Sonera content gateway webservice-rajapinnan ohjelmointi

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittely

Hämeenlinna 11.6.2010

Esa Kukkamäki

HAMK
HÄMEEN AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

Tietojenkäsittely
Hämeenlinna

Työn nimi	Sonera content gateway webservice-rajapinnan ohjelmointi
Tekijä	Esa Kukkamäki
Ohjaava opettaja	Lasse Seppänen
Hyväksytty	_____._____.20_____
Hyväksyjä	

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Hämeenlinna
Tietojenkäsittely
Systeemityö

TIIVISTELMÄ

Tekijä Esa Kukkamäki **Vuosi** 2010

Työn nimi Sonera content gateway webservice-rajapinnan ohjelmointi

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli toteuttaa webservice–rajapinta, joka lähettää tekstiviestejä. Tätä palvelua on tarkoitus hyödyntää Logiasoftwaren järjestelmissä.

Tutkimusmenetelmänä oli kehitysprojekti. Tavoitteena oli toteuttaa webservice–rajapinta ja analysoida toteutuksessa ilmenneitä ongelmia ja toteutusta yleensä. Teoriaosuudessa käydään läpi SOAP–arkkitehtuuria sekä webservice-palveluihin yleisesti liittyviä asioita.

Työn aineistona käytettiin lähes pelkästään internet-lähteitä. Tämä osoittautui hyväksi tavaksi, koska kehitysprojektityyppisessä työssä helpoimmin ongelmiin ratkaisu löytyy etsimällä vastaukset internetin kautta kuin etsimällä jokaiseen ongelmakohtaan ratkaisu kirjalähteistä. Apuna ongelmaratkaisuun ja kerrontaan on käytetty myös runsaasti työn kautta saatua tietoa.

Opinnäytetyön tuloksena voidaan esittää toimiva rajapinta, joka lähettää tekstiviestejä. Rajapinnalle on määritelty useita raja-arvoja, jotka on käyty tarkemmin läpi toteutusta kuvaavassa kappaleessa.

Työssä käydään läpi myös lyhyesti palveluun tullut jatkokehitys. Tämä kuitenkin käydään läpi vain komponenttien sijoittelun kautta. Työn tarkoituksena on siis kuvata pääasiassa vain Sonera content gatewayn toteutukseen liittyviä asioita.

Avainsanat **SOAP, Webservice, http**

Sivut 30 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

Business Information Technology
Software Engineering

Author Esa Kukkamäki **Year** 2010

Subject of Bachelor's thesis Sonera content gateway webservice-interface developing

ABSTRACT

The main purpose of this thesis was to implement to implement a web service interface, which sends SMS messages (Short Message Service). The aim was to utilize this interface in the systems of Logia Software Oy in the future.

The method of this study was a developing project. The aim of the study was to implement a web service interface, analyze problems that occurred in the course of the implementation and implementation in general. The theoretic part of the thesis focused on SOAP architecture and web services on the general level.

Background information was collected entirely from the Internet. This proved to be a good way, because this thesis is a development project and solutions are easier to find on the Internet than using literary sources. In addition, the author's own competency gained from work experience has contributed to the problem solutions.

The result of this thesis was a working interface, the purpose of which is to send SMS messages. There are several boundary values specified to the interface, which are explained in more detail in the implementation part of the thesis.

This thesis also describes briefly an additional development to this interface. The main focus of this additional development project is on the main components. The aim of this work was, however, to describe the main points of how to implement the interface using the Sonera content gateway service.

Keywords **SOAP, web service, http**

Pages 30 p.

SISÄLLYS

1 JOHDANTO 1

2 MITÄ ON SOAP? 2

2.1 SOA-arkkitehtuuri 2
2.2 SOAP 3
2.2.1 SOAP ja HTTP 3
2.2.2 SOAP – Envelopen rakenne 4
2.2.3 SOAP:in nimiavaruudet 4
2.2.4 SOAP – sanoman attribuutit 5

3 WEBSERVICE-PALVELUIHIN LÄHEISESTI LIITTYVÄT TEKNIIKAT 6

3.1 XML 6
3.2 XML SCHEMA 9
3.3 LOG4NET 11
3.4 HTTP 13
3.5 UDDI 14

4 PROJEKTIN ETENEMINEN VAIHEITTAIN 15

4.1 LUONNOSVAIHE JA IDEOINTI 15
4.2 TOIMINTA SONERA CONTENT GATEWAYN KANNALTA 16
4.3 WEBSERVICE RAJAPINNAN TOTEUTUS 18
4.4 JATKOKEHITYS SMS-PROXYSSÄ 19

5 ONGELMAT 21

5.1 MERKISTÖ 21
5.2 REVERSE PROXY 22
5.3 VALIDOINTI 24
5.4 TESTAUS 25

6 JOHTOPÄÄTÖKSET 26

6.1 OMA NÄKÖKULMA 26
6.2 *YRITYKSEN NÄKÖKULMA* 29

LÄHTEET 30

JOHDANTO

Tämä työ on toteutettu Logiasoftwaren tarpeita silmälläpitäen. Logiasoftware on vuonna 1975 perustettu yritys, joka on toiminut siitä lähtien logistiika alan merkittävänä ohjelmistotalona. Keväällä 2008 kävi ilmi, että heillä olisi tarve saada palvelurajapinta, joka huolehtisi tekstiviestien lähettämisestä. Tämä työ on tehty rajapinnan ohjelmoinnin ja eri tekniikoiden näkökulmasta.

Ensimmäisessä vaiheessa tarve oli yrityksen puolelta saada palvelu toimimaan kehitteillä olevassa ohjelmistossa, jonka nimi oli Partnertrack. Tähän tarvittiin yhdensuuntainen liikenne, mutta huomioon oli otettava jo tässä vaiheessa mahdolliset jatkokehityspiirteet. Tiedossa oli, että myöhemmin tarve kasvaa saada tieto tekstiviestin läpimenosta reaaliaikaisesti. Tässä kohtaa toteutettiin kuitenkin vain viestien lähettäminen asiakkaalle.

Siihen kuinka eri järjestelmät integroituvat SMS–proxy palveluun ei työssä sen tarkemmin oteta kantaa. Ainoastaan millaisiin tarpeisiin palvelua käytetään, käydään työssä pintapuolisesti lävitse. Ensimmäisenä tekstiviestin lähetyspalvelu otettiin käyttöön PartnerTrack nimisessä sovelluksessa. PartnerTrack hyödyntää SMS–proxyä Viron R-kioskien pakettiliikennettä seuratessa. Kun asiakkaalle saapuu paketti Viron noutopisteelle, lähetetään tästä tekstiviesti asiakkaalle. Asiakas tietää näin, että hänen tilaamansa paketti on saapanut ja noudettavissa.

Toisen vaiheen toteutus liittyy SSI-palveluun, joka on aiheuttanut muutoksia myös SMS–proxyyn. SSI tulee sanoista sähköinen saapumisilmoitus. Näihin jatkokehityspiirteisiin ei työssä kuitenkaan sen tarkemmin paneuduta. Pääpaino on Sonera Content Gateway – palvelussa.

Opinnäytetyössä käydään läpi tarkasti webservice-palvelurajapinta sekä SOAP–arkkitehtuurin mahdollisuudet. Työn toteutuksesta kertovassa osuudessa tutustutaan myös Content gatewayn tarjoamiin ohjelmointimahdollisuuksiin.

SOAP –arkkitehtuurin ymmärtämisellä on tärkeä merkitys työn onnistumisen kannalta. Tämän vuoksi tätä käsitellään työssä laajasti. Webservice –palvelu hyödyntää SOAP sanomia tiedonsiirrossa. Sanomat siirretään SOAP: n avulla eri rajapintaa hyödyntäviltä järjestelmiltä Soneralle, josta viesti sitten fyysisesti lähetetään. Vastauksena SMS –proxy palvelu palauttaa tiedon siitä onnistuiko lähetys vai ei.

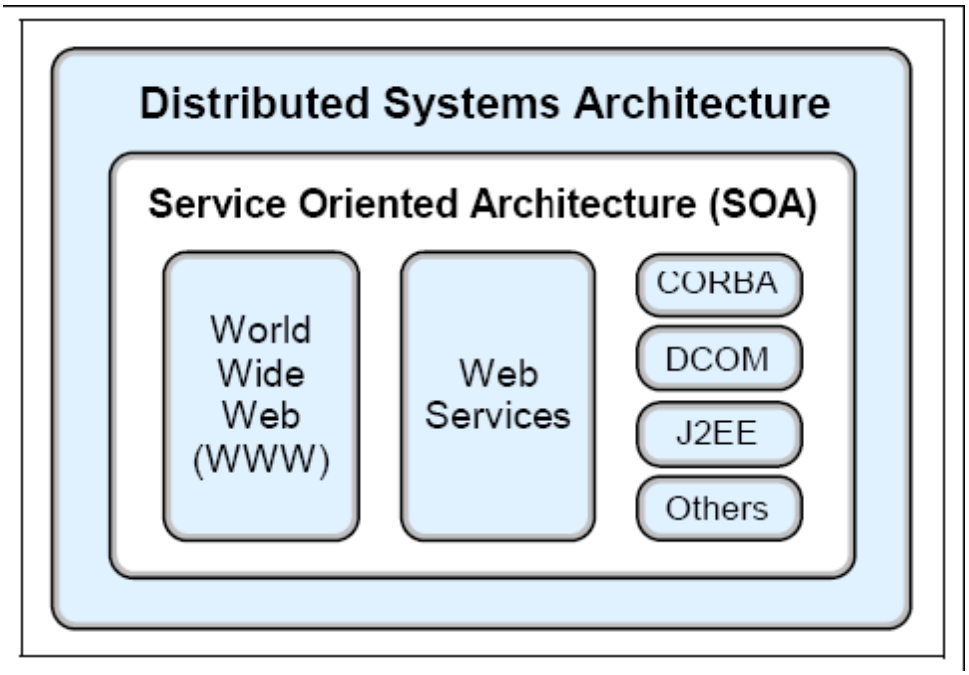
Työn ongelmia käsitellään omassa luvussaan työn loppupuolella.Näitä ovat erityisesti kyrilliset kirjaimet sekä SOAP:n ja web service –palveluiden kokonaiskuvan ymmärtäminen.

MITÄ ON SOAP?

Tässä luvussa käsitellään SOAP:in perustoimintaperiaate, jota on hyödynnetty SMS-proxy rajapinnan toteutuksessa. Samalla käydään läpi pääperiaatteet HTTP kutsuista sekä WSDL-kuvauskielen toimintaperiaate.

SOA-arkkitehtuuri

SOA(Service Oriented Architecture) eli palvelukeskeinen arkkitehtuuri on arkkitehtuurikehys, jossa eri prosessit on suunniteltu toimimaan itsenäisinä palveluina. Näitä palveluita on sitten mahdollisuus hyödyntää standardien rajapintojen avulla. Tarkoituksena on saada aikaan alustariippumaton vuorovaikutus kahden eri järjestelmän välillä. SOA siis määrittää Web-palveluiden käyttöä ja kahden järjestelmän välisiä standardeja. Web palveluiden käyttäminen ei kuitenkaan itsessään välttämättä vaadi SOA-palveluiden käyttämistä. SOA:n toteuttamiseen on muitakin vaihtoehtoja kuin web-palvelut. Yleisin tapa Internet – maailmassa on kuitenkin hyödyntää Web-palveluita sekä SOA-arkkitehtuuria. Internetissä suurin ongelma on loppujenlopuksi nopeuden ja toimintavarmuuden takaaminen. Usein ei tiedetä, millä alustoilla eri sovellukset toimivat ja tämä ongelma yleensä ratkaistaan hyödyntämällä SOA-arkkitehtuuria sekä webservice-rajapintoja. Kommunikaatiosta kahden eri järjestelmän välillä saadaan alustariippumatonta, kun kahden eri sovelluksen väliin toteutetaan web-service-rajapinta ja sille määritellään tietyt standardit, jolloin kommunikointi saadaan toimimaan riippumatta järjestelmien erilaisuuksista. Tällä aikaansaadaan se, että järjestelmät voidaan olla toteutettu täysin eri ohjelmointikielillä tai olla fyysisesti eri paikassa. webservice-rajapintoja on mahdollista toteuttaa useilla eri ohjelmistokielillä, kuten .net, php tai java ympäristöissä. Peruslähtökohtana pitäisi kuitenkin olla järjestelmäriippumattomuus eli riippumatta ohjelmointikielestä pitäisi rajapinta olla kuvattu niin, että sitä on mahdollista kutsua miltä tahansa alustalta. Suurin osa ohjelmointitavoista nykyään generoi rajapintakuvauksen eli wsdl-kuvausdokumentin automaattisesti. Tällöin se on useimmiten täysin toimiva myös muiden ohjelmointikielien kanssa. Webservice asioihin palataan vielä myöhemmin luvussa 4. Kuvassa 1 on esitelty jaettujen järjestelmien arkkitehtuuria. Vanhempaa tekniikkaa edustavat Corba sekä DCOM, joita on käytetty järjestelmien välisissä integraatioissa jo vuosikaudet. Näiden ongelma on, että ne sopivat huonosti hyödynnyttäviksi uusissa projekteissa, koska yleinen standardi kuinka rajapinnat liitetään toisiinsa on puuttunut. (Endrei 2004.)



Kuva 1 SOA:n mallikuva[Endrei 2004.]

SOA koostuu kahdenlaisista elementeistä:

SOAP

SOAP-termi on tiivistys sanoista Simple Object Access Protocol. Se on tiedonsiirtoprotokolla sovellusten väliseen kommunikointiin.
SOAP perustuu XML kuvauskieleen ja koostuu kolmesta eri osasta: kirjekuoresta(Envelope), koodaussäännöistä(encoding rules) sekä etäkutsujen merkintäkäytänteistä.
SOAP-sanomat pakataan XML-dokumenttiin, jota kutsutaan suomennettuna kirjekuoreksi. Yksi kirjekuori lohko sisältää aina yhden SOAP-sanoman. Kirjakuori on kuvaa osuvasti sen merkitystä, koska sen sisään on tarkoitus asettaa kaikki tarpeellinen, mitä SOAP-sanoman operaatio vaatii. Vastaanottaja ottaa kirjeen vastaan muuntaa sen käsiteltävään muotoon ja toteuttaa operaation, mihin palvelu on tarkoitettu. Sanoma jota kahden etäisen järjestelmän välillä on toteutettu sellaiseen muotoon, että sillä ei ole väliä onko vastaanottajalla ja lähettäjällä sama ohjelmointikieli. Kummallakin osapuolella sekä lähettäjällä, että vastaanottajalla on tieto siitä, mitä elementtejä kirjekuori sisältää, mutta ne ovat sellaisessa muodossa, että sanoma on purettavissa lähes mille tahansa ohjelmointikielelle.

SOAP ja HTTP

SOAP:in tarkoituksena on hyödyntää kahden eri järjestelmän tietosisältöä. Tiedonsiirtokanavana toimii useimmiten HTTP arkkitehtuuri. Muitakin tiedonsiirto –protokollia kuten FTP, SMTP tai JMS on mahdollisuus käyttää, mutta nämä ovat niin harvinaisia, että tässä keskitytään vain HTTP-protokollaan. Protokollalla tarkoitetaan standardia, jolla määritellään lait-

teiden tai ohjelmien välisiä yhteyksiä ja tässä tapauksessa puhutaan ohjelmien välisistä yhteyksistä. HTTP – protokolla käsittelee kahden suuntaista liikennettä. Yksinkertainen esimerkki on WWW–sivu, jota käytetään selaimen välityksellä. Selain lähettää HTTP–protokollan avulla pyynnön palvelimelle, joka sitten vastaa pyyntöön lähettämällä pyyntöä vastaavaa dataa takaisin selaimelle.

SOAP arkkitehtuuri siis hyödyntää tätä HTTP – protokollan perusperiaatetta. SOAP – sanoma on kuvattu vain tarkasti siltä osin miltä pyyntö- ja vastaussanoma näyttävät. Jos kutsuvan järjestelmän pyyntösanoma on väärä, palauttaa SOAP kutsuvalle järjestelmälle vastaussanomassa virheviestin. Virheviestit voi kuvata vastaussanomaan halutulla tavalla. Envelopen eli kirjekuoren sisällöstä kuitenkin edempänä.

SOAP – Envelopen rakenne

SOAP:in kirjekuori kuvaa vastaus- tai pyyntösanoman kokonaisuutta. Envelope on aina pyynnön ylin lohko, jonka alle sitten sanoma rakentuu. Kirjekuori voi sisältää Header-elementin ja jos sisältää niin silloin se tulee heti Envelope-lohkon jälkeen. Header kuvaa yleensä järjestelmä spesifistä informaatiota, kuten esimerkiksi tunnustautumiseen liittyvää tietoa.(http://www.w3schools.com/soap/soap_header.asp) Envelope-osio rakentuu melko samankaltaisesti kuin lähettäisi kirjeen toiselle ihmiselle. Header – tieto voidaan ajatella olevan vastanottajan osoitetieto, joka ei ole kirjeen sisältöä. Varsinainen kirje on sitten pyyntö- tai vastaussanoma. Viimeisenä elementtinä ennen itse vastausta tai pyyntöä tulee body – elementti, jonka sisään sisältö rakentuu. Header eli otsikkotieto voi sisältää kahden tyyppisiä attribuutteja käyttäjän itse määrittelemiä sovelluskohtaisia ja SOAP – sanoma sidonnaisia attribuutteja, joilla on erityinen merkitys käsiteltäessä sanomaa. Nämä attribuutit käsitellään hieman myöhemmin tarkemmin(W3C 2007b).

SOAP:in nimiavaruudet

XML nimiavaruuksien ymmärtäminen on tärkeä osa SOAP – sanomien ymmärtämisessä. Pääasiassa niiden tehtävä on estää XML elementtien ja attribuuttien välistä monitulkinnallisuutta. Toisinsanoen ne on tehty kuvaamaan sisällön tai elementin tarkoitusta. Esimerkiksi sama puhelinnumero voi olla käytössä useammalla eri henkilöllä verkko-operaattorista riippuen. Nämä numerot on mahdollista erottaa toisistaan suuntanumeroiden avulla. Suuntanumeroa vastaa XML – dokumentissa nimiavaruus johon viitataan. Nimiavaruuksiin viittaamalla elementit ja attribuutit voidaan erotella toisistaan yksiselitteisesti(W3C,2007b).

Nimiavaruus koostuu URL – osoitteesta ja siinä mahdollisesti sijaitsevasta XML – dokumentista. XML – dokumentti kuvaa kyseiseen nimiavaruuteen liittyvien attribuuttien ja elementtien dokumentaation. Nimiavaruuksien tärkein tehtävä ei ole kuitenkaan viitata XML – dokumenttiin vaan antaa yksiselitteinen tunniste elementille.

Nimiavaruuteen viittaamalla voidaan todentaa tietyn elementtien tai attribuuttien tietotyypin oikeanlaisuus. Jos annettu tietotyyppi ei läpäise tarkistusta, pitäisi sanomassa palautua käyttäjälle virheviesti. Tällä mallilla on mahdollista myös saavuttaa eri SOAP-versioiden yhteensopivuus(Tsenov2002.).

Käytännössä nimiavaruuteen viittaaminen tapahtuu asettamalla attribuutti xmlns: elementille. Viittaus kirjoitetaan kokonaisuudessaan seuraavasti:

```
1  <po:purchaseOrder orderDate="2003-09-22"
2  xmlns:po="http://www.schema.fi">
3  ...
4  </po:purchaseOrder>
```

Kuva 2 Nimiavaruuteen viittaaminen

Tässä esimerkissä siis on purchaseOrder -elementti, jonka yksilöivä tunniste on po ja elementin attribuuttina orderDate. Nimiavaruuteen on viitattu aloitustagillä xmlns. po kuvaa elementin tunnistetta ja loppuosa on nimiavaruuden URL – osoite. Eli tässä tapauksessa nimiavaruudesta www.schema .fi pitäisi löytyä kuvaus minkä tyyppistä ja missä muodossa tietoa tuolle elementille tai attribuutille pitää syöttää(http://www.informit.com/articles/article.aspx?p=169106&seqNum=2) Jos saman elementin sisällä halutaan viitata useampaan nimiavaruuteen erotetaan nimiavaruuksien tunnus/url yhdistelmät toisistaan tyhjällä merkillä(W3C 2007b).

SOAP – sanoman attribuutit

SOAP – sanoman otsikkotieto osioon on määritelty ennalta attribuutteja, jotka vaikuttavat sanoman käsittelyyn. Nämä attribuutit ovat kooditustyyliattribuutti, rooliattribuutti, otsikkolohkon ymmärtämisattribuutti ja eteenpäinsiirtoattribuutti.

Encodingstyle eli kooditustyyliattribuutti kertoo SOAP – sanomalle tarkemmin mitä säännöstöä noudatetaan kun sanoma muunnetaan dokumentin siirron ajaksi XML – muotoon ja siitä takaisin alkuperäiseen muotoon. Tälle on oma nimiavaruutensa osoitteessa(W3C 2007b).

http://schemas.xmlsoap.org/soap/encoding/

Järjestelmien kehittäjät voivat myös luoda omia serialisointi sääntöjä. Jokaiseen elementin serialisointiin on mahdollista vaikuttaa erikseen. Jos yhdelle elementille määrittelee koodaussäännöstöjä, se periytyy aina lapsielementille. Tämä on mahdollista välttää asettamalla lapsielementille seuraavanlainen nimiavaruus(Khoshafian):

http://www.w3.org/2003/05/soap-envelope/encoding/none

Rooliattribuutti kertoo otsikko – osiolle tiedon siitä pitääkö kyseinen lohko käsitellä vai ei. Jos rooliattribuuttia ei määritellä, ei osiota pysty käsittelemään muu noodi kuin viestin lopullinen vastaanottaja. Noodilla tai solmulla tarkoitetaan loogista operaatiota, joka käsittelee SOAP – sanomaa. Nämä noodit voivat lähettää, vastaanottaa viestejä, käsitellä sanomaa tai välittää eteenpäin viestin tietoja. Noodeja taas voivat olla viestin lähettäjä, viestin vastaanottaja, viestin alkuperäinen luoja, soap -viestin välittäjä tai lopullinen viestin saaja.(https://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=/com.ibm.cics.ts.webservices.doc/concepts/soap/dfhws_nodes.html)Vaihtoehtoja on kolme mitä rooliattribuutille on mahdollista määrittää:

http://www.w3.org/2003/05/soap-envelope/none
http://www.w3.org/2003/05/soap-envelope/next
http://www.w3.org/2003/05/soap-envelope/ultimateReceiver

Roolilla none otsikkotietoa ei käsittele mikään noodi suoranaisesti. Tätä roolia käytetään kuljettamaan tarpeellisia tietoja muille otsikkolohkon osioille.

Roolilla next kaikilla noodeilla pitäisi olla mahdollista käsitellä kyseisen lohkon sisältö. Jos otsikkolohkoille on määritelty viestin välittäjänoodeja ja rooliattribuutti next silloin kyseinen lohko käsitellään aina vain kerran, jonka jälkeen se poistetaan.

Ultimate receiver roolilla tarkoitetaan sitä, että vain viestin lopullinen vastaanottaja käsittelee lohkon. Tämä määritys on sama kuin roolia ei olisi määritelty ollenkaan.

Rooleja on mahdollista määritellä myös sovelluskohtaisesti. Tämä voidaan tehdä halutessa myös niin, että saman sovelluksen tai webservice-asiakasohjelmiston eri operaatiolla on omat roolinsa.(W3C 2007b.)

WEBSERVICE-PALVELUIHIN LÄHEISESTI LIITTYVÄT TEKNIIKAT

Tässä luvussa on tarkoitus käsitellä eri tekniikoita tarkemmin, jotka liittyvät läheisesti webservice -palveluiden ohjelmointiin.

XML

Xml tekniikkaa käytetään webservice -palveluiden kuvauskielenä ja myös tiedon siirto eri järjestelmien välillä tapahtuu xml:n avulla. Tämän vuoksi xml – tekniikan ymmärtäminen on välttämätöntä, jos haluaa toteuttaa webservice – rajapintoja.

Xml – merkintäkieli on tarkoitettu kuvaamaan asioita helposti ymmärrettävässä muodossa. Seuraavassa yleinen periaate miten xml-tiedostot määräytyvät.

```
<Yritys>
    <Nimi>Logiasoftware</Nimi>
    <Osoite>Åkerlundinkatu 3</Osoite>
    <Puhelinnumero>031112222</Puhelinnumero>
</Yritys>
```

Kuva 3 Xml-esimerkki

Yllä olevassa esimerkissä on yksinkertainen xml–tiedosto joka kuvaa yrityksen tietoja xml:n avulla. Päätason tieto on yritys ja sen alla on eroteltuna yrityksen tarkemmat tiedot. Xml–tiedostossa on aina oltava aloittava elementti, joka yllä olevassa esimerkissä on Yritys. Lopetusmerkintä alkaa merkillä </ ja loppuu merkkiin >. Toinen mahdollisuus kuvata jokin asia on suoraan, niin että se loppuu merkintään />. Tällöin kuvataan vain että xml–tiedostossa on mahdollista antaa kyseinen elementti, mutta sille ei anneta arvoa. Esimerkiksi yllä olevan esimerkin puhelinnumero kentän voisi esittää seuraavasti <Puhelinnumero/>. Tällöin xml–tiedosto kertoo puhelinnumero kentän olevan olemassa mutta sille ei ole asetettu arvoa. Tärkeää xml kuvauskielessä on, että jokainen lohko suljetaan aina lopuksi. Jos jotain tietosisältöä ei suljeta on tiedosto silloin epävalidi ja esimerkiksi sovellukset, jotka sitä yrittävät lukea eivät siinä onnistu.

Xml –tiedostolla on oltava aina siis päätason elementti. Tätä kutsutaan juurielementiksi. Esimerkki tapauksessa tämä olisi Yritystieto. Jokaista toisen elementin sisällä olevaa tietoa kutsutaan lapsielementiksi. Päätason elementin sisällä voi olla useampia saman tietosisällön omaavia lohkoja. Näitä kutsutaan sisarelementeiksi. Monimutkaistetaanpa hieman esimerkkiä ja tehdään esimerkki xml:stä useampaa yritystä kuvaava xml–dokumentti.

```
<Yritykset>
    <Yritys>
        <Nimi>Logiasoftware</Nimi>
        <Osoite>Åkerlundinkatu 3</Osoite>
        <Puh>031112222</Puh>
    </Yritys>
    <Yritys>
        <Nimi>Tietoenator</Nimi>
        <Osoite>Sibeliuksenkatu 5</Osoite>
        <Puh>011225222</Puh>
    </Yritys>
</Yritykset>
```

Kuva 4 Useampi yritys samassa xml-tiedostossa

Xml dokumenteilla siis saadaan nopeasti kuvattua yksinkertaisella tavalla melko monimutkaisiakin tietorakenteita. Tieto on ymmärrettävää, eikä oikeastaan aseta mitään rajoitteita tietosisällön määrälle. Ainoa ongelma on se, että elementtejä kuvataan useampaan kertaan ja tästä johtuen tiedostojen koko saattaa nopeasti kasvaa suureksi.

Kolmas merkityksellinen asia xml–tiedostojen ymmärtämisessä on attribuuttien käyttö. Attribuutilla tarkoitetaan xml tiedostossa elementin määräävän lohkon sisäistä määrittäjää. Esimerkkinä tästä seuraavanlainen xml –tiedosto.

```
1  <Yritykset>
2      <Yritys tyyppi=”siivousala”>
3          <Nimi>Sol</Nimi>
4          <Osoite>Kuninkaankatu 1</Osoite>
5          <Puh>07-4442222</Puh>
6      </Yritys>
7  </Yritykset>
```

Kuva 5 Attribuuttien käyttö xml-kielessä

Edellisen esimerkin yritys lohkon sisään on nyt määritelty Yrityksen tyyppi attribuutti määreellä. Attribuutilla siis on mahdollista määrittää lohkolle joku sitä kuvaavaa tekijä. Esimerkkitapauksessa on käytetty yrityksen määrittyjänä tyyppi attribuuttia, joka kertoo lukijalle minkä alan yritys on kysymyksessä.

Webservice-rajapinnoissa xml–muotoa käytetään pyyntö- ja vastaussanomissa kehystettynä soap–kehyksellä, josta on kerrottu tarkemmin 1 luvussa. Esimerkkinä pyyntösanoma Smsproxyn rajapinnasta

```
1  <soap:Envelope xmlns:xsi-"http://www.w3.org/2001/XMLSchema-instance"
2  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
4    <soap:Body>
5      <SendSMSMessage xmlns="http://logiasoftware.fi/SMSProxy/2008/">
6        <AuthId>string</AuthId>
7        <SenderNumber>string</SenderNumber>
8        <RecipientNumber>string</RecipientNumber>
9        <Message>string</Message>
10     </SendSMSMessage>
11   </soap:Body>
12 </soap:Envelope>
```

Kuva 6 Esimerkki Soap–pyynnöstä xml–kielen avulla

Periaate siis tässäkin tapauksessa on sama kuin aiemmissa esimerkeissä. Jokaiselle lohkolle on aloitus ja lopetus merkintänsä. Tässä juurielementtinä on Soap-kehyksen kirjekuori, mutta se mitä rajapinta ottaa vastaan on SendSMSMessagen sisällä. Tämä xml–dokumentti kuvaa vielä tämän lisäksi, minkä tyyppisiä arvoja kyseisiin kenttiin on mahdollista syöttää.

Vastaavasti Soap vastaussanomassa on sama logiikka. Smsproxyn tapauksessa vastaussanoma on vieläkin yksinkertaisempi xml–tiedosto kuin pyyntösanoma. Tämä on esitettynä alla olevassa kuvassa.

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SendSMSMessageResponse xmlns="http://logiasoftware.fi/SMSProxy/2008/">
      <SendSMSMessageResult>boolean</SendSMSMessageResult>
    </SendSMSMessageResponse>
  </soap:Body>
</soap:Envelope>
```

XML SCHEMA

Xml–tiedostojen käsittelyssä scheema–tiedostojen ymmärtäminen on lähes yhtä merkityksellistä kuin itse xml–kielenkin. Lähes mitä tahansa xml–muotoista tietoa järjestelmien välillä liikuteltaessa on se kuvattu scheema tiedostolla, minkälaisia arvoja tai tyyppejä yksittäinen elementti ottaa vastaan. Scheema–tiedostot ovat siis xml–dokumentin tietosisällön kuvaavia tiedostoja.

Xml-scheeman ymmärtäminen kuitenkin vaatii enemmän kuin pelkän xml–kielen tuntemuksen. Scheema–tiedostot sisältävät paljon ennalta määritettyä tietoa, kuten elementtien nimiä ja niille annettavia arvoja. Rakenne scheema-tiedostossa on kuitenkin vastaava kuin xml:ssä. Lohko alkaa merkinnällä <> ja loppuu merkintään </> tai joissain tapauksissa />.

Scheema-tiedostojen rakenne on helpoin selittää esimerkin kautta. Otetaan ensin yksinkertainen xml–tiedosto, joka kuvaa edelleen yrityksen tietoja. Tiedoston päätaso on Yritysrekisteri ja sen sisällä on yritys ja kaksi työntekijää.

```
<?xml version="1.0"?>
<Yritysrekisteri Lisäyspäivä="2010-03-03">
   <Yritys Osastontunnus="Es">
      <Nimi>Muro Oy</Nimi>
      <Osoite>Muropolku 3</Osoite>
      <Kaupunki>Helsinki</Kaupunki>
      <Postinumero>11000</Postinumero>
      <Henkilöt>
        <Työntekijä Sukupuoli="M">
            <Nimi>Teppo Salama</Nimi>
            <Työtehtävä>Ohjelmoija</Työtehtävä>
        </Työntekijä>
        <Työntekijä Sukupuoli="N">
            <Nimi>Mari Salanoja</Nimi>
            <Työtehtävä>Sihteeri</Työtehtävä>
        </Työntekijä>
      </Henkilöt>
   </Yritys>
   <comment>Tässä uuden osaston tiedot/comment>
</Yritysrekisteri>
```

Kuva 7 Scheema-esimerkki

Vastaavanlaista xml–tiedostoa voitaisiin käyttää esimerkiksi jossakin ohjelmistossa tallentamaan yrityksen tietoja. Tässä tapauksessa pitäisi asiakkaalle, joka yrittää tiedostoa järjestelmään tuoda luoda scheema-tiedosto, mitä tietoja sanoma tarvitsee, jotta se on tallentavan ohjelman kannalta oikeassa muodossa. Scheema-tiedosto voisi olla seuraavanlainen.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="YritysRekisteri" type="YritysRekisteriTyyppi"/>

  <xsd:complexType name="YritysRekisteriTyyppi">
    <xsd:sequence>
     <xsd:element name="Yritys"  type="YritysTyyppi"/>
     <xsd:element name="Kommentti" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="LisäysPäivä" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="YritysTyyppi">
    <xsd:sequence>
      <xsd:element name="Nimi"    type="xsd:string"/>
      <xsd:element name="Osoite" type="xsd:string"/>
      <xsd:element name="Kaupunki"    type="xsd:string"/>
      <xsd:element name="Postinumero"  type="xsd:integer"/>
      <xsd:element name="Henkilöt"  type="Henkilöt"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Henkilöt">
    <xsd:sequence>
      <xsd:element name="Työntekijä" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Nimi" type="xsd:string"/>
            <xsd:element name="Työtehtävä">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Sihteeri"/>
                  <xsd:enumeration value="Ohjelmoija"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="Sukupuoli">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="M"/>
                  <xsd:enumeration value="N"/>
                </xsd:restriction>
              </xsd:simpleType>
          </xsd:attribute>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Kuva 8 Scheema- esimerkki2

Edellinen kuvanmukainen esimerkki xml–schema tiedostosta kertoo toteuttajalle, mitä arvoja xml–dokumenttiin voi ottaa mukaan. Tiedoston alkuun eli schema elementin sisään on määriteltävä namespace, joka erottelee elementit toisistaan. Nimiavaruuksista eli namespace asioista on puhuttu tarkemmin ensimmäisessä kappaleessa. Esimerkissä ensimmäinen xsd:element tieto kertoo, että juurielemetti xml–tiedostossa on yritysrekisteri. Elementeille on mahdollista määritellä type–arvo, jonka arvoksi voi-

daan asettaa complextype lohkossa oleva arvo. Tässä tapauksessa siis siihen on asetettu yritysrekisterityyppi, joka kertoo että scheema tiedostosta on löydyttävä yritystyyppi nimellä complexType–lohko. Tämän complexType lohkon elementit taas määräytyvät Yritysrekisteri tiedon sisälle eli tämän kyseisen lohkon sisällä voivat olla kommentti elementti, attribuutti lisäyspäivä sekä elementti Yritys, jonka tiedot ovat complexTypen sisällä, joka pitäisi löytyä nimellä Henkilöt.

Samoilla tavoin edetään edelleen prosessissa eli Henkilöt complexTypen sisällä voi esiintyä Työntekijä. Arvo minoccurs ja maxOccurs kertoo sen kuinka monta kertaa kyseinen elementti voi xml–tiedostossa ilmetä. Tässä tapauksessa kyseinen lohko voi olla 0-n kertaa. Työntekijän sisällä taas on Nimi,Työtehtävä sekä attribuutti Sukupuoli. Sukupuolelle ja työtehtävälle on vielä erikseen määritelty rajaehdot, mitä arvoja niihin hyväksytään.(W3C,2004.)

Edellinen esimerkki antaa hyvän kuvan siitä, mikä on scheema-tiedostojen tarkoitus. Syntaksi on sopivan helppolukuista ja ylimääräisten dokumenttien kirjoitustyö vähenee huomattavasti, kun yksinkertaisella xsd–tiedostolla pystyy xml:n kuvaamaan melko tarkallakin tasolla.

Scheema–tiedostot ovat erittäin tärkeitä myös rajapintoja julkaistessa, koska näistä on lähes aina tehtävä scheema-tiedostot, jotka kuvaavat ainakin sen minkä muotoisia pyyntö xml–sanomia palvelu lukee sisään ja minkä muotoista tietoa se palauttaa. Näiden scheema–tiedostojen pohjalta kyseistä webservice-rajapintaa hyödyntävä ohjelmoija osaa tehdä pyynnön palvelulle ja lukea vastauksen omalle järjestelmälleen.

LOG4NET

Log4net on dot.net ohjelmointikielessä yleisesti käytetty kirjasto, jonka tarkoituksena on toimia apuvälineenä tiedostoon kirjoituksessa sekä sähköpostien lähettämisessä. Kirjaston ehdottomia hyötyjä on yksinkertainen konfigurointi sekä helppo laajennettavuus. Laajennettavuudesta hyvänä esimerkkinä on edellä mainittu sähköpostien lähetys mahdollisuus.

Log4netin konfigurointi tapahtuu web.config – tiedostossa tai sitten tälle voi tehdä oman konfigurointi tiedostonsa, mikä on usein järkevää. Web.config tiedostoon on usein sisällytetty paljon muutakin tietoa ohjelman paranmetreistä, joita joudutaan mahdollisesti muuttamaan aika-ajoin. Selkeyden vuoksi nämä olisikin järkevintä erotella omiksi tiedostoikseen. Tämä on mahdollista tehdä esimerkiksi seuraavalla tavalla.

Dot.net web–sovelluksissa on global.asax tiedosto, johon on mahdollista määrittää toimenpiteitä sessioiden alustamisesta, sovelluksen käynnistymistä, virhetilanteista sekä monia muita web–sovelluksessa tapahtuvia asioita.

```
Sub Application_start(byval sender as object, byval e as EventArgs)
        'koodi joka ajetaan kun ohjelma käynnistyy
        SMSProxy.SMSProxyApplication.FirstTimeInitialize(System.configuration.ConfigurationManager.AppSettings)
    End sub
```

Edellisessä esimerkissä ohjelman käynnistyessä suoritetaan application luokalta firsttimeInitialize–metodi. Tämä alustaa ohjelman alussa tapahtuvat asiat.

```
    Public shared sub FirstTimeInitialize(byval settings as System.Collection.Specialized.NameValueCollection)
            'alustetaan logger luokka
            Util.Logger.InitializeLogger(Globalsettings.LogConfigurationFile)
    End sub
```

FirstTImeInitialize–metodi kutsuu vielä logger luokkaa hakemana web.config–tiedostoon asetettua polkua vastaavaa logger-luokan konfigurointitiedostoa.

```
Public shared sub InitializeLogger(byval log4netConfigurationFileName as String)
        'hakee logger tiedoston konfiguraation omasta tiedostostaan
        XmlConfigurator.Configure(New System.Io.FileInfo(log4netConfigurationFileName))
    End sub
```

Lopuksi vielä haetaan konfigurointi asetukset tiedostosta, jonka tiedostonnimi sekä polku on määritelty web.configiin.

Tällä yksinkertaisella tavalla saadaan eroteltua log4.netin konfigurointi omaan tiedostoon, joten tiedoston muokkaaminen voi tapahtua ilman, että web-sovelluksen käytössä tulee katkosta silloin, kun halutaan esimerkiksi logitusta lisätä.

Seuraavaksi perehdytään miten log4net olisi järkevä konfiguroida. Log4netin-konfiguraatiotiedostossa ensimmäinen määritettävä asia on eri logger–tasot.

```
<logger name="AlertLogger">
  <level value="DEBUG"/>
  <appender-ref ref="LogFileAppender" />
  <appender-ref ref="SmtpAppender" />
</logger>
```

Tässä esimerkissä on kuvattu alert–tason hälytys. Tällaista siis käytettäisiin todennäköisesti, kun jotain erittäin poikkeavaa tapahtuisi ja siihen pitäisi välittömästi reagoida, koska tämän esimerkin smtpAppender tarkoittaa sähköpostia lähettävää ilmentymää niin kuin seuraavista esimerkeistä käy selville. Tämänkaltaisia eritasoisia logger–ilmentymiä voi olla web–sovelluksessa rajoittamaton määrä.

```
  <appender name="LogFileAppender" type="log4net.Appender.RollingFileAppender">
    <file value="c:\www\test\Log\test.log" />
    <appendToFile value="true" />
    <rollingStyle value="Date" />
    <datePattern value="yyyyMMdd" />
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %-5level - %m%n" />
    </layout>
  </appender>
```

Edellinen esimerkki kertoo kuinka appender–ilmentymä määritellään. Ensimmäisellä rivillä on appenderin nimi ja tyyppi. Nimen voi jokaiselle konfiguroitavalle appenderille valita vapaasti. Logger lohkon viittaus pitää olla vastaava kuin appenderin nimi. Appenderin tyyppejä on useita, mutta yleisimmät ovat fileAppender sekä smtpAppender. Appenderin tyyppi siis kertoo mitä toiminnallisuutta se tekee. FileAppender esimerkiksi kirjoittaa tiedostoon, ja SmtpAppender lähettää sähköpostia. Muut määreet ovat ensimmäisen rivin alla liittyvät appenderin tyypistä riippuen sen toiminnallisuuteen. Esimerkin appenderin on tarkoitus kirjoittaa tekstitiedostoon test.log tietoa.(Leghari 2003.)

Muuta konfigurointia ei log4nettiä varten tarvita. Kaikki määrittely tapahtuu appender–lohkoon. Tämän vuoksi kirjasto onkin erittäin käyttökelpoinen väline erityisesti dot.net ohjelmoinnissa.

HTTP

Http–protokolla on tyypillinen clientin- ja serverin pään keskusteluun tarkoitettu tapa välittää tietoa. Clientilla tarkoitetaan siis asiakaspäätä eli asiakas hakee tietoa, jostain paikasta ja saa sen konkreettisesti itselleen. Server eli palvelinpää taas tarjoaa tietoa asiakkaalle. Tyypillinen esimerkki on web–selaimen ja serverin välinen kommunikointi HTTP–protokollaa hyödyntäen.

Tähän kuva serverin pään ja http keskustelusta.

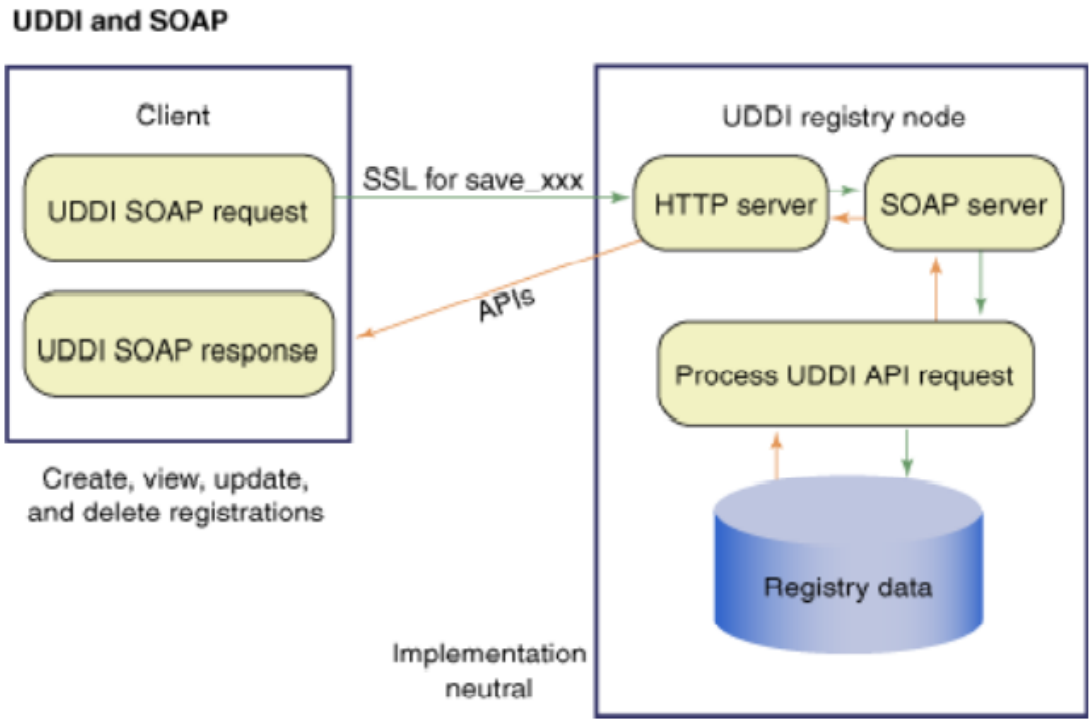
Http–protokollan yhteys toiseen kuuntelevaan päähän eli server puolelle tapahtuu Url–osoitteen avulla. Näitä siis käytetään esimerkiksi web–selaimessa, kun valitaan web–resurssia, jota halutaan tarkastella. Http–protokollan mukaisia standardeja ovat http ja https. Nämä eroavat toisistaan sillä erotuksella, että https tarjoaa suojatun yhteyden suojaamattoman sijasta. Tällä suojauksella tarkoitetaan SSL/TLS–protokollan mukaista suojaa. Tämä on erityisesti internet–maailmassa yleisesti käytetty suojauskeino.(W3C,2008.)

Http–protokolla ei itsessään ota kantaa siihen mitä siirtoväylää käytetään, mutta yleisin ja myös web–selainten hyödyntämä protokolla on TCP. Selaimet ottavat tätä protokollaa hyödyntäen yhteyden porttiin 80. jos tätä ei erikseen määritä. Eli siis esimerkiksi web–selainta käytettäessä HTTP–protokolla lähettää GET-pyynnön TCP–protokollaa hyödyntäen palvelimen porttiin 80, ellei porttia sitten ole erikseen määritelty. Yleensä kuitenkin selaimeen kirjoitetut osoitteet ovat muotoa http://google.fi eli siis porttia harvemmin määritellään. Jos portti halutaan määritellä, voidaan se kertoa URL-osoitteeseen seuraavasti: http://google.fi:82. Tässä tapauksessa siis haluttaisiin kutsua http–protokollalla Googlen TCP–porttia 82.(Kekkonen 2007.)

UDDI

UDDI on tarkoitettu helpottamaan web–palvelujen julkaisuja ja löytämistä. Tätä on usein tämän vuoksi verrattu puhelinluettelon keltaisiin- ja valkoisiin sivuihin. UDDI on siis palvelukeskeisen arkkitehtuurin rekisteri, jonka kautta palvelujen on mahdollista tavoittaa helpommin käyttäjiä. Arkkitehtuuri perustuu Xml-, HTTP- ja SOAP protokolliin ja palvelut kuvataan WSDL–kielellä. Uddilla on wsdl-kuvauskielestä huolimatta mahdollista kuvata, millä tekniikalla tahansa toteutettuja palveluja. Sillä onko palvelu Corba-, SOAP- vai Java RMI–tekniikalla tehty ei ole merkitystä.(Koljonen 2007.)

Vertaus puhelinluetteloon tulee Uddin kolmitasohierarkiasta. Valkoisilla sivuilla on tietoa palveluntarjoajasta, keltaisilla sivuilla täsmällisempää tietoa toimialasta jne. ja vihreillä sivuilla taas teknisempää tietoa palvelusta. Seuraava kuva kertoo tarkemmin, kuinka Uddin kuvauskieli teknisestä näkökulmasta toimii asiakaspään ja rekisterin välillä.



Kuva 9 Asiakaspään ja rekisterin välinen liikenne

Asiakaspää lähettää Soap-kirjekuoreen kehystetyn HTTP–pyynnön vastaanottavalle palvelimelle. Tämä käsittelee pyynnön sisällön ja suorittaa käyttäjän haluaman toimenpiteen. Operatiot, joita pyynnössä rekisteriin on mahdollista viedä ovat find, get, save, delete sekä security. Find ja get-toiminnoilla voi hakea rekisteristä tietoa, delete toiminnolla voi rekisteristä poistaa palvelukuvauksia, save operaatiolla on mahdollista lisätä näitä ja security salaa tietoja rekisterissä.(Bellwood 2002.)

PROJEKTIN ETENEMINEN VAIHEITTAIN

Opinnäytetyön aihevalinta kävi ajankohtaiseksi syksyllä 2008, kun työharjoittelun jälkeen opinnot olivat siinä pisteessä, että aihevalinta oli järkevää tehdä. Kyselin eri vaihtoehtoja yrityksestä, jossa suoritin työharjoittelun. Valinta lopulta kohdistui Smsproxy-rajapinnan ohjelmointiin. Tämä oli mielenkiintoisimman kuuloinen, joten päätin ottaa haasteen vastaan. Valintaan vaikutti myös se, että jäin töihin vielä työharjoittelun jälkeen tähän kyseiseen yritykseen ja tietotaitoa olisi välttämätöntä kehittää nimenomaan eri järjestelmien välisten yhteyksien ohjelmoinnin kohdalla.

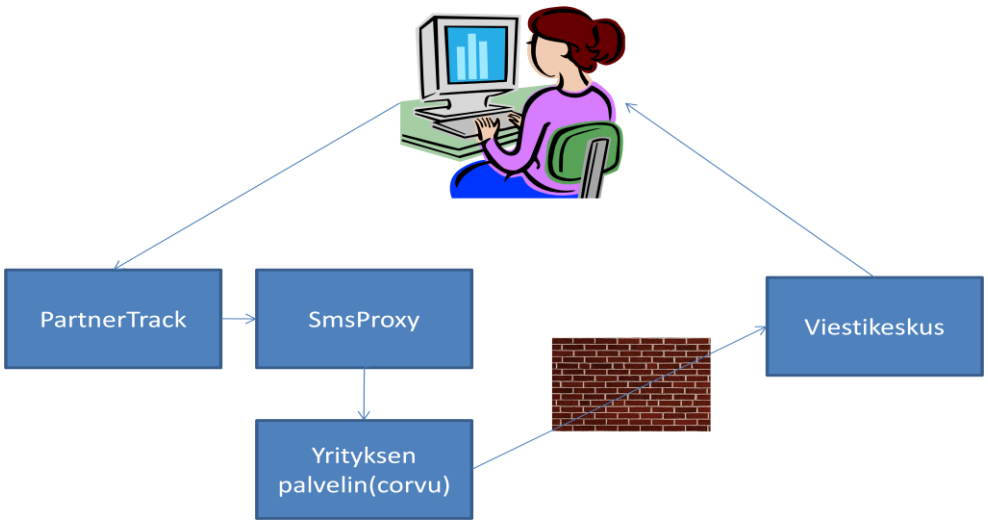
LUONNOSVAIHE JA IDEOINTI

Projektin kehitys lähti siitä, että sain tehtäväksi tutustua Content gateway -rajapinnan mahdollisuuksiin Soneran sivuilla olevien dokumenttien pohjalta. Tässä vaiheessa toteutus tapa ei ollut muilta osin selvillä kuin, että tarkoitus oli tehdä webservice–palvelu, jolla pystyy lähettämään tekstiviestejä. Tarkoitus olikin kartoittaa tässä vaiheessa, mitä mahdollisuuksia Content gateway tähän antaa. Pääasiassa tehtävänäni oli tutkia asiaa teknisestä näkökulmasta, ei niinkään siitä, mitkä ovat kustannukset. Näistä lähtökohdista dokumentteja löytyi kaksi, joihin perehtyminen keskittyi. Ensimmäinen dokumentti kuvaa palvelun eri mahdollisuuksia ja toinen eri protokollien vaatimuksia. Alkuvaiheessa mietintä keskittyi pääasiassa palvelun kuvaavaan dokumentaatioon, joka sijaitsee osoitteessa: http://www.sonera.fi/files/Sonera.fi/Yrityksille/Operaattorit%20ja%20kumppanit/Palveluntarjoajille/Content%20Gateway%20Guide%20for%20Service%20Development%20v.4.0.pdf?LinkType=Static%20File.

Tästä dokumentista alkoi suunnittelu siltä pohjalta, että onko tarvetta tehdä liikenteestä kahdensuuntaista vai riittääkö vain pelkkä lähettävä palvelu. Tämä selvittelytyö päättyi siihen, että riittävää oli tässä vaiheessa ainoastaan yhdensuuntainen liikenne. Riittävää olisi siis se, että palvelu lähettää tekstiviestejä ja tallentaa näistä tiedoista informaation, josta liikennettä on mahdollista jälkeenpäin selvittää. Kahdensuuntainen liikenne toteutettaisiin seuraavassa vaiheessa. Tähän tulokseen päädyttiin sen vuoksi, että operaattorin tarjoama palvelun toteuttaminen kahdensuuntaiseksi olisi lisännyt kustannuksia, eikä tähän tässä vaiheessa ollut välttämätöntä tarvetta.

Seuraava vaihe kehitystyössä oli lähteä käytännössä suunnittelemaan palvelun teknistä puolta. Tarkoitus oli siis päättää, mitä eri tekniikoita palvelun toteuttaminen vaatii. Tässä vaiheessa siis olisi tarkoitus piirtää eri teknisten komponenttien väliset yhteydet ja miten nämä eri komponentit keskustelevat keskenään. Suunnittelutyö vaatikin melko paljon perehtymistä asiaan, koska nämä webservice–palvelut yksistään jo olivat itselle melko uusi asia. Raja-arvoja kehitystyölle ja suunnittelulle ei oltu juurikaan asetettu, mikä osaltaan vielä vaikeutti työtä. Ainoita päätettyjä asioita oli, että kehitystyö tehtäisiin Dot.netillä ja tekstiviestejä lähetettäisiin webservice–rajapinnan lävitse.

Seuraava kuva kertoo eri komponenttien väliset yhteydet. Ensimmäisen vaiheen suunnitelmat löytiin lukkoon tämän mallin pohjalta.



Kuva 10 Komponenttien väliset yhteydet

Rajapinta oli tarkoitus ottaa käyttöön pakettien reitittämistä ohjaavassa järjestelmästä, joka kutsuu rajapintaa SOAP–protokollaa hyödyntämällä. Tämä komponentti on kuvassa nimeltä Partnertrack, joka on siis palvelua hyödyntävän sovelluksen nimi. Pyyntö ohjataan Smsproxy-palvelulle, joka ohjaa pyynnön viestikeskukselle HTTP protokollaa hyödyntäen. Viestikeskus taas lähettää itse tekstiviestin eteenpäin ja palauttaa HTTP–protokollalla vastauksen Smsproxylle onnistumisesta tai epäonnistumisesta.

Tämän komponenttisuunnitelman lukkoonlyömisen jälkeen tuli vielä jatkovaatimuksena käyttöliittymä, josta pystyisi seuraamaan palvelun lähettämiä tekstiviestäjä. Tämän työkalun pitäisi tulostaa raportteja liikenteestä käyttäjän antamien raja-arvojen mukaisesti. Suunnittelu tämän osalta ei kuitenkaan vielä tässä vaiheessa käynnistynyt vaan tämä päätettiin jättää myöhempään kehitysvaiheeseen. Tämä kuitenkin oli otettava huomioon tarkempia suunnitelmia tehdessä, joka käy tarkemmin selville seuraavassa luvussa.

TOIMINTA SONERA CONTENT GATEWAYN KANNALTA

Jotta edellisen luvun kaltainen toteutusmalli olisi mahdollista toteuttaa, on ymmärrettävä Content gateway -palvelun toimintatapa. Seuraava kuva havainnollistaa vieläkin tarkemmalla tasolla kuinka Sonera on palvelun ajatellut toimivan:

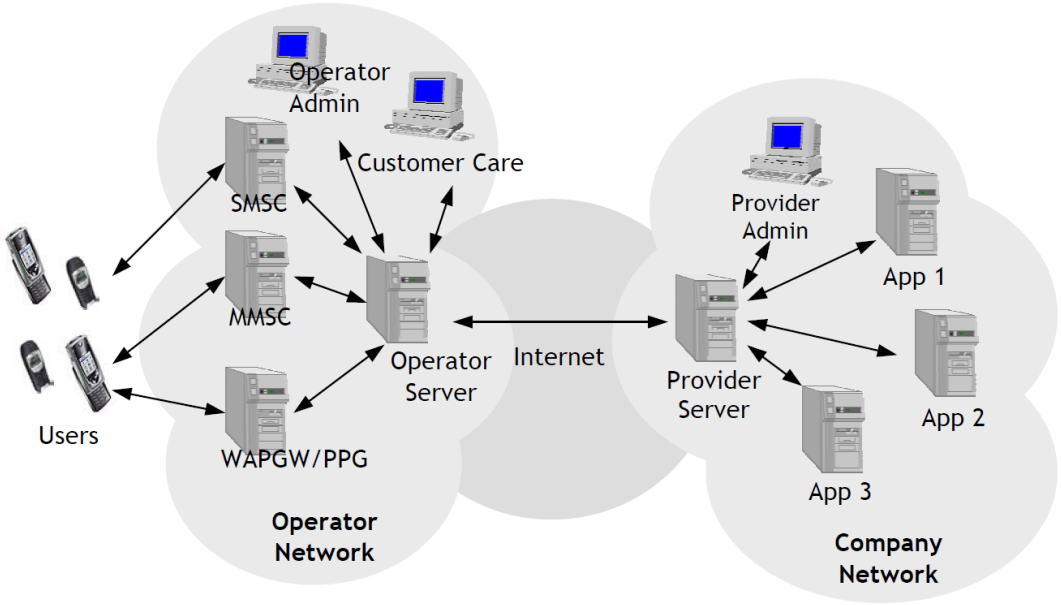


Figure 1. The Content Gateway Platform

Kuva 11 Content gatewayn toimintaperiaate

Tässä kuvassa oikealla puolella on yrityksen tietoliikenneverkko. Kuvaan on piirretty useampikin sovellus, joka kutsuu provider-palvelinta. Smsproxyn tapauksessa ei ollut tiedossa kuin yksi sovellus, joka palvelua käyttää. Ratkaisu kuitenkin on oltava kuvan mukainen eli useamman sovelluksen on pystyttävä hyödyntämään tätä samaa palvelua. Tämän vuoksi SMS–proxy onkin tarkoitus toteuttaa webservice–tekniikkaa hyödyntäen. Kuvan mukainen ratkaisu eli jokainen sovellus kutsuisi suoraan http:tä hyödyntäen ei siis SMS –proxyn tapauksessa toteudu, vaan kuvasta puuttuu webservice rajapinta, joka sijoittuu palveluntarjoajan palvelimen ja kutsuvien ohjelmistojen väliin. Eri sovellukset kutsuvat tätä rajapintaa ja SMS–proxy palvelu taas tekee kutsun palveluntarjoajan serverille.

Tämä sovellusten vasemmalla puolella oleva palvelin on siis yrityksen konesalissa. Palvelimen tarkoituksena on reitittää tekstiviestipyynnöt Internetin yli Soneralle, joka loppujenlopuksi hoitaa tekstiviestin lähetyksen. Operaattorin- ja palveluntarjoajan palvelimet käyttävät tietojen siirrossa Internetin yli TCP/IP–protokollaa. Provider-serverille eli Logia Softwaren palvelimelle on asennettuna sovellus, joka on yhteydessä operaattorin palvelimeen eli Soneraan. Operaattorin ja palvelutarjoajan välinen tunnistus tehdään lähettävän numeron perusteella ja tämä on konfiguroituna yrityksen palvelimelle. Jos tarvittaisiin useampia lähettäviä numeroita, tarvittaisiin palvelimia myös useampi. Tässä tapauksessa kuitenkin riittää vain yksi numero, koska kaikki tulevat sovellukset voivat hyödyntää Smsproxy-palvelua suoran operaattorin palvelimen kutsumisen sijasta. (Content Gateway guide for service)

Soneran palvelin (kuvassa Operator server) tarkoituksena on reitittää tekstiviesti oikeaa protokollaa hyödyntäen loppukäyttäjän matkapuhelimeen. Tämän palvelimen tiedot ovat salattuja muilta kuin operaattorin palvelinta konfiguroivilta henkilöiltä, jotka ovat Logia Softwaren teknistä henkilös-

töä. Rajapinnan ohjelmoinnin kannalta riittävä tieto on tietää palvelin minne sovellus on asennettuna. Palvelimelle asennetun ohjelmiston toimintaa ohjataan SMS –proxyn välityksellä. Rajapinta kertoo tiedon minkälaisia toimintoja halutaan tehdä yrityksen palvelimelle asennetulle sovellukselle parametrien avulla. Parametrillä tarkoitetaan tässä tilanteessa HTTP–post pyynnössä lähetettäviä tietoja. HTTP–protokollan toiminta on esitetty aiemmassa luvussa tarkemmin.

WEBSERVICE RAJAPINNAN TOTEUTUS

Rajapinnan toteutuksessa ensimmäinen asia, mistä tarkempi suunnittelu alkoi oli palvelun vaatimukset. Kuten aiemmin jo kävi selväksi, palvelun päätavoite oli lähettää tekstiviestejä, eikä ensimmäisessä vaiheessa ollut tarkoituksena hyödyntää sen laajemmin Content Gatewayn tarjoamia mahdollisuuksia. Joitakin tärkeitä kriteerejä oli kuitenkin huomioitava. Koska yrityksellä ei ollut käytössä kuin yksi lähettävän järjestelmän numero, olisi palveluun välttämätöntä saada mukaan autentikointi. Autentikoinnilla siis tarkoitetaan, että palvelun pitää tunnistaa käyttäjä tai tässä tapauksessa ennemminkin sovellus, joka sitä yrittää käyttää.

Autentikointiin on webservice–rajapintojen ohjelmoinnissa useampikin eri tapa, mutta seuraavista kriteereistä johtuen Smsproxyyn asia toteutettiin, niin, että sanomaan tuodaan mukana lähettävän järjestelmän tunnus, sekä salasana. Tähän ratkaisuun päädyttiin sen vuoksi, että tulevaisuudessa useampi eri järjestelmä mahdollisesti hyödyntää rajapintaa ja ne on tunnistettava eri sovelluksiksi. Toinen yleisesti käytettävä tapa olisi tehdä asia windows autentikoinnilla. Tässä tapauksessa vain ei saataisi tietoa lähettävästä järjestelmästä, mikä on kriittistä eri sovellusten lähettämien tekstiviestien erottelun kannalta. Windows autentikointi mahdollisuus on, jos sovellus on asennettuna Windows käyttöjärjestelmään.(Windows Authentication)

Muita tärkeitä tietoja, joita tarvitaan viestien lähetykseen ovat vastaanottajan puhelinnumero sekä lähetettävä viesti. Koska tässä vaiheessa ei muita rajaehtoja asetettu, toteutettiin rajapinta näillä parametreilla. Vastauksena rajapinta suunniteltiin palauttamaan yksinkertaisesti onnistuiko viestin lähetys vai ei. Eli siis järjestelmä asetettiin palauttamaan onnistuneesta viestistä true ja epäonnistuneesta viestistä false.

Seuraavaksi suunnitteluun tuli tekniseltä näkökulmasta rajapinnan ohjelmointi. Toiminnallisuus siis lähti niistä lähtökohdista, kuinka saadaan lähettävä sovellus tunnistettua. Tätä varten rajapintaan saapuvaan sanomaan määritettiin lähettävän järjestelmän tunnus sekä salasana. Näitä varten pitäisi siis jossakin sijaita tietovarasto, josta nuo tiedot olisivat haettavissa. Tietovarastolla tarkoitetaan varastoa, jonne on tarkoitus säilöä sovelluksen toiminnan kannalta tärkeää tietoa.(Hovi 2006.) Usein tietovarastolla tarkoitetaan tietokantaa. Smsproxyn kohdallakin tähän ratkaisuun päädyttiin, koska tunnistautumistiedot eivät olleet ainoita mitä piti säilyttää vaan myös jokaisesta lähetetystä tekstiviestistä pitäisi tietokantaan jäädä mer-

kintä. Smsproxylle siis perustettiin tietokanta, joka suunniteltiin tunnistautumistietojen sekä tekstiviestien lähetykseen liittyvien tietojen pohjalta.

Sovelluksen toimintalogiikka siis alkaa käyttäjän tunnistamisella. Jos käyttäjä tai sovellus on vahvistettu validoidaan rajapintaan tulevat muut tiedot. Tämän jälkeen yksinkertaisesti ”parsitaan” pyynnöstä tulevat parametrit HTTP–post pyyntöön ja yritetään lähettää tekstiviesti. Post–pyynnön vastauksesta tarkistetaan menikö viesti perille vai ei. Vastaanottavan pään pitäisi palauttaa kuittaus tieto, jos ”post” pyyntö on vastaanotettu. Vastauskoodi ei kerro sitä onko viesti varmasti saavuttanut vastaanottajan vai ei, mutta tähän ei ensimmäisessä vaiheessa ollutkaan tarkoitus saada kuittausta. Lopuksi vielä tallennetaan tieto siitä, onko viesti lähetetty onnistuneesti.

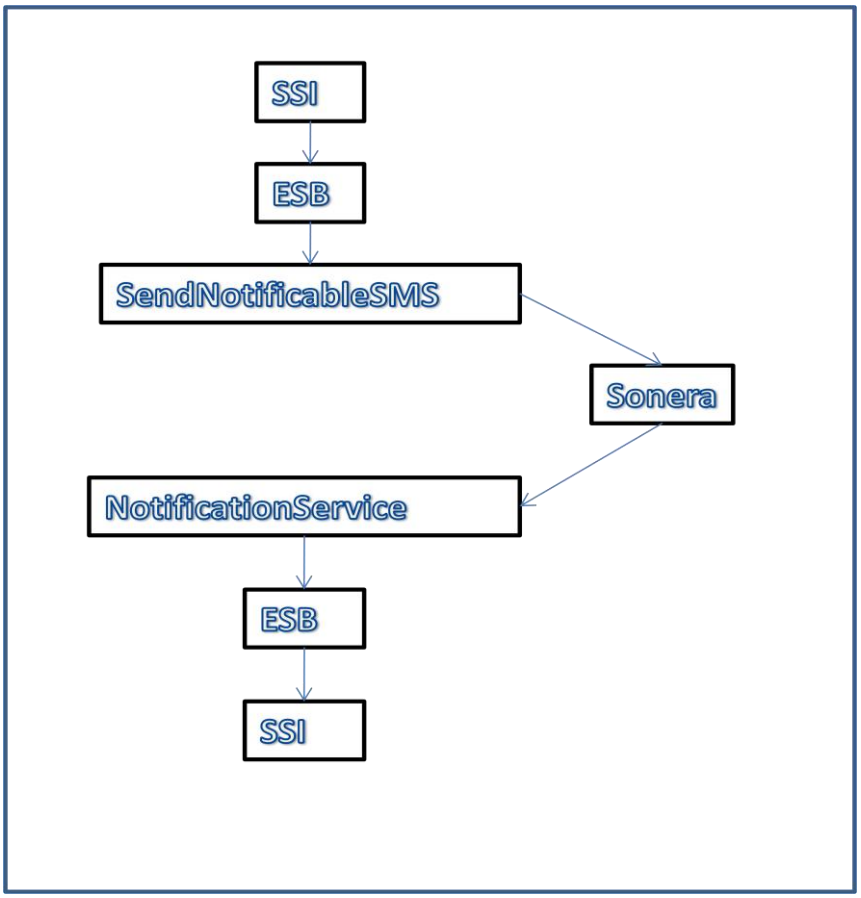
JATKOKEHITYS SMS-PROXYSSÄ

Ensimmäisen vaiheen toteutuksen valmistuttua tarvittiin SMS–proxyyn jatkokehitystä. Toinen vaihe käynnistyi kahdensuuntaisen liikenteen suunnittelulla. Tarkoituksena oli siis saada seurattua tekstiviesti liikennettä niin, että saadaan reaaliaikaisesti tieto siitä, onko tekstiviestin vastaanottaja saanut viestin matkapuhelimeensa.

Suunnittelu alkoi siitä, että kartoitettiin Sonera content gateway -rajapinnan mahdollisuuksia. Lopulta kuitenkin kävi selväksi, että tämä toteutus tapa ei tarjonnut toivottua lopputulosta. Sovellus, jossa kyseinen malli oli tarkoitus ottaa käyttöön, vaati palvelun luotettavuutta, eikä kuittausta viestin saamisesta perille ollut mahdollista saada kuuntelemalla HTTP–liikennettä.

Tästä johtuen käytettävää palvelua jouduttiin vaihtamaan Content gatewaystä toiseen palveluntarjoajaan. Tämän vuoksi käsittelen tämän laajan kokonaisuuden vain pintapuolisesti.

Toteutus eroaa merkittävästi vanhasta toteutuksesta, koska tähän liittyy paljon enemmän komponentteja ja sen myötä huomioitavia asioita. Seuraava kuva kertoo Smsproxyn väliset yhteydet toisiin järjestelmiin.



Kuva 12 Komponenttien väliset yhteydet uudessa SMS –proxy toteutuksessa

Jatkokehitystarve muodostui SSI–sovelluksen vuoksi. SSI on lyhenne sanoista sähköinen saapumisilmoitus. SSI lähettää asiakkaille tiedotteita siitä, kun asiakkaan tilaama lähetys on saapunut määränpäähänsä. Pääasiallinen ilmoituskanava on tekstiviesti, mutta kuittauksia lähetetään myös sähköpostin välityksellä. Oheinen komponenttien sijoittelu kuva kertoo kuitenkin vain SSI:n ja Smsproxyn väliset yhteydet.

Toiminta tapahtuu seuraavasti: SSI saa ensimmäiseksi tiedon saapuneesta lähetyksestä. Tämän jälkeen SSI lähettää kuittauksen ESB:lle, joka reitittää sen, joko Smsproxylle tai sähköpostin lähetyspalvelulle. Jos asiakkaan haluama tiedotuskanava on tekstiviesti lähettää ESB sen Smsproxyn operaatiolle SendNotificableSMS. Rajapinta kutsuu Soneran vastaavaa lähetysrajapintaa, joka lopullisesti lähettää viestin. Kun viesti on saavuttanut vastaanottajan lähettää sonera kuittauksen Smsproxyn operaatiolle NotificationService. Tämä palvelu käsittelee vastauksen ja lähettää siitä kuittauksen ESB:lle toteutetulle rajapinnalle, joka taas edelleen lähettää siitä kuittauksen SSI:lle.

ESB:llä tarkoitetaan palvelurajapintaa, jonka tarkoituksena on ottaa sanomia vastaan missä tahansa muodossa. Yleisimpiä tiedonsiirtotapoja ovat EDI–sanomat ja webservice palvelut, joita SSI:n toteutuksessakin on käytetty. ESB–prosessi osaa muuntaa saamansa sanoman seuraavan palvelun haluamaan muotoon ja edelleen reitittää sen oikeaan osoitteeseen.

Tämä edellinen esitelty ratkaisu ei kuitenkaan poista vanhaa toteutusta, vaan ainoastaan kutsuttavan operaation nimi muuttuu. Tietokanta tasolle on määritelty, mitkä sovellukset käyttävät vanhaa toteutusta ja mitkä taas hyödyntävät tätä edellä esiteltyä mallia, josta on mahdollista saada tieto, koska viesti saavuttaa vastaanottajan.

Tähän uuteen toteutukseen sisältyi monia ongelmia sekä asioita, mistä voisi kertoa laajasti, mutta niin kuin aiemmin mainittiin, tämän osion tarkoitus oli käydä vain pintapuolisesti asia lävitse.

ONGELMAT

Tässä luvussa on tarkoitus käydä tarkemmin läpi erilaisia ongelmakohtia, joita palvelua ohjelmoitaessa ja suunniteltaessa kohtasin. Ensimmäinen ongelmakohta oli merkistö kysymykset. Tekstiviesteissä erikoismerkit saapuivat matkapuhelimeen väärässä muodossa johtuen palvelun encoodauksesta, joka on selvitetty tarkemmin merkistö osiossa.

ongelmallinen kohta oli myös rajapinnan dynaamisuudessa, joka on selvitetty Reverse proxy -osiossa. Viimeinen vielä merkille pantava asia oli validoinnin monimutkaisuus, ja miten se on mahdollista tehdä järkevästi. Tähän otetaan kantaa validointi osiossa.

MERKISTÖ

XML-tiedostoja käsitellessä ja muutenkin tiedonsiirrossa järjestelmästä toiseen usein esille nousevat merkistöongelmat. Myös Smsproxyn kanssa ongelmat alkoivat, kun ensimmäiset testiviestit rajapinnan lävitse menivät onnistuneesti. Aluksi testiviestit koskivat vain suomalaista merkistöä ja nopeasti kävi selville, että suomalaiset erikoismerkit kuten ä ja ö eivät matkapuhelimeen saapuneet oikein. Tämä johtuu vastaanottavan palvelun encoodauksesta eli suomeksi koodauksesta. Tällä tarkoitetaan sitä missä muodossa erilaiset merkistöt järjestelmien välillä tunnistetaan. (Encoding,2010.)

Encoodauksessa on kysymys siis merkkijonojen lähettämisestä tietynkokoisessa jonossa, josta se osataan kääntää oikeaan muotoon. Yleisimmät Euroopassa käytettävät koodausstandardit ovat ISO-8859-x, UTF-8 sekä jonkin verran on käytössä vielä ASCII-encoodausta. ISO-8859-standardi on yleinen Acii–standardiin pohjautuva koodaustapa. ISO-8859–standardilla on eri koodaustapoja 16 erilaista alkaen ISO-8859-1:stä päättyen ISO-8859-16:sta. Smsproxyn testeissä kävi selväksi, että jos ISO–standardin haluaisi ottaa käyttöön, pitäisi käyttää ISO-8859-4:sta, koska tämä on näistä ainoa standardi, joka kattaa myös Viron, Latvian ja Liettuan.

Smsproxyn kannalta kuitenkin paras vaihtoehto oli UTF-8, koska se on yleisimmin käytetty koodaustyyli rajapinnoissa ja tukee lähes kaikkia merkkilajeja. UTF-8:n eduiksi on myös laskettava se, että UTF-8 koodaus

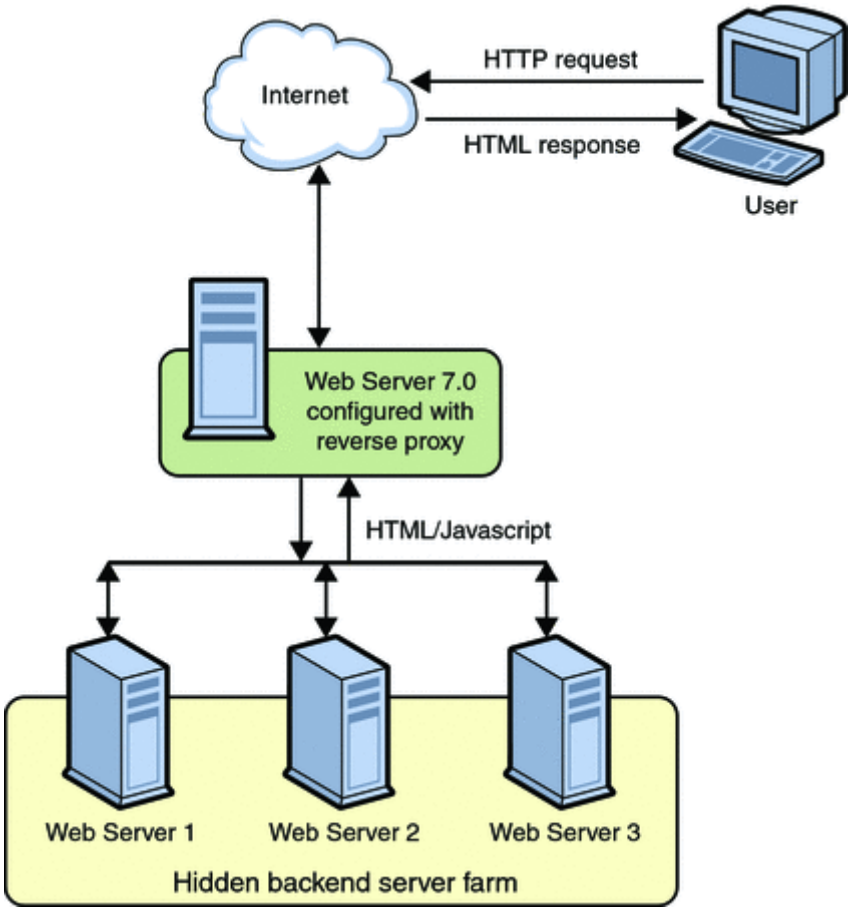
on helpompi ohjelmoida, koska se on suurimmassa osassa ohjelmointikielistä oletuksena. Oletuksena tämä oli myös Dot.net ympäristössä, jolla Smsproxy on toteutettu.

Ongelma siis oli siinä missä koodaus muodossa http-post pyyntö tulisi lähettää. Jotta vältyttäisiin siltä, että jokaista tekstiviestiä kohden jouduttaisiin valitsemaan koodaustyyli uudestaan, tuli ratkaisuksi ensimmäisessä vaiheessa ohjelmoida tiedossa olevat erikoismerkit ascii–merkistön mukaisiksi arvoiksi. Soneran palvelu ymmärtää erikoismerkit oikein jos nämä kääntää ascii–taulukon mukaisesti html–muotoon. Merkistö Virossa ei poikkea merkistön osalta Suomesta, joten tämä oli riittävä ratkaisu tässä vaiheessa. Tulevaisuudessa kun Smsproxyä hyödyntävän Partnertrackin mukaan tulee Latvia ja Liettua ratkaisua voidaan joutua miettimään uudestaan. Todennäköisesti tässä vaiheessa joudutaan kuitenkin siirtymään toiseen palveluntarjoajaan. Ongelman muodostaa nimenomaan nämä merkistö kysymykset.

REVERSE PROXY

Dot.net ohjelmoinnissa wsdl generoituu automaattisesti request eli pyyntösanoman ja response eli vastaussanoman mukaisesti. Ohjelmoija voi määrittää parametrit mitä webservice–palvelu ottaa vastaan ja wsdl:n request pyyntö generoituu tämän mukaisesti. Vastaavasti se mitä webservice–palvelun halutaan palauttavan määrittää ohjelmoija antamalla palautussanomassa haluamansa informaation. Melkein kaikki asiat mitä wsdl–tiedosto tiedosto pitää sisällään on määriteltävissä ohjelmoijan toimesta. Dot.net generoi näiden tietojen pohjalta automaattisesti eli dynaamisesti wsdl:n, jota sitten useampikin palvelu kerralla voi hyödyntää.(Poddar 2003.)

Smsproxyä ohjelmoitaessa asiat yhtä lukuun ottamatta muodostuivat niinkuin pitivätkin wsdl-rajapintakuvaukseen. Webservice –palveluilla on niin sanottu endpoint –tieto wsdl:ssä, mikä myös generoituu automaattisesti. Endpointilla tarkoitetaan osoitetta, missä palvelu fyysisesti sijaitsee. Palvelua hyödyntävä ohjelmoija siis tietää tämän tiedon perusteella. minne Soap–pyyntö ohjataan. Dot.net ympäristössä palvelun wsdl–tiedosto sijaitsee asmx-loppuisessa tiedostossa. Tähän tiedostoon endpoint–tieto generoituu sen mukaan minne palvelimelle webservice–rajapinta sijoitetaan. Tähän tietoon ei pysty ohjelmoijan toimesta vaikuttamaan mikä aiheuttaa tämän Reverse proxy ongelman, josta seuraavassa kerrotaan tarkemmin.(Poddar 2003.)



Kuva 13 Reverse proxy

Reverse proxyn -toiminta kuvataan yllä olevassa kuvassa. Komponentin toiminta siis perustuu siihen, että loppukäyttäjän ei tarvitse tietää palvelimen oikeaa nimeä, vaan tässä tapauksessa webservice–palvelun palvelimen nimeksi on mahdollista antaa kuvaavampi nimi kuin se mikä palvelimen nimi on oikeasti.

Kuvan ala laidassa on palvelimia. Yhden näistä nimi on Metkula ja palvelun nimi on Smsproxyservice. Webservice–rajapinta siis sijaitsee oikeasti osoitteessa http://metkula.logiasoftware.fi/Smsproxyservice.asmx. Tämä ei asiakkaan eikä välttämättä toteuttavan yrityksenkään näkökulmasta ole hyvä asia, että palvelu löytyy näin epämääräisestä osoitteesta. Usein ainakin palveluntarjoaja haluaa pitää palvelimien nimiavaruudet pelkästään yrityksen sisäisinä. Tästä johtuen käytetään reverse proxy -ohjausta, joka antaa palvelulle kuvaavamman nimen. Kuvassa reverse proxy siis ohjaa http://ohjelmat.posti.fi/Smsproxyservice.asmx osoitteeseen tulevat pyynnöt http://metkula.logiasoftware.fi/Smsproxyservice.asmx osoitteeseen. Näin ollen asiakkaalle voidaan julkaista tämä kuvaavampi nimi, eikä tarvitse julkaista palvelua hyödyntävälle yritykselle suoraa osoitetta.

Ongelma smsproxyssä tähän liittyen on tuon palvelun fyysisen osoitteen muodostuminen wsdl tiedostoon. Wsdl-tiedoston endpointiksi eli palvelun päätepisteeksi muodostuu tuo suora osoite eikä Dot.netin automaatiolle pysty tuota määrittämään ohjelmoitaessa. Tästä johtuen ongelma on pakko kiertää kirjoittamalla wsdl–tiedosto manuaalisesti. Tätä wsdl:n manuaali-

sesti kirjoittamista kutsutaan staattiseksi wsdl:n luonniksi. Smsproxy–palvelun endpointiksi kirjoitetaan reverse proxyn osoite ja tämä staattinen wsdl–tiedosto julkaistaan asiakkaalle.

VALIDOINTI

Webservice–rajapinnat ottavat tietoa vastaan samalla tavalla kuin esimerkiksi syötteitä vastaanottava tekstikenttä web–sivulla. Samalla tavoin kuin web–sivulla käyttäjän tai sovelluksen antamat tiedot on validoitava eli tarkistaa ovatko ne järkeviä. Palvelun ylimääräinen kuormitus ja mahdollinen väärinkäyttö on tarkoitus näillä toimenpiteillä estää. (Howard & Ingenix & Ridge 2003.)

Smsproxy–palvelun syötteiden validointi oli muilta osin melko yksinkertainen lukuun ottamatta puhelinnumeron oikeellisuus tarkistusta. Muita tietoja puhelinnumeron ja viestin pituuden lisäksi ei ollut tarpeellista tarkistaa, koska jos lähettävän järjestelmä tai salasana tiedon jättää – palvelulle antamatta loppuu prosessin suoritus siihen. Prosessilla tarkoitetaan siis koodirivien suoritusta rivi riviltä. Jos nämä tiedot eivät täsmää tietokannan tietoihin palautetaan käyttäjälle epätosi vastaussanomassa.

Viestin pituuden validointi eli yksinkertainen asia. Tähän arvioitiin jo alustavissa suunnitelmissa sopivaksi kolmen viestin mittainen rajoitus eli 480 merkkiä. Haastavammaksi kysymykseksi nousikin puhelinnumeron validointi. Puhelinnumerot suomessa ovat melko samankaltaisia, mutta haasteen tuo se, miten voidaan validointi suorittaa niin, että hyväksytään sekä ulkomaisia että kotimaisia numeroita.

Validointi olisi melko mahdotonta ilman regular expression ohjelmointikieltä. Tällä syntaksilla on mahdollista tarkistaa merkkijono tyyppisten kenttien oikeelisuuksia. Syntaksilla tarkoitetaan muotosääntöä eli millaisessa muodossa oleva syöte on oikeanlainen. Internetistä on haettavissa helposti monia käyttökelpoisia validointeja erityisesti puhelinnumeroiden oikeellisuustarkistuksiin. Näitä on hyödynnetty laajasti myös Smsproxyn puhelinnumero tiedon validoinnissa. Ongelmia kuitenkin tuotti se, että valmiit Regular expressionille toteutetut validoinnit olivat useasti suunnattu vain yhden tietyn maan puhelinnumero syötteen validointiin. Tästä johtuen Smsproxyssä on jouduttu melko paljon soveltamaan jo olemassa olevia toteutuksia.

Seuraavassa yksi esimerkki Regular expressionin puhelinnumeron validoinnista:

```
1  ^(\()?(787|939)(\)|-)?([0-9]{3})(-)?([0-9]{4}|[0-9]{4})$
```

Kuva 14 Regular expression

Yllä oleva syntaksi validoi, että puhelinnumero voi alkaa joko 787 tai 939 ja nämä voivat olla sulkujen sisällä. Jos ensimmäiset kolme merkkiä ovat sulkujen sisällä, on seuraavan kolmen merkin oltava numeroita 0-9 välillä. Jos ensimmäiset kolme merkkiä taas eivät olleet sulkujen sisällä ja seuraa-

va merkki on viiva, on syötteen oltava kolme numeroa 0-9 väliltä, viiva ja tämän jälkeen neljä merkkiä 0-9 väliltä. Jos taas ensimmäiset kolme merkkiä eivät olleet sulkujen sisällä, eikä seuraava merkki ole viiva on lopun oltava 8 merkkiä pitkä ja sisältävän numeroita 0-9 väliltä.

Kuten tästä käy selville on syntaksi melko hankala lukuista, mutta toisaalta taas pienellä ohjelmoinnilla saa aikaan melko kattavan validoinnin. Jos tämän saman asian yrittäisi tehdä pelkästään ehtolauseita muodostamalla, olisi koodi pitkä ja loppujen lopuksi myös melko epäkäytännöllinen. Ehtolauseilla tarkoitetaan siis jos – muuten tyyppistä ohjelmointia.(Pattam 2005.)

TESTAUS

Webservice-rajapinnan testaus on aina melko työläs toimenpide verrattuna web –sivun testaamiseen. Testattava palvelu saattaa kaatua jossain kohtaa ohjelmaa, eikä pistettä, jossa tämä tapahtuu ole mahdollista debugata samalla tavoin kuin perinteisen web–sivun koodia.

Debuggaaminen on nykyohjelmointia helpottava asia, jota on mahdollista tehdä lähes kaikilla moderneilla ohjelmointikielillä. Tämä tarkoittaa siis sitä, että johonkin kohtaan koodia on mahdollista asettaa niin sanottu breakpoint eli pysäytyskohta. Ohjelman suorittaminen keskeytetään tähän kohtaan, ja ohjelmoijan on tässä vaihessa mahdollista tarkastella eri muuttujien tai objektien sisältöä. Muuttajat ovat ohjelmassa nimettejä tietovarastoja joista voi tietoa lukea tai kirjoittaa. Objektit taas ovat laajempia tietovarastoja, jotka sisältävä useita muuttujia. Debuggaamalla koodia siis voidaan näiden eri tietovarastojen sisältöä tutkia tietyssä vaihessa koodia ja näin paremmin hahmottaa, jos jossain kohtaa jokin asia ei mene toivotulla tavalla. Tätä on mahdollista tehdä rivi riviltä aina ohjelmakoodin loppuun asti tai aina siihen asti, kun tapahtuu odottamaton virhe.(Davis 2010.)

Tätä asiaa webservice–rajapintojen kanssa ei ole mahdollista tehdä ilman erillistä työkalua. Smsproxyn kanssa tämä ongelma myös konkretisoitui, kun ohjelmointityö alkoi. Tähän kuitenkin löytyi erittäin hyvä ja ilmainen ohjelma nimeltään SoapUi.

Dot.net ympäristössä webservice–rajapinnat on mahdollista julkaista localhost-ympäristössä. Localhostilla tarkoitetaan paikallisen koneen kehitysympäristöä. Tarkemmin ottaen Visual studio, jolla Dot.net koodia tuotetaan asettaa työasemaan webserverin päälle ja tämän kautta webservice–rajapinta on käytettävissä paikallisesti kyseisellä työasemalla. Webserverillä tarkoitetaan työasemalle tulevaa paikallista palvelinta, jonka kautta http–protokollalla se simuloi ohjelmakoodia. Tätä voi tarkastalle selaimella tai rajapintojen ollessa kyseessä muillakin seuraavassa mainituilla tavoilla.(webserver)

Smsproxyn kannalta siis tuo edellä mainittu SoapUi oli erittäin hyvä työkalu ohjelmoinnin kannalta. Kun rajapinnan pyyntö- ja vastaussanoman parametrit olivat päätetty, oli rajapinta helppo julkaista paikallisen web-

serverin avulla. Kun tämä oli tehty, voi rajapinnan wsdl-tiedoston ensin hakea selaimeen. Tällä toimenpiteellä näkee sen, että rajapinta on varmasti localhost–palvelimella suorituksessa. Rajapinta pitäisi löytyä näin ollen esimerkiksi osoitteesta http://localhost:8810/Smsproxyservice.asmx?wsdl. Alkuosa on aina paikallisissa webserver tapauksissa localhost-alkuinen. Kaksoispisteen jälkeen tulee aina käyttöjärjestelmän käyttämä portti. Tämän voi tarkistaa Windows–ympäristössä näytön oikeassa alakulmassa olevasta webserver–kuvakkeesta. Kun vie hiiren tämän päälle, pitäisi ruudulle ilmestyä localhostin käyttämä portti. Jos ei ole varma omasta palvelunnimestään, voi kirjoittaa vain tämän alkuosan selaimeen. Tällä päästään serverin root tasolle ja tästä on nähtävissä kaikki tiedostot joita on sillä hetkellä suorituksessa. Silloin kun on kysymyksessä webservice–rajapinta, joka on ohjelmoijalla suorituksessa Visual studion kautta, valitaan asmx-loppuinen tiedosto. Tämän tiedoston klikkaamisen jälkeen pitää osoiteriville vielä kirjoittaa ”?wsdl”. Tämä näyttää palvelun wsdl–tiedoston.

SoapUi-ohjelmalla webservice–rajapintoja on helppo testata, koska se ei tarvitse muuta kuin toimivan wsdl–tiedoston, jonka pohjalta tulostuu käyttäjän näytölle automaattisesti tämän tiedoston mukainen pyyntösanoma. Tähän käyttäjä voi täyttää haluamansa arvot ja lähettää kutsun webservice–rajapinnan osoittamaan osoitteeseen, joka tässä tapauksessa osoittaa localhostiin. Tällä tavoin saavutetaan se, että käyttäjä voi samaan tapaan kuin web–sivuja tehdessä debugata ohjelmakoodia ja löytää mahdollisen ongelmakohdan helpommin.

JOHTOPÄÄTÖKSET

Tässä osiossa on tarkoitus ottaa kantaa kuinka rajapinnan ohjelmoinnissa olen onnistunut ja minkälaisia ajatuksia työn tekeminen on itsessä herättänyt. Työtä aloittaessa monet asiat olivat täysin uusia ja tässä luvussa on tarkoitus purkaa sitä mitä olen oppinut ja mitä mahdollisesti yritys on tästä hyötynyt.

OMA NÄKÖKULMA

Lähtökohdat työn onnistumiselle tuntuivat aluksi melko mahdottomilta, koska olin juuri tullut uuteen työpaikkaan työharjoitteluun ja monissa asioissa oli paljon opeteltavaa. Näistä lähtökohdista kuitenkin päätin ottaa haasteen vastaan ja ottaa opinnäytetyöksi Smsproxy–rajapinnan kehittämisen. Aiheen valintaan vaikuttivat se, että työ oli mielenkiintoisen tuntuinen kaikilta osin ja vastasi sitä mitä olisi erittäin hyödyllistä oppia työelämää ajatellen. Työn käytännön osuuden aloittamisen aikaan olin harjoittelussa kyseisessä yrityksessä ja integraatiot eri järjestelmien välillä olivat merkittävässä roolissa. Työstä olisi mahdollista saada samalla tärkeää oppia tulevaisuutta ajatellen, sekä samalla tehdä opinnäytetyö mielenkiintoisesta aiheesta

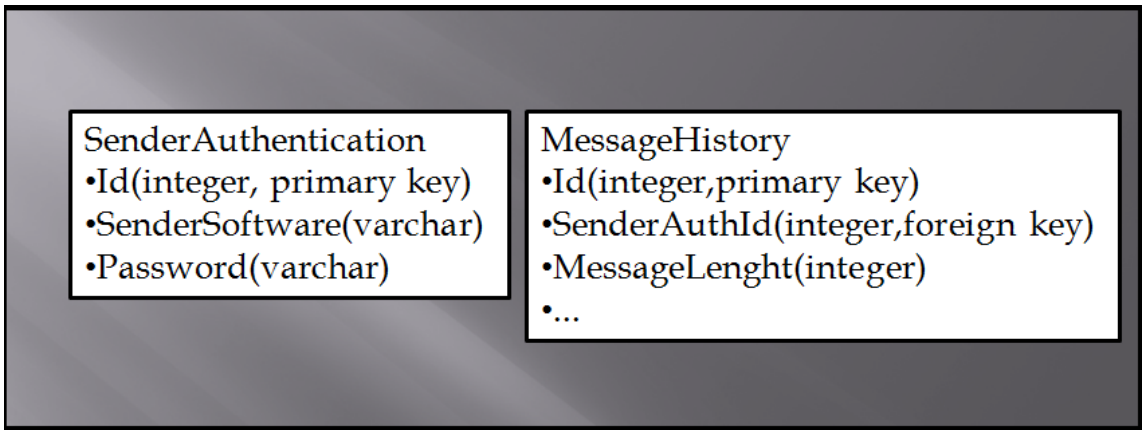
Haastavaksi työn alkuvaiheen teki se, että vastuuta rajapinnan määrittelyn ja toteutuksen suhteen oli alkuvaiheessa ehkä liiankin paljon. Työssä alkuun pääsemiseksi olisi voinut olla parempi saada jonkinlainen koulutus-

jakso, kuinka tällaisia asioita olisi järkevin toteuttaa. Toisaalta oli erittäin opettavaista ottaa asioista itse selvää ja päästä mukaan suunnittelutyöhön. Oma osaaminen vain ei aluksi tuntunut oikein riittävän, koska webservice–rajapintojen ohjelmointi dot.net kehitysympäristössä oli aivan uutta.

Työ kuitenkin toteutui aikataulussa ja palvelee käyttötarkoitustaan. Näillä kriteereillä mitattuna voisi sanoa, että työ on onnistunut, mutta joitakin asioita jälkeenpäin ajateltuna olisi voinut tehdä erilaillakin. Näistä kerrotaan tarkemmin seuraavassa luvussa.

Omasta mielestäni työ onnistui siinä mielessä hyvin, että kokemusta ei niihin aikoihin ollut, kun työtä aloin tekemään. Aikaa meni paljon asioihin perehtyessä ja tämän vuoksi nyt kun tätä myöhemmin kirjoitan herää ajatuksia muutamista asioista, jotka tekisin varmastikin erilailla. Nämä asiat, joista seuraavassa kerron eivät liity siihen, ettei rajapinta olisi ohjelmoitu niin kuin pitikin. Lähinnä on kysymys siitä, kuinka ne olisi ehkä ollut järkevämpi tehdä.

Ensimmäinen asia harmittamaan jäänyt asia on tietokanta ja sen suunnittelu. Tällä hetkellä se on toteutettu niin, että relaatioita taulujen välillä ei ole, vaikka tietokanta on erittäin yksinkertainen. Siinä ei ole kuin kaksi taulua, mutta jälkeenpäin ajateltuna niillä pitäisi olla relaatio toistensa välillä, eikä niin kuin asia on nyt eli sama järjestelmän tunnus on molemmissa tauluissa. Relaatiolla tarkoitetaan siis taulujen välistä yhteyttä. Jälkeenpäin ajateltuna tietokanta relaatio olisi pitänyt tehdä kuvan osoittamalla tavalla, eikä niin kuin aikaisemmin tietokantoja suunniteltiin eli samaan tauluun merkittiin lisää kenttiä ja hallinnointi kävi jossain vaiheessa täysin mahdottomaksi. Tietokantakuvasta on jätetty yksinkertaistamisen vuoksi joitain kenttiä pois.



Kuva 15 Tietokannan rakennekuva

Edellinen kuva on luonnos siitä, kuinka tuon taulujen välisen yhteyden voisi järkevämmin tehdä. MessageHistory–taululla olisi relaatio senderAuthenticationiin. Tämä tehtäisiin kuvan osoittamalla tavalla eli messagehistory–taulussa olisi senderAuthenticationId kenttä, joka olisi vierasavain kenttä. MessageHistory–taululla olisi näinollen relaatio aina senderAuthentication tauluun, kunhan tuo senderAuthentication–taulun mu-

kainen pääavain kenttä on asetettu. Tämä tietysti olisi asetettava vielä pakolliseksi tiedoksi, niin saadaan varmuus tuon lähettävän järjestelmän tiedon säilymisestä. Nyt tuo asia on tehty niin, että näillä tauluilla ei ole suoraa yhteyttä vaan niillä on vain epäsuora yhteys lähettävän järjestelmän tunnuksien välillä. Tämä mahdollistaa sen, että SenderAuthentication–taulusta ei välttämättä löydy tietoa, joka on messageHistory–tauluun tallennettu.

Toinen asia on logituksen minimaallisuus. Dot.net palveluissa on usein virhetilanteen sattuessa virhettä mahdotonta jäljittää, ellei ohjelma kirjoita, tiedostoon tietoa siitä mikä virheen mahdollisesti aiheutti. Smsproxy kirjoittaa logitustietoja päätason poikkeuksista, mutta minkään tasoisia validointivirheitä tai virheitä muutenkaan virhetyyppi sidonnaisesti logille kirjata. Dot.net sovelluksissa erityyppiset virheet on mahdollista ottaa kiinni virhetyypin mukaan. Esimerkiksi http–protokollan palauttamat virheet voidaan hallita omassa catch–lohkossaan.

```
1  try
2  {
3  // koodi joka voi aihettaa poikkeuksen
4  }
5  catch(System.Web.HttpException ex)
6  {
7  // tähän miten hallitaan poikkeus
8  }
```

Eli niin kuin edellisestä esimerkki näyttää mielestäni erityyppisiä virheitä pitäisi hallita eri tavalla. Joistakin virhetyypeistä pitäisi lähettää mahdollisesti jopa sähköpostia ohjelmistoa hallinnoiville ihmisille, jotta virheenselvitystyö saataisiin aloitettua riittävän nopeasti. Tällä hetkellä mahdollisen virheen tapahtuessa ei tieto välttämättä mene kenenkään tietoon.

Kolmas asia, joka mieleen tulee nyt, kun kokemusta on ohjelmoinnista ja tietojärjestelmistä yleensäkin on klusterointi kysymykset. Klusteroinnilla tarkoitetaan komponenttien kahdentamista. Joskus saattaa tulla tilanne, että esimerkiksi tietokanta lopettaa toimintansa odottamattomasta syystä. Siinä tapauksessa myös Smsproxy–palvelu ei pysty toimimaan eli lähettämään tekstiviestejä. Tässä tapauksessa olisi hyvä, että tietokanta olisi kahdennettu eli toinen noodi aloittaisi siinä vaiheessa toimintansa, kun toisella on ongelmia. Tämä ratkaisu ei tietenkään ollut ohjelmoijan eli minun vastuullani, mutta tällainen olisi mielestäni syytä olla olemassa. Vastaava asia voisi olla järkevää tehdä myös tämän palvelun kohdalla eli kun jotain odottamatonta sattuu, olisi varajärjestelmä, joka ottaisi toimintavastuun näissä tapauksissa.

Hyviä puolia on aina näin itsekriittisenä ihmisenä vaikeampi nostaa esille kuin ongelmakohtia, mutta mielestäni on syytä kuitenkin olla tyytyväinen lopputulokseen, koska kokemus projektin alkaessa oli niin vähäinen. Webservice–palvelu on toiminut kokonaisuutena sen ajan, mitä se on ollut tuotannossa luotettavasti, eikä suurempia ongelmia ole ollut. Merkistö asiat

olisivat voineet aiheuttaa ylitsepääsemättömän esteen, mutta nämä ongelmat onnistuttiin kiertämään.

YRITYKSEN NÄKÖKULMA

Yrityksen vaatimus oli, että ohjelma palvelee ensimmäisessä vaiheessa Partnertrack–nimistä sovellusta. Smsproxy tuli siis tässä sovelluksessa Viron R-kioskeihin käyttöön pakettien saapumisilmoitusten lähettämiseen. Asiakkaan tilaaman paketin saapuessa kioskille lähtee siitä ilmoitus asiakkaalle, että paketti on noudettavissa.

Tämä vaatimustaso oli tiedossa, kun Smsproxyn kehittäminen tuli ajankohtaiseksi. Muita kriteerejä oli, että palvelulle oli saatava jonkinlainen web–käyttöliittymä, jolla viestien määrää pystyi seuraamaan. Kun peilaa näitä vaatimuksia Smsproxyn toteutukseen, täyttää sovellus varmasti yrityksenkin näkökulmasta vaatimukset kirkkaasti. Web–käyttöliittymälle ei loppujenlopuksi annettukaan niin merkittävää painoarvoa, kun alun perin tarkoitus oli. Tämäkin kuitenkin projektin puitteissa toteutettiin ja palvelee myös käyttötarkoitustaan hyvin. Käyttöliittymällä pystyy tulostamaan halutulta ajanjaksolta raportteja sekä ottamaan raportteja haluamaltaan ajanjaksolta näytölle.

LÄHTEET

Mark Endrei. 2004. Patterns: Service oriented archticture and web services http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf

Setreg Khoshafian. 2007. Service oriented enterprise. Viitattu 10.12.2009 http://books.google.fi/books?id=5sFAvQWMsu4C&pg=PA143&lpg=PA143&dq=own+encoding+rules&source=bl&ots=SIMoIcyG5w&sig=ZX0fkq6x4CoCv3L9DhfXKysLv2g&hl=fi&ei=Ml-WSub-KIHp-Qbo4cDPCQ&sa=X&oi=book_result&ct=result&resnum=1#v=onepage&q=own%20encoding%20rules&f=false

Martin Tsenov. 2002 Application of SOAP protocol in e-commerce solution, Varna, Bulgaria, 59-62

CCC Group. 2004. Service development guide http://www.sonera.fi/files/Sonera.fi/Yrityksille/Operaattorit%20ja%20kumppanit/Palveluntarjoajille/Content%20Gateway%20Guide%20for%20Service%20Development%20v.4.0.pdf?LinkType=Static%20File

W3C,2007. Latest SOAP versions. http://www.w3.org/TR/soap/

Windows authentication. Integrated windows authentication. http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/iis/523ae943-5e6a-4200-9103-9808baa00157.mspx

Ari Hovi. 2006. Data warehouse – tietovarasto. http://www.pcuf.fi/sytyke/lehti/kirj/st19954/hovi954.htm

encoding,2010 http://msdn.microsoft.com/en-us/library/system.text.encoding.aspx

Seumendra Poddar. 2003. Introduction to .NET Web Services http://www.codeproject.com/KB/IP/intro2websvc.aspx

Howard, N. & Ingenix & Ridge, B. 2003 http://www2.sas.com/proceedings/sugi28/058-28.pdf

Prasanna Pattam. 2005. Server-Side Validation Using Regular Expression. http://www.15seconds.com/issue/010301.htm

Tom Davis. 2010. Debugging http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci211915,00.html

webserver

http://en.wikipedia.org/wiki/Web_server

W3C,2004. Xml Schama Part0:Primer Second Edition
http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

Nauman Leghari. 2003. Using log4net
http://ondotnet.com/pub/a/dotnet/2003/06/16/log4net.html

W3C,2008. HTTP – Hypertext transfer protocol
http://www.w3.org/Protocols/

Tom Bellwood. 2002. Understanding UDDI
http://www.ibm.com/developerworks/webservices/library/ws-featuddi/

Seppo Koljonen. 2007. UDDI(Universal Description, Discovery and Integration)
http://www.cs.helsinki.fi/u/skoljone/uddi/uddi.pdf

Saku Kekkonen. 2007. HTTP –palveluiden suorituskykytestaus.
https://publications.theseus.fi/bitstream/handle/10024/11906/2007-10-26-08.pdf?sequence=1