olla riittävästi tietoa kaikista työhön liittyvistä alueista. Tämä huomattiin ja projektin edetessä vaatimukset tarkentuivat ja joiltain osin menivät uusiksi. Vaikeuksista huolimatta voi sanoa, että tavoitteet saavutettiin. Mikäli projekti olisi sujunut ilman ongelmia, sen aikana olisi opittu paljon vähemmän.

Omat työtavat kävivät tutuiksi projektin aikana yllättäen itsenikin. Välillä saattoi mennä viikko siihen, että etsi tietoa kohdattuihin ongelmiin. Ongelman ratkettua työpäivät venyivät yleensä todella pitkiksi. Jatkossa tarvitsee kiinnittää enem-

män huomiota siihen, että saa riittävästi lepoa ja kiinnittää huomiota muuhunkin kuin työhön.

LÄHTEET

Apache Software Foundation 2013, Apache Tomcat. Viitattu 1.4.2013 http://tomcat.apache.org/

AccuRev 2013, Support Plans.Viitattu 2.4.2013 http://www.accurev.com/product-support-support-plans

BitKeepper 2013, Cost of ownership. Viitattu 2.4.2013 http://www.bitkeeper.com/Sales.Cost.html

DTD Tutorial 2013, Viitattu 2.4.2013 http://www.w3schools.com/dtd/default.asp#gsc.tab=0

FAQ Httpd Wiki 2013, Viitattu 2.4.2013 http://wiki.apache.org/httpd/FAQ#What_is_Apache.3F

Goldberg, D. 1991, What every computer scientist should know about floating-point arithmetic. ACM Computing Survey. Vol 23, No 1, Article 1, 5-48.

Introduction to XML 2013, Viitattu 26.8.2013 http://www.w3schools.com/xml/xml_whatis.asp

ISO/IEC 7498-1:1994, Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model

JBossWeb 2013, What is JBoss Web. Viitattu 1.4.2013 http://www.jboss.org/jbossweb

GlassFish>>Metro>>Project JAXB, 2013, JAXB Reference Implementation - Project Kenai. Viitattu 11.5.2013 https://jaxb.java.net/

Perforce 2013, How to Buy. Viitattu 2.4.2013 http://www.perforce.com/purchase/pricing-licensing

Quin, L. R. E. 2013. Schema – W3C. Viitattu 25.5.2013 http://www.w3.org/standards/xml/schema

Rfc4122, Viitattu 2.10.2012 http://www.ietf.org/rfc/rfc4122.txt

SCM Matrix 2013, Viitattu 14.2.2013 http://docs.codehaus.org/display/SCM/SCM+Matrix

Smart, J. F. 2012, Continuous delivery-with-maven. Viitattu 14.2.2013 http://www.slideshare.net/wakaleo/continuous-deliverywithmaven

Toponce, A. 2013, Password Attacks, Part I – The Brute Force Attack. Viitattu 18.4.2013 http://pthree.org/2013/04/16/password-attacks-part-i-the-brute-force-attack/

Trustwave 2013, Trustwave SSL – Support – How SSL Works. Viitattu 25.6.2013 https://ssl.trustwave.com/support/support-how-ssl-works.php

WISE 2013, Viitattu 1.3.2013 http://wise.turkuamk.fi