REST-pohjaisen web-rajapinnan kuvaaminen – Case Tax Treatment Service

Rauhala, Teemu

2015 Leppävaara

Laurea-ammattikorkeakoulu
Laurea Leppävaara

REST-pohjaisen web-rajapinnan kuvaaminen – Case Tax Treatment Service

Rauhala, Teemu
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu, 2015

Laurea-ammattikorkeakoulu
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

Rauhala, Teemu

REST-pohjaisen web-rajapinnan kuvaaminen – Case Tax Treatment Service

Vuosi 2015 Sivumäärä 29

Tämä toiminnallinen opinnäytetyö perustuu Profit Software Oy:n toimeksiantoon kuvata REST-arkkitehtuuriin perustuva web-rajapinta. Casena toimi Tax Treatment Service, joka on Profit Softwaren kehittämä vakuutustuotteen edunsaajan verokohtelupalvelu.

Työssä määritellään Tax Treatment Servicen RESTiin pohjautuva web-rajapinta ja resurssit.

REST on HTTP-protokollaan pohjautuva erilaisista rajoitteista muodostuva arkkitehtuurimalli. Opinnäytetyössä määritellään web service –teknologioita ja esitetään HTTP:n käyttämistä REST-rajapintojen kuvauksessa.

Tax Treatment Service on teknologia- ja ohjelmistokieliriippumaton. Lisäksi se on itsenäinen palvelu, joka on helposti liitettävissä mihin tahansa toiseen ohjelmistoon. Tax Treatment Service tässä muodossa on tarkoitettu Profit Softwaren sisäiseen käyttöön, mutta on helposti muunnettavissa kaupalliseksi palveluksi.

Asiasanat: web-rajapinta, Web Service, REST, HTTP, verotus

Laurea University of Applied Sciences **Abstract**
Laurea Leppävaara
Business Information Technology

Rauhala, Teemu

The Description of REST based web interface – the Case of Tax Treatment Service

Year 2015 Pages 29

This functional thesis is based on Profit Software Ltd's commission to describe web interface based on the REST architecture. As a case there is a service named Tax Treatment Service, which defines the tax treatment of an insurance beneficiary. The Tax Treatment Service is developed by Profit Software.

This work defines the REST based web interface and resources of Tax Treatment Service.

REST is an architectural pattern consisting of various constraints based on the HTTP protocol. The thesis defines web service technologies and describes HTTP protocol as part of the REST interface description.

The Tax Treatment Service is independent of technology and software language. In addition, it is an independent service that is easily connected to any other software. The Tax Treatment Service in this form is intended to Profit Software for the company`s internal use, but the Tax Treatment Service is easily convertible into commercial service.

Keywords: Web interface, Web Service, REST, HTTP, taxation

Sisällys

Johdanto

Kesällä 2014 aloitin työharjoitteluni Profit Softwaressa, jossa samaan aikaan käynnisteltiin projektia REST-pohjaisen verokohtelupalvelun Tax Treatment Service -palvelun kehittämisestä. Palvelu on osa yrityksen sisäistä ohjelmistokehitystä. Pääsin projektiin osalliseksi rakentamaan ja suunnittelemaan web-rajapintoja.

Kuten monet yritykset viime aikoina, niin myös Profit Software näkee REST:in tehokkaana keinona rakentaa moderneja web-rajapintoja ja juuri näistä syistä työharjoitteluni aikana kehitetty Tax Treatment Service noudattaa REST-arkkitehtuurin periaatteita.

Tax Treatment Servicen käyttämät web-rajapinnat on rakennettu REST-arkkitehtuurimallin rajoitteita mukaillen ja ne noudattavat HTTP-protokollaa. Palvelun tarkoitus on tarjota verokohtelu kysytylle vakuutustuotteelle. Verokohtelu Tax Treatment Servicessä toteutuu Suomen lakien mukaisesti.

Tax Treatment Servicen tavoitteena on helpottaa yrityksen ohjelmoijien työtä tarjoamalla itsenäisesti omalla palvelimella pyörivä palvelu, jota muut Profit Softwaren ohjelmistosovellukset voivat kutsua HTTP-protokollan kautta. Tax Treatment Servicestä on olemassa jo aikasempi Java-pohjainen toteutus Profit Softwaren sisäisessä järjestelmässä. Se on kuitenkin täysin Java-kieleen sidottu palvelu, joten sitä ei pysty kutsumaan esimerkiksi Internetin kautta.

Web-rajapinnoista ei ole saatavilla suomenkielistä materiaalia, joten lähdemateriaali on englanninkielistä. Suomenkielisen lähdemateriaalin puutteen vuoksi myös osa termeistä on englanninkielisiä, koska virallisia suomenkielisiä vastineita näille ei ole.

Työni on toiminnallinen, jossa pääpaino on Tax Treatment Servicen web-rajapinnassa ja sen kuvaamisessa. Esittelen Tax Treatment Servicen resurssit ja selitän niiden merkitykset palvelussa.

Työni alussa määrittelen web-rajapinta-termin ja esittelen web-rajapintojen tarjoamia liiketaloudellisia mahdollisuuksia. Tämän jälkeen esittelen web-rajapinta -teknologioita. REST-pohjaisessa palvelussa sovellukset keskustelevat HTTP-protokollan kautta. Tax Treatment Servicen resursseja kutsutaan HTTP:n yli, joten HTTP-toiminnot esittelen omassa kappaleessaan.

1.1 Profit Software

Profit Software tarjoaa IT-ratkaisuja vakuutusyhtiöiden tuotekehityksen, myynnin, sopimusten hallinnan ja korvausten käsittelyn tueksi. Profit Software on perustettu vuonna 1992 ja se palvelee tällä hetkellä 40 asiakasta yhdeksässä eri maassa.

Profit Softwarella on kolme toimipistettä kahdessa maassa: Suomessa Espoossa ja Porissa sekä Virossa Tallinnassa. Yrityksen pääkonttori sijaitsee Espoossa.

Profit Softwaren kaksi päätuotetta ovat Profit Life&Pension sekä Profit Property&Casualty.

Profit Life&Pension (PLP) on henki- ja eläkevakuutusratkaisu, jolla tuetaan vakuutusyhtiöiden liiketoiminnan ydinprosesseja. Tuotteen tavoitteena on automatisoida esimerkiksi tuotekehitystä, myyntiä sekä korvausten käsittelyä mahdollisimman pitkälle. Tavoitteena on tukea liiketoimintaprosesseja alusta loppuun. PLP perustuu Java EE-standardeihin, Business Rule -teknologiaan ja prosessimallintamiseen. PLP:n ytimen muodostaa Profit Platform -alusta. PLP on palvelukeskeinen arkkitehtuuriltaan, joten se on mahdollista ottaa käyttöön tietyn vakuutuslajin osalta, tietyn prosessin tueksi tai koko liiketoiminnan ratkaisuksi.

Profit Property&Casualty (PPC) on eri moduuleista koostuva ratkaisu, jota voidaan käyttää yksityis-, yritys-, henkilö- ja ajoneuvovakuutusten hoidossa. PPC:n avulla on mm. mahdollista hoitaa myynti, riskien valinta ja vakuutusten hoito yhdellä alustalla. Se myös sisältää tehokkaat työkalut olemassa olevien ja uusien vakuutustuotteiden hallintaan. Asiakkaita, jotka käyttävät PPC-tuotetta ovat mm. Fennia, Pohjantähti ja If.

Profit Softwaren pääpainopiste asiakkaiden suhteen on vakuutusliiketoiminnassa. Tärkeimmät asikasryhmät tulevat Suomen ja Baltian maiden henki- ja eläkevakuutusyhtiöistä. Yrityksen asiakkaita ovat muun muassa OP Henkivakuutus, Aktia Henkivakuutus, Keskinäinen Vakuutusyhtiö Fennia, IF, Compensa, SP-Henkivakuutus, AXA, Mandatum Life sekä Pohjantähti. Edellä mainitut yritykset käyttävät Profit Softwaren tarjoamia Profit Life&Pension sekä Profit Property&Casualty -tuotteita.

1.2 Tax Treatment Service

Tax Treatment Service on REST-pohjainen web-rajapintasovellus, joka kehitettiin aiemmin yrityksessä käytössä olleen JAVA-pohjaisen toteutuksen pohjalle. Se on tarkoitettu yrityksen sisäiseen käyttöön, sillä REST-pohjaisuutensa vuoksi se on helppo liittää mihin tahansa Profit Softwaren ohjelmaan, joka tarvitsee verokohtelupäättelyä tai -tietoa.

Tax Treatment Service -palvelussa ei ole omaa käyttöliittymää, sillä se on tarkoitettu vain ohjelmien väliseen keskusteluun.

2 Web-rajapinnat

Web-rajapinta-palvelussa kaksi yhteensopivaa tietokonetta kommunikoivat keskenään Internetin yli - yleensä HTTP (Hypertext Transfer Protocol) -protokollan välityksellä. Asiakasohjelma (client) lähettää Internetin välityksellä viestin palvelimelle (server), joka prosessoi viestin ja vastaa tähän lähettämällä vastauksen takaisin asiakasohjelmalle.

2.1 Määritelmä

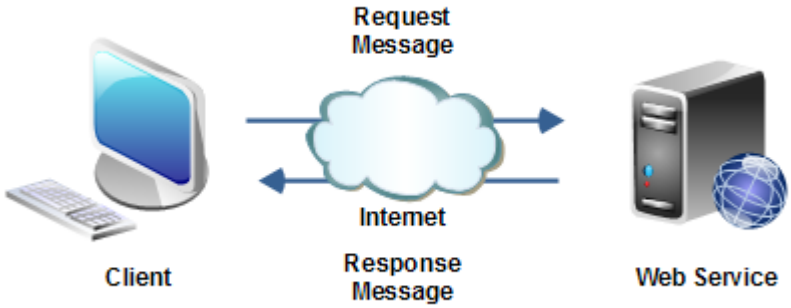
API (Application Programming Interface) eli ohjelmointirajapinta tarkoittaa rajapintaa, joka luodaan keskenään kommunikoivien tietokoneiden välille. Toisin sanoen, kaksi ohjelmaa tekevät pyyntöjä ja lähettävät tietoa keskenään. Esimerkiksi monet käyttöjärjestelmät käyttävät ohjelmointirajapintoja. Ohjelmointirajapinta on laajempi käsite web-rajapinnasta.

Molemmilla rajapintamäärittеillä (ohjelmointirajapinta ja web-rajapinta) on sama periaate, mutta niillä on yksi merkittävä ero – Internet. Ohjelmointirajapintoja käsiteltäessä ei puhuta pelkästään Internetin kautta toimivista rajapinnoista. Esimerkiksi Microsoftin Windows-käyttöjärjestelmän sisältämät rajapinnat ovat ohjelmointirajapintoja. Niiden kautta ohjelmat eivät välttämättä kommunikoi keskenään Internetin välityksellä. Web-rajapinnat puolestaan ovat täysin Internetiin kytkettyjä. Tax Treatment Service perustuu web-rajapintoihin. Kaikki kutsut palveluun tehdään Internetin kautta. Palvelua ei siis voi kutsua ilman toimivaa Internet-yhteyttä.

Web-rajapintoja koskevaa suomenkielistä materiaalia ei ole paljoa saatavissa, eikä tästä syystä web-rajapinnoille ole oikeastaan vakiintunut virallista termiä. Englanninkielisessä materiaalissa web-rajapinnoista käytetään esimerkiksi termejä Web Service tai Web API. Termien merkitys on periaatteessa sama, mutta niissä on pieniä eroja. Web Service –termi on todennäköisesti web-rajapintoja käsittelevistä termeistä yleisin. Suomenkielisessä materiaalissa niitä käsitellään Web Service –teknologioina. Tänä päivänä Web Service –palvelu yhdistetään monesti SOAP (Simple Object Access) –protokollaan perustuvaksi palveluksi. Web API –palvelulla viitataan yleensä REST (Representational State Transfer) –arkkitehtuurimalliin perustuvaan rajapinta-palveluun.

Tässä työssä käsittelen web-rajapintoja ja viittaan tällä termillä Internetin välityksellä kutsuttaviin rajapinta-palveluihin.

Kuvassa 1 havainnollistetaan web-rajapinta –palvelun toimintaa.



Kuva 1: Web-rajapinta –palvelun toiminta

W3C (World Wide Web Consortium) määrittelee web-rajapinnan seuraavalla tavalla: ” a software system designed to support interoperable machine-to-machine interaction over a network”. Kaksi yhteensopivaa tietokonetta kommunikoivat keskenään internetin yli - yleensä HTTP (Hypertext Transfer Protocol) -protokollan välityksellä. Asiakasohjelma (client) lähettää Internetin välityksellä viestin palvelimelle (server), joka prosessoi viestin ja vastaa tähän lähettämällä vastauksen takaisin asiakasohjelmalle. (W3C 2004)

Kuva 1 havainnollistaa myös tavallista Internetin käyttöä. Asiakasohjelma lähettää HTTP-pyynnön ja saa vastauksen HTML (Hypertext Markup Language) –rakenteessa. Web-rajapinta –palvelut toimivatkin samalla periaatteella kuin tavalliset HTML-pohjaiset Internet sivut, mutta näitä palveluita eivät yleensä kutsu ihmiset suoraan, vaan ihmisten ohjelmoimat asiakasohjelmat (client). Kyseessä on siis kirjaimellisesti kahden tietokoneohjelman kommunikoiminen. Tästä syystä esimerkiksi ihmisten silmille tarkoitetut HTML-elementit on karsittu varsinaisesta web-rajapinta –palvelusta monesti pois. Tiedonvälitys web-rajapinta –palveluissa tapahtuu tiedon rakennetta kuvaavien formaattien avulla. Kaksi yleisintä formaattia web-rajapinta -palvelun tiedonvälitykseen ovat JSON (JavaScript Object Notation) sekä XML (Extensible Markup Language) –standardi.

2.2 Hyödyt

Ymmärtääkseen kunnolla web-rajapintojen toiminnallisuutta, yritysten ja organisaatioiden on syytä tarkastella rajapintojen hyötyä myös liiketaloudellisesta näkökulmasta. Teknisen puolen osaaminen on tärkeää web-rajapintoja toteutettaessa, mutta jos yritykset eivät ymmärrä taloudellisia hyötyjä rajapintojen takana – työ voi olla tarpeetonta. Taloudellisten hyötyjen avulla yrityksen on mahdollista luoda menestystä luomalla uusia jakelukanavia liiketoimilleen.

Tämän päivän liiketoiminnassa tietokoneohjelmien merkitys on entistä suurempi. Ajatus yrityksestä, joka pyörittää menestyksekästä liiketoimintaa vuonna 2015 ilman taustalla

toimivaa tietojärjestelmää on lähes absurdi. Vielä noin vuosikymmen sitten toimiva tietojärjestelmä liiketoiminnan taustalla nähtiin yrityksen etuna. Nykyään toimivaa tietojärjestelmää yrityksen liiketoiminnassa pidetään itsestään selvyytenä.

Pilvipalvelut, sosiaalinen media sekä mobiilipalvelut ovat sovellusten (applikaatioiden) käytön laajentumisen pääsyitä. Nämä edellä mainitut palvelut ovat muuttaneet sääntöjä tehdä liiketoimintaa viimeisen kymmenen vuoden aikana. Sovellukset ovat kohoamassa jopa merkittävimmiksi kuin Internet-sivut. Mobiilisovellusvetoinen talous on pitkälti mahdollistettu web-rajapintoja hyödyntäen. Esimerkiksi Google, Facebook ja Twitter ovat nähneet Web-rajapintojen tarjoamat mahdollisuudet ja hyödyntäneet näitä. (Winning in the API Economy)

Web-rajapinta –palvelut voidaan jakaa sisäisiin ja ulkoisiin rajapintoihin.

Sisäiset web-rajapinnat ovat käytössä vain yritysten yksityisissä tietojärjestelmissä. Nämä palvelut saattavat pyöriä esimerkiksi yrityksen sisäisissä verkoissa, joihin on pääsy vain yrityksen työntekijöinä toimivilla ohjelmoijilla. Web-rajapinta –palveluiden ohjelmointikieliriippumattomuus helpottaa yhtenäistämään yritysten sisäisiä tietojärjestelmiä. Tax Treatment Service on tällä hetkellä yrityksen sisäisessä verkossa pyörivä palvelu. Sitä ei voi kutsua yrityksen verkon ulkopuolelta.

Ulkoiset web-rajapinnat ovat täysin julkisia palveluja. Näitä palveluita voi kutsua kaikki ohjelmoijat ympäri maailman. Jos yrityksen liiketoimintastrategiana on levittää toimintaansa mahdollisimman laajasti, niin silloin ulkoisiin eli julkisiin web-rajapinta –palveluihin satsaaminen on suositeltavaa.

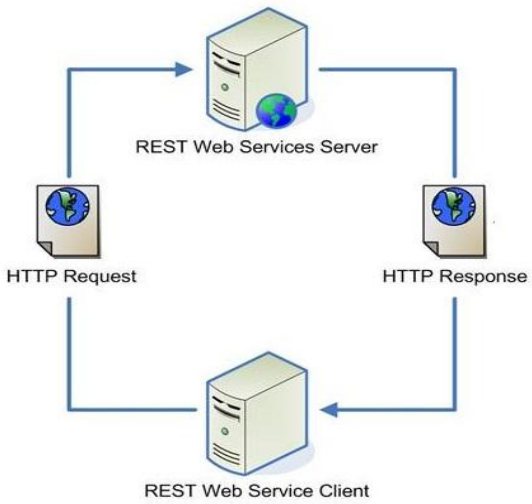
3 Web-rajapinta –teknologiat

Ennen Tax Treatment Servicen tarkempien teknisten yksityiskohtien esittelyä kerron yleisimpien web-rajapinta –teknologioiden perusteita. Esittelen tässä kappaleessa kaksi tämän hetken yleisintä web-rajapinta –teknologiaa: REST (Representational State Transfer) –arkkitehtuurimalliin perustuvat palvelut ja SOAP (Simple Object Access Protocol)-protokollaan perustuvat palvelut. Lisäksi esittelen myös Richardsonin mallin, jonka tasoihin REST-pohjaisia web-rajapinta –palveluja voi verrata.

3.1 REST

REST-pohjainen palvelu toimii web-rajapinta -periaatteiden mukaisesti. Tietoliikenne tapahtuu asiakasohjelman ja palvelimen kesken Internetin välityksellä. REST-palveluissa tiedonsiirtoprotokollana toimii HTTP (kuva 3.) REST-palvelut hyödyntävät toiminnoissaan

HTTP-protokollan toimintoja. Käytössä ovat HTTP-metodit (GET, POST, PUT ja DELETE) ja jokaisella kutsuttavalla resurssilla on REST-palvelussa oma URI (Uniform Resource Indentifier). Jos REST-pohjaisen palvelun resurssilla ei ole omaa kutsuttavaa URIa, kyseessä ei ole resurssi. Asiakas lähettää HTTP-kutsun (yleensä GET) ja palvelin vastaa tähän lähettämällä HTTP-viestin takaisin asiakkaalle. Kaikki tarvittava tieto REST-pohjaisessa palvelussa määritellään HTTP-viestien sisällössä. Kerron tarkemmin HTTP-protokollan toiminnoista kappaleessa 4.



Kuva 2: REST-pohjaisen web-rajapinta –palvelun toiminta

3.2 RESTin rajoitteet

REST-arkkitehtuurimalli muodostuu erilaisista rajoitteista. Näitä rajoitteita kutsutaan Fieldingin rajoitteiksi, koska ne määriteltiin ensimmäisen kerran Roy Fieldingin vuonna 2000 julkaistussa tietokoneohjelmien arkkitehtuuria käsittelevässä väitöskirjassa. Rajoitteet on määritelty väitöskirjassa termin REST:in alaisiksi. (RESTful Web APIs, s.29)

Roy Fielding esittää väitöskirjassaan kuusi REST:in rajoitetta, joihin viitataan aina, kun puhutaan REST:in rajoitteista. Rajoitteet ovat: asiakas-palvelin (client-server), tilattomuus (stateless), välimuisti (cache), kerroksittainen järjestelmä (Layered System), ladattava koodi (Code on Demand) sekä yhtenäinen rajapinta (Uniform Interface). Rajoitteet ja niiden määritteet ovat hyvin teoreettisia, kuten teksti Fieldingin väitöskirjassa, mutta ovat kuitenkin oleellinen osa REST-ajattelumallia, joten ne on tarpeellista esittää.

3.2.1 Asiakas-palvelin

Saadakseen toiminnon tehtyä, asiakasohjelma lähettää pyynnön palvelimelle välikappaleen (connector) kautta. Palvelin joko hylkää tai toteuttaa pyynnön ja lähettää vastauksen takaisin asiakasohjelmalle. Kaikki kommunikointi Internetissä tapahtuu kahden komponentin välillä. (Roy Fielding, kappale 5

3.2.2 Tilattomuus

Tilattomuuden tavoitteena on parantaa palvelimen skaalautuvuutta eliminoimalla palvelimen tarpeen ylläpitää huomiota asiakasohjelman tilasta yli tämänhetkisen pyynnön. Toisin sanoen, silloin kun asiakasohjelma ei lähetä palvelimelle HTTP-pyyntöä, palvelin ei ole tietoinen asiakasohjelman olemassa olosta. (Roy Fielding, kappale 5)

3.2.3 Välimuisti

Asiakasohjelma voi tallentaa tiloja verkon yli käyttämällä aikaisempia palvelimen lähettämiä vastauksia välimuistista. Tämän mahdollistavat itsekuvaavat (self-descriptive) vastaukset -rajoite, joissa kaikki tarvittava informaatio löytyy yhden vastauksen sisällöstä. Tilattomuus -rajoitteen ansiosta yksittäinen HTTP-pyyntö voidaan käsitellä omana itsenään, muista HTTP-pyynnöistä välittämättä. (Roy Fielding, kappale 5)

3.2.4 Kerroksittainen järjestelmä

Kerroksittainen järjestelmä lisää välityspalvelin (proxy) sekä yhdyskäytävä (gateway) –komponentit asiakas-palvelin järjestelmään. Nämä ylimääräiset välittäjät voidaan lisätä moninkertaisiin kerroksiin välittämään järjestelmän valvonnan (security checking) kaltaisia toimintoja.

Välityspalvelin saa HTTP-pyyntöjä komponenteilta (asiakasohjelmat, välityspalvelimet tai yhdyskäytävät) samaan tapaan kuin web-palvelin. Toisin kuin palvelin, välityspalvelin ei käsittele pyyntöä itse vaan lähettää sen toiselle komponentille (palvelin, välityspalvelin tai yhdyskäytävä) ja jää odottamaan vastausta. Vastauksen saatuaan välityspalvelin lähettää sen takaisin komponentille, jolta sai alun perin HTTP-pyynnön.

Yhdyskäytävä on välityspalvelin, joka kääntää tietoa HTTP:n tai jonkin muun protokollan välillä. Se voi ottaa vastaan esimerkiksi HTTP-pyynnön, kääntää sen komennoiksi hakea tietoa FTP (File Transfer Protocol) –serveriltä ja muuttaa ladatun tiedoston taas HTTP-muotoon.

Kun asiakasohjelma lähettää HTTP-pyynnön palvelimelle, se ei voi olla varma kommunikoiko se palvelimen, välityspalvelimen vai yhdyskäytävän kautta. Välityspalvelin ja yhdyskäytävä ovat niin kutsuttaja näkymättömiä komponentteja asiakasohjelman näkökulmasta.
(Roy Fielding, kappale 5)

3.2.5 Ladattava koodi

Asiakaskomponentilla on pääsy resursseihin, mutta ei tietoa kuinka prosessoida niitä. Asiakas lähettää pyynnön etä-palvelimelle saadakseen koodin resurssien prosessoimiseen paikallisesti. Merkittävin rajoitus on näkyvyyden puute johtuen siitä, että palvelin lähettää koodin tietorakenteen sijasta. Näkyvyyden puute voi aiheuttaa ongelmia käyttöönotossa, jos asiakasohjelma ei voi luottaa palvelimeen. Palvelin voi lähettää ajotiedoston koodin ylimääräisenä tietorakenteena. Koodi on automaattisesti otettu käyttöön, kun asiakasohjelma kutsuu sitä. Koodi mukautuu myös automaattisesti, jos se muuttuu. (Roy Fielding, kappale 5)

3.2.6 Yhtenäinen rajapinta

Yhtenäinen rajapinta koostuu neljästä eri käsitteestä: resurssien tunnistaminen (Identification of resources), resurssien hallinta representaatioden kautta (Manipulation of resources through representations), itsenäiset viestit (Self-descriptive Messages) sekä hypermedia–rajoite (The hypermedia constraint).

Resurssien tunnistaminen tapahtuu siten, että jokaisella resurssilla on oma URI (resurssien tunnistaminen). Palvelin kuvaa resurssin tilaa lähettämällä representaation asiakasohjelmalle. Asiakasohjelma voi muokata kyseisen resurssin tilaa ja lähettää siitä representaation (esitysmuodon) takaisin palvelimelle (resurssien hallinta representaatioden kautta). Kaikki tarvittava informaatio pyynnön tai vastauksen ymmärtämiseen on sisällytetty tai linkitetty viestin sisältöön (itsenäiset viestit). Palvelin manipuloi asiakasohjelman tilaa lähettämällä hypermedia-valikon asiakkaalle. Valikko sisältää vaihtoehtoja, joista asiakasohjelma voi valita (hypermedia-rajoite). (Roy Fielding, kappale 5)

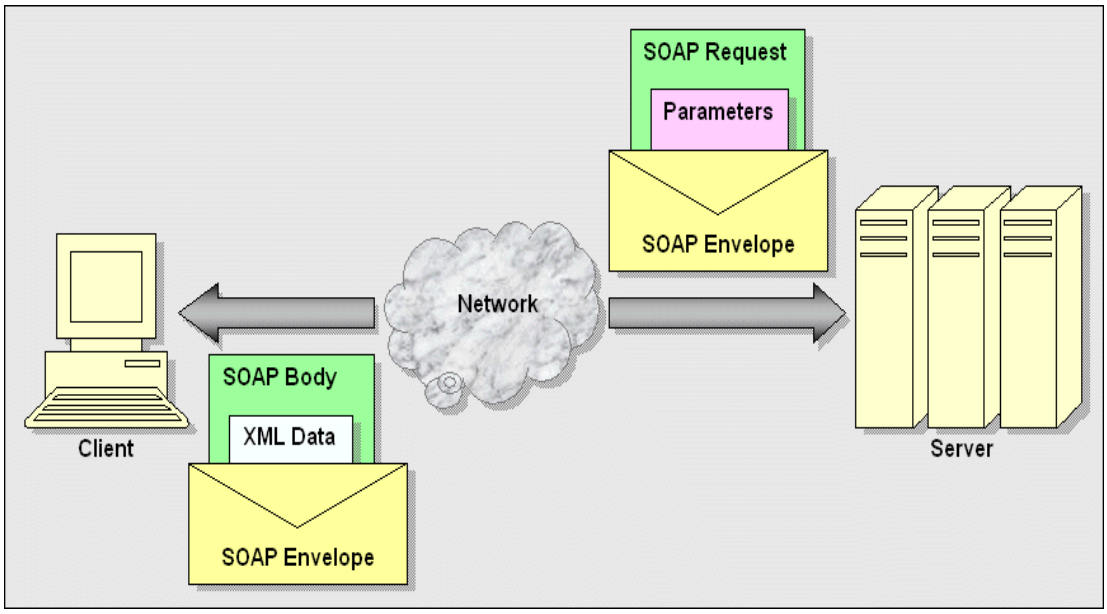
3.3 SOAP

Microsoftin kehittämä SOAP-protokolla on Web Service –tekniikoista varhaisimpia.

SOAP–protokollassa viestin välitys tapahtuu asiakasohjelman ja palvelimen välillä web-rajapinta–toiminnallisuuden mukaisesti. SOAP-protokolla määrittelee XML (Extensible Markup Language) –pohjaisen viestiformaatin, joka mahdollistaa tiedonvaihdon Internetin yli. SOAP perustuu XMLään, koska protokollan avulla lähetettävät viestit ovat XML-formaatissa. SOAP:ia

käytetään yleensä HTTP-protokollan kautta, mutta se on yhteensopiva myös muiden siirtoprotokollien, kuten esimerkiksi SMTP:n (Simple Mail Transfer Protocol) ja FTP:n (File Transfer Protocol) kanssa. (W3C 2007)

SOAP:in kautta lähetettävää XML-formaattia noudattavaa viestiä kutsutaan kirjekuoreksi (envelope). Kirjekuori koostuu ylätunnisteesta (header) ja rakenteesta (body) sisältäen tarvittavaa informaatiota itse viestin lähettämiseen. Tämä informaatio kertoo muun muassa kuinka viestin tietorakenne on tarkoitus prosessoida. Kirjekuori sisältää tietoa myös viestin lähettäjästä ja saajasta. Ylätunnus (header) ei ole välttämätön ominaisuus SOAP:in viestissä. Ylätunnisteesta löytyvä informaatio voidaan hyödyntää esimerkiksi käyttöoikeuksien tarkistamista tarvittaessa. SOAP:in rakenne (body) sisältää varsinaisen datan. Rakenne-osion sisältö voi olla pyyntö informaatiolle tai vastaus informaation pyynnölle. Osion datarakenne on organisoitu ala-elementeiksi. Rakenne-osio voi koostua yhdestä tai useammasta ala-elementistä. Kuva 3 havainnollistaa SOAP:in viestien rakennetta. (W3C 2007)



Kuva 3: SOAP-pohjaisen web-rajapinta –palvelun toiminta

WSDL (Web Service Definition Language) on XMLään perustuva kuvauskieli, jota käytetään SOAP:in palveluiden kuvaamiseen.

3.4 Richardsonin malli

REST-palveluista on tällä hetkellä olemassa lukemattomia erilaisia toteutuksia ja niitä suunnitellaan ja toteutetaan lisää jatkuvasti. Leonard Richardsonin malli (Richardson Maturity

Model) ottaa kantaa web-rajapintojen totetutuseroihin.
(http://www.crummy.com/writing/speaking/2008-QCon/act3.html)
Leonard Richardson on ollut vahvasti mukana RESTiä koskevassa kirjallisuudessa. Hän on yksi RESTful Web APIs –teoksen tekijöistä, joka toimii tämän työn kirjallisena lähdemateriaalina.

Mallin tavoitteena on erotella REST –pohjaiset ja SOAP –pohjaiset web-rajapinta –palvelut toisistaan. Esimerkiksi

Richardsonin mallin perusteella palvelut voidaan jakaa neljään eri tasoon sen mukaan, kuinka paljon ne noudattavat RESTin määritteitä. Nämä tasot ovat: taso 0, taso 1, taso 2 ja taso 3.

Tasosta 0 puhuttaessa ei voida puhua oikeastaan lainkaan REST-palvelusta. Tällä tasolla on vain yksi URL, joka pitää sisällään kaikki palvelun resurssit. HTTP-metodeita on käytössä vain yksi ja se on yleensä POST-komento. Jos web-rajapinta –palvelu vastaa tason 0 vaatimuksia, niin kyseessä on todennäköisesti SOAP-palvelu.

Tasoa 1 noudattava web-rajapinta –palvelu toteuttaa osan REST-toiminnallisuudesta. Tällä tasolla jokaisella resurssilla on oma URL. Tasolla 0 kaikki resurssit sijaitsevat saman URL:n takana, mutta tasossa 1 URL:iä on monia. Tason 1 palvelulla on vain yksi HTTP-metodi käytössä.

Tasoa 2 noudattava REST-pohjainen palvelu ottaa huomioon HTTP-metodit (GET, POST, PUT ja DELETE). Siinä jokainen metodi toteuttaa omia operaatioita. Resursseilla on omat URL:t tason 1 mukaisesti. Tason 2 palvelu hyödyntää myös HTTP:n statuskoodeja viestinnän yhteydessä.

Tason 3 REST-pohjainen palvelu on muuten identtinen tason 2 kanssa, mutta tällä tasolla palvelun vastauksissa hyödynnetään vielä lisäksi hypermedia-linkkejä. Jokaisessa vastauksessa on siis hypermedialinkki sisällyttenä, joka ehdottaa asiakasohjelmalle seuraavaa mahdollista pyyntöä. Asiakkaan ei tarvitse siis välttämättä päätellä seuraavaa pyyntöä itse. Tason 3 palvelu on niin sanottu ”Fully RESTful” eli se noudattaa vahvasti REST-arkkitehtuurin periaatteita.

3.5 Mediatyypit

Mediatyyppi on lyhyt merkkijono, joka identifoi dokumentin formaatin. Kun dokumentin mediatyyppi on tiedossa, sen voi jäsennellä. REST-palvelu tukee lukuisia eri medityyppi-vaihtoehtoja. Suositeltavaa onkin valita mediatyyppi omien tarpeidensa mukaisesti.

Tax Treatment Servicen mediatyypiksi valittiin JSON (JavaScript Object Nation). JSON on standardi, joka esittää yksinkertaiset tietorakenteet tavallisena tekstinä. Se on nimestään huolimatta JavaScript riippumaton.

JSON kuvaa merkkijonot seuraavalla tavalla:

”Tämä on merkkijono”

Listat esitetään seuraavasti:

[1, 2, 3]

Avainparit esitetään seuraavasti:
{”avain”: ”arvo”}

(www.json.org)

4 HTTP

REST-pohjaisessa web-rajapinta –palvelussa kaikki tiedonvälitys tapahtuu HTTP-protokollan välityksellä. Tässä kappaleessa esittelen HTTP-protokollan ja sen tärkeimpiä toiminnallisuuksia web-rajapintojen näkökulmasta. Tax Treatment Servicen web-rajapinta on rakennettu tässä kappaleessa esiteltävien HTTP-toimintojen varaan. On huomioitavaa, että REST-arkkitehtuurimallin rajoitteet määritellyt Roy Fielding on ollut kehittämässä HTTP-protokollaa 1990-luvulla.

4.1 Määritelmä

HTTP-protokolla kuuluu TCP/IP –mallin ylimpää eli sovelluskerrokseen. TCP/IP on Internetin arkkitehtuurin kuvaamiseen tarkoitettu viitemalli. Se koostuu viidestä eri kerroksesta. Nämä ovat: sovelluskerros (Application Layer), kuljetuskerros (Transports Layer), verkkokerros (Internet Layer) sekä peruskerros (Link Layer). (Internet protocol suite)

4.2 Resurssit ja esitysmuoto

HTTP:n merkitys tulee esille jo hyvin aikaisin REST-rajapintoja suunnitellessa. Suunnitteluvaihessa nimetään toteutettavassa palvelussa noudatettavat resurssit. REST-palvelussa jokaisella nimetyllä resurssilla on oma URI. Jos resurssilla ei ole omaa URI:ia, on kyse jostakin muusta kuin REST-palvelusta.

Resurssi on yleensä jotakin, mikä voidaan tallentaa tietokoneelle. Se voi olla sähköisessä muodossa oleva dokumentti tai esimerkiksi rivi tietokannassa. Asiakasohjelman näkökulmasta katsottuna ei ole väliä mikä resurssi itsessään on, koska se ei koskaan näe resurssia. Asiakasohjelma näkee vain URI:it ja resurssien esitysmuodot. (RESTful Web APIs, s.30)

Kun asiakasohjelma lähettää kutsun resurssille, palvelin lähettää dokumentin, jonka tarkoitus on kuvata resurssia halutulla tavalla. Tätä lähetystä kutsutaan esitysmuodoksi – koneellisesti käsiteltäväksi (machine-readable) esitykseksi resurssin sen hetkisestä tilasta. Palvelin lähettää esitysmuodon asiakasohjelman haluamalla tavalla. Palvelin voi kuvata rivin tietokannassa XML-dokumenttina, JSON-objektina tai pilkulla toisistaan erotettavina arvoina (comma-separated values). Esitysmuoto kuvaa siis aina pyydetyn resurssin tilaa. Esitysmuoto voi olla mikä tahansa koneellisesti käsiteltävä dokumentti, joka sisältää mitä tahansa informaatiota pyydetystä resurssista. (RESTful Web APIs, s.30)

Esitysmuodot mielletään yleensä tiedostoina tai dokumentteina, jotka palvelin lähettää asiakasohjelmalle. Käyttäessään Internetiä, ihminen on tottunut pyytämään resursseita eri palvelimilta. Mutta kun kyseessä REST-ohjelmointi, asiakasohjelma lähettää esitysmuotoja myös palvelimelle. Palvelimen tehtävä on tämän jälkeen muotoilla resurssi asiakasohjelman lähettämän esitysmuodon mukaiseksi. Toisin sanoen palvelin lähettää esitysmuodon pyydetystä resurssin tilasta asiakasohjelmalle. Sen jälkeen asiakasohjelma lähettää esitysmuodon samasta resurssista takaisin palvelimelle, jossa resurssin tila on puolestaan asiakasohjelman pyynnön mukainen. Tätä toimenpidettä kutsutaan nimellä Representational State Transfer (REST). (RESTful Web APIs, s.32)

4.3 HTTP-statuskoodit

HTTP-statuskoodi on kolminumeroinen luku, joka palautetaan HTTP-vastauksen yhteydessä. HTTP-statuskoodeja on yhteensä 41 kappaletta ja ne on määritelty HTTP:n spesifikaatiossa. Tavalliselle Internetin käyttäjälle statuskoodit eivät ole niin oleellisia, mutta niiden merkitys kasvaa huomattavasti web-rajapintoja suunniteltaessa. Statuskoodi kertoo asiakasohjelmalle kuinka tarkastella palautettua dokumenttia: onko kyseessä normaali viestin esitysmuoto vai ilmoitus ilmenneestä virheestä. Statuskoodien avulla asiakasohjelma tietää miten pyyntö on onnistunut. (RESTful Web APIs, s.295)

Esittelen statuskoodeista viisi niin kutsuttua perhettä, joihin statuskoodit jaetaan ensimmäisen numeronsa perusteella:

1xx: Informational: Näitä statuskoodeja käytetään vain asiakasohjelman keskenäisessä neuvottelussa.

2xx: Succesful: Palautetaan silloin, kun asiakasohjelman pyytämä muutos on tapahtunut.

3xx: Redirection: Palautetaan silloin, kun asiakasohjelman pyytämä muutos ei ole tapahtunut. Mutta jos asiakasohjelma pystyy tekemään pienen muutoksen HTTP-pyyntöön, pyydetyn resurssin pitäisi palautua asiakasohjelman haluamalla tavalla.

4xx: Client Error: Palautetaan silloin, kun asiakasohjelman pyytämä muutos ei ole tapahtunut, HTTP-pyynnön kanssa tapahtuneen ongelman takia. Pyyntö on joko epämuodostunut, ristiriitainen tai muodoltaan sellainen, jota palvelin ei pysty hyväksymään. Näissä tilanteissa vika on asiakasohjelman puolella. Se on saattanut esimerkiksi kutsua resurssia, jota ei ole ollenkaan olemassa.

5xx: Server Error: Palautetaan silloin, kun asiakasohjelman pyytämä muutos ei ole tapahtunut palvelimen puolella tapahtuneen ongelman johdosta. Tässä tilanteessa asiakasohjelmalla ei ole oikeastaan muuta vaihtoehtoa kuin odottaa, että palvelin taas vastaa pyyntöihin. (RESTful Web APIs, s.297)

4.4 HTTP-metodit

REST-palvelussa liikenne tapahtuu asiakasohjelman ja palvelimen välillä HTTP-protokollan välityksellä. REST-palvelut koostuvat URIsta, jotka määrittävät yksittäisen resurssin. Resurssien tilaa muokataan esitysmuotojen avulla. Kun puhutaan resurssien hallinnasta, niin tällöin HTTP-metodien merkitys tulee esille. Kaikki liikenne HTTP:n kautta tapahtuu HTTP-metodien avulla.

HTTP-metodeja on yhteensä yhdeksän: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT sekä PATCH. Neljä yleisintä metodia ovat GET, PUT, POST ja DELETE. HEAD ja OPTIONS ovat hyödyllisiä asiakasohjelman näkökulmasta. CONNECT ja TRACE –metodit liittyvät enemmän välityspalvelimiin.

REST-palvelussa olevaa operaatiota kutsutaan idempotenssiksi, kun asiakasohjelman lähettäessään pyynnön kerran ja muuttaessaan resurssin tilaa voi tehdä saman kutsun tämän jälkeen toistuvasti saamalla koko ajan saman tuloksen. Tulos on siis aina sama, riippumatta siitä tehdäänkö pyyntö kerran vai useamman kerran. (RESTful Service Best Practices, s.10)

Niin kutsutut turvalliset metodit eli safe-metodit ovat operaatioita, jotka voi suorittaa ilman huolta mahdollisista taustavaikutuksista. Saman pyynnön voi suorittaa lukemattomia kertoja samalle palvelimelle muuttamatta yhdenkään resurssin tilaa. Safe-metodit onkin implementoitu vain vastausten lukemista varten. (RESTful Service Best Practices, s.11)

Yleisimmin käytössä olevat HTTP-metodit REST-pohjaisissa web-rajapinnoissa ovat GET, POST, PUT ja DELETE, joten käsittelen kyseiset metodit seuraavaksi yksitellen.

GET-metodi palauttaa kysytyn resurssin esitysmuodon. GET-komentoa on käyttänyt varmasti jokainen, joka on käyttänyt Internetiä. Sivuilta toiselle siirtyminen tapahtuu GET-metodin avulla. Normaalille Internetin käyttäjälle esitysmuoto tulee yleensä HTML-muodossa. Web-rajapinnoissa esitysmuotona on yleensä XML tai JSON. HTTP-spesifikaation mukaan GET-pyyntö on tarkoitettu vain datan lukemiseen, ei sen muokkaamiseen. GET on safe-metodi. Tämän vuoksi GET-pyynnön voi lähettää samalle resurssille esimerkiksi kymmenen kertaa resurssin tilaa muuttumatta. HTTP–statuskoodi GET-komennon jälkeen on yleensä 200 (OK) tai 404 (Not Found). (RESTful Service Best Practices, s.11)

POST-metodia käytetään uusien resurssin luomiseen. POST ei ole safe-metodi eikä idempotenssi. Jos POST-käskyn tekee samalle resurssille esimerkiksi kaksi kertaa, niin sen jälkeen on todennäköisesti kaksi identtistä saman informaation omaavaa resurssia. Onnistuneen resurssin luomisen jälkeen HTTP palauttaa statuskoodin 201 (Created). (RESTful Service Best Practices, s.12)

PUT-metodi muuttaa resurssin tilan esitysmuodon pyynnön mukaiseksi. PUT-pyynnön tavoitteena on muuttaa olemassa olevan resurssin tilaa päivittämällä sen esitysmuotoa. PUT ei ole safe-metodi, koska se muuttaa palvelimen tilaa mutta se on kuitenkin idempotenssi. Jos PUT-komennolla muuttaa resurssin tilaa, niin se muuttuu vain kerran. Kyseisen resurssin tila pysyy täsmälleen samana, kun tekee saman komennon uudelleen. PUT-komennon jälkeen HTTP:n palauttama status-koodi on yleensä 200. (RESTful Service Best Practices, s.12)

DELETE-metodi poistaa halutun resurssin. DELETE-komennon jälkeen haluttu resurssi yksinkertaisesti poistuu koko järjestelmästä. Onnistuneen komennon jälkeen HTTP palauttaa status-koodin 200. DELETE-komennot ovat idempotensseja. Yhden DELETE-komennon jälkeen resurssi ei ole enää olemassa. Jos sitä kutsuu uudelleen, HTTP lähettää todennäköisesti status-koodin 404. (RESTful Service Best Practices, s.13)

Neljällä päämetodilla on kaikilla omat tehtävät ja merkityksensä, mutta POST- ja PUT-metodit ajavat osittain samaa asiaa. PUT-pyynnöllä on mahdollista myös luoda resurssi. PUT-komentoa on suositeltavaa käyttää resurssin luomiseen, kun asiakasohjelma päättää, mikä URI

kyseiselle resurssille annetaan. POST-komennon käyttö on suositeltavaa silloin, kun palvelin on vastuussa siitä, mikä URI uudelle resurssille annetaan. (RESTful Service Best Practices, s.13)

5 Tax Treatment Service

5.1 Implementaatio

Web-rajapintaohjeet eivät ota yleensä kantaa implementaatio-vaiheeseen. REST:n periaatteita noudattavan palvelun yksi suurimpia etuja on se, että sen voi implementoida haluamallaan tavalla. Tästä syystä web-rajapintojen suunnitteluvaiheessa ei oteta implementaatiota huomioon, jonka takia en kuvaa Tax Treatment Servicen taustalla olevaa implementaatiota ollenkaan.

5.2 Resurssit

Tax Treatment Service koostuu viidestä eri resurssista. Kyseessä on REST-palvelu, joten resurssit määritellään URI:n avulla ja niitä käsitellään HTTP-metodeilla. Palvelun resurssit ovat: /productTypes, /coverTypes, /claimTypes, /roleTypes sekä /taxTreatment.

Tax Treatment Service palauttaa arvot JSON-muodossa. Kuten aiemmin mainittiin, kyseessä on koneellisesti käsiteltävää tietoa, joten HTML- ja CSS-koodit ovat karsittu pois. Vaihtoehtoinen tapa JSON-formaatille on XML-formaatti, mutta Tax Treatment Servicessä on päätetty hyödyntää JSONia.

Tax Treatment Service määrittää yksittäisen edunsaajan verokohtelun yksittäiseesä tarkasti rajoitetussa tilanteessa.

5.2.1 /productTypes

Vakuutusyhtiöllä voi Profit Life & Pensionissa olla useita kymmenä vakuutustuotteita. Niille jokaiselle ei ole olemassa omaa verotuskohtelua. Verotuksen kannalta merkitsevä asia on tuotetyyppi (product type). Kukin vakuutustuote kuuluu tasan yhteen seuraavista tuotetyypeistä: savings (säästövakuutus), pension (eläkevakuutus) tai capitalization (pääomavakuutus).

/productTypes palauttaa kaikki palvelussa mukana olevat vakuutustuotetyypit.
http://workbench.profitsoftware.com:8080/tax-treatment/productTypes

Koska Tax Treatment Servicen mediatyyppinä on JSON, niin productTypes -kysely palauttaa seuraavanlaisen vastauksen:

JSON vastaus: {"productTypes":[{"name":"group pension","id":"4444"},{"name":"new savings","id":"6341"},{"name":"capitalization","id":"5678"},{"name":"savings","id":"1234"},{"name":"pension","id":"2345"}]}

Yllä olevan vastauksen sisällöstä löytyy aiemmin mainitut tuotetyypit (savings, pension ja capitalization). ProductTypes -resurssi ei sisällä lainkaan parametrejä.

5.2.2 /coverTypes

Kullakin tuotetyypillä on tietty joukko vakuutusturvatyyppejä (cover type). Tyypillisesti vakuutuksia otetaan sen vuoksi että saadaan turva joidenkin tapahtumien varalle.

/coverTypes palauttaa palvelusta kysytyt covertyypit määrätyn tuotetyypin osalta. Resurssin parametri on productType.

http://workbench.profitsoftware.com:8080/tax-treatment/coverTypes?productType=savings

coverTypes -resurssin kysely sisältää yhden parametrin: productType. On kaksi vaihtoehtoa kuvata parametreja REST-kyselyssä. Nämä vaihtoehdot ovat seuraavat.

/tax-treatment/coverTypes/savings (as part of the URL-path)
/tax-treatment/coverTypes?productType=savings (query argument)

Tax Treatment Servicessä on käytetään kaikkien parametrien osalta query argument -menettelyä. Parametrit erotellaan siis URIsta löytyvällä ? -merkinnällä.

JSON vastaus: {"coverTypes":[{"name":"death","id":"3333"}]}

5.2.3 /claimTypes

Eri tuotetyypeillä voi olla erilaiset korvaustyypit (claim type).

/claimTypes palauttaa palvelusta kysytyt korvaustyypit. Resurssin parametrit ovat productType ja coverType

http://workbench.profitsoftware.com:8080/tax-treatment/claimTypes?coverType=death&productType=savings

JSONvastaus: {"claimTypes":[{"savingType":null,"name":"death","id":"8315"},{"savingType":null,"name":"cancel poli-
cy","id":"6327"},{"savingType":null,"name":"maturity","id":"7319"},{"savingType":null,"name":"partial maturi-
ty","id":"2735"},{"savingType":null,"name":"surrender","id":"5555"},{"savingType":null,"name":"partial surrender","id":"4444"}]}

Yllä oleva kysely palauttaa seuraavat korvaustyypit: vakuutuksen peruminen (cancel), osittainen erääntyminen (partial maturity) sekä osittainen keskeyttäminen (partial surrender).

5.2.4 /roleTypes

Vakuutussopimuksiin liittyy aina tyypillisesti useita rooleja. Esimerkkinä rooleista on mm. Vakuutuksenottaja, vakuutettu ja maksaja. Kaikki vakuutusroolit eivät verotuksen kannalta ole merkityksellisiä. roleTypes resurssi kyselyllä saadaan tietää mitkä roolit missäkin tilanteessa ovat tarpeen ja se, mitä tietoja kustakin tarvittavasta roolista tarvitaan.

/roleTypes palauttaa palvelusta kysytyt roolityypit. Kysely palauttaa kunkin tarvittavan roolin lisäksi ko. rooliin liittyvät tiedot, jotka on annettava taxTreatment- kyselylle. Resurssin parametrit ovat productType, coverType sekä claimType.

http://workbench.profitsoftware.com:8080/tax-treatment/roleTypes?claimType=surrender&coverType=death&productType=savings

JSON vastaus:

{"roleTypes":[{"idReq":true,"closeRelativeReq":false,"personReq":false,"limitedTaxReq":false,"claimPaidFromReq":false,"taxBaseReq":false,"name":"insured","id":6666},

{"idReq":true,"closeRelativeReq":false,"personReq":true,"limitedTaxReq":false,"claimPaidFromReq":false,"taxBaseReq":false,"name":"policyholder","id":7777},

{"idReq":true",closeRelativeReq":true,"personReq":true,"limitedTaxReq":true,"claimPaidFromReq":false,"taxBaseReq":false,"name":"beneficiary","id":8888}]}

Verotuksen kannalta merkityksellisiä roolityyppejä on kolme. Nämä ovat vakuutettu (insured), vakuutuksen ottaja (policyholder) sekä vakuutuksen edunsaaja (beneficiary)

closeRelativeReq = Onko tämä kyseinen rooli lähisukulainen vakuutetulle. Req (Required) = false – ei tarvita /taxTreatment –pääkyselyssä. Req = true – tarvitaan /taxTreatment – pääkyselyssä. personReq = yksityishenkilö tai yritys. limitedTax = rajoitettu verotus. claimPaidFrom = Onko osuus, jota maksetaan edunsaajalle pääomaa (capital) vai voittoa (yield). taxBase = verokanta. Verokantoja on olemassa muutama kappale. Niiden arvoja muun muassa IT, IT2, CT, CT2, CT3. IdReq = tarvitaanko kyseisen roolin id antaa (true) tai ei

tarvitse (false). Verotuksen kannalta on oleellista mm. se, onko edunsaaja (roleBen) sama henkilö kuin vakuutuksenottaja (rolePH).

5.2.5 /taxTreatment

/taxTreatment on palvelun pääresurssi. Tämä resurssi palauttaa kysytyn korvauksen maksun verotuskohtelun. Resurssin parametreinä ovat mm. productType, coverType, claimType ja roleTypes.

http://workbench.profitsoftware.com:8080/tax-treatment/taxTreatment?productType=savings&rolePHPerson=true&roleBenPerson=true&roleBenCloseRelative=false&roleBenLimitedTax=true,claimType=surrender&rolePHId=123&roleBenId=345&roleInsId=123

rolePHPerson=true tarkoittaa, että vakuutuksen ottaja on yksityishenkilö. Vastaavasti rolePHPerson = true tarkoittaa, että myös vakuutuksenottaja on yksityshenkilö (individual). Tässä kyselyssä edunsaaja ei ole lähisukulainen vakuutetun kanssa (role<benCloseRelative=false). Lisäksi edunsaaja on oikeutettu rajoitettuun verotukseen (roleBenLimitedTax=true). Korvaustyyppi casessa on takaisinosto (surrender). Ja tässä on kesessä tilanne, jossa vakuutuksenottaja on eri henkilö kuin edunsaaja (rolePHId != roleBenId)

JSON vastaus:

{"beneficiaryType":"person not next of kin","taxCode":"D1","taxClassValue":"0-99","taxDefaultPercentage":30,"taxReduction":true,"taxClass":"withholding tax"}

beneficiaryType = yksityishenkilö joka ei ole lähimainen (person not next of kin), taxCode = DI – verottajan verokoodi. taxClassValue = 0-99. taxDefaultPercentage = 30 – oletusveroprosentti. taxReduction = true – eli alennettu verotus. taxClass = withholding tax – ennakkovero.

5.3 Tax Treatment Service ja Richardsonin malli

Tax Treatment Service noudattaa Richardsonin mallin tasoista tasoa numero 1. Palvelulla on käytössään monia resursseja, mutta HTTP-metodeista käytössä on vain GET-metodi. Tax Treatment Servicen kohdalla se on kuitenkin tarkoituksen mukaista, ettei kukaan ulkopuolinen taho voi muuttaa Suomen verotukseen perustuvaa verotuslakia. Täten palvelu on

ainoastaan kyselypalvelu. Jos Suomen laki muuttuu verotuksen osalta, lain muutokset hoidetaan muuttamalla implementaatiota.

6 Yhteenveto

Profit Life & Pension (PLP) on ollut Profit Softwaren asiakkailleen tarjoama päätuote viimeisen kymmenen vuoden ajan. PLP:tä on myyty eri vakuutusyhtiöille. Suurin osa asiakkaista on suomalaisia yrityksiä. Tuotetta myydään myös ulkomaille, muun muassa Liettuaan, Latviaan, Puolaan, Viroon sekä Ruotsiin. PLP kattaa eläke-, henki- sekä säästövakuutukset. PLP on standalone -sovellus, joka on nykyään myös web-pohjainen. Tax Treatment Servicen tarkoitus on olla vaihtoehtoinen tapa korvata aikasempi Java-toteutus REST-pohjaisella web-rajapinta -palvelulla. Palvelun toiminnallisuus siis pysyisi samana kuin aikaisemmassa Java-toteutuksessa, mutta tekniikka puolestaan olisi uusi.

REST-pohjaiset web-rajapinnoilla on kiistattomia etuja. Ne ovat ohjelmointikieliriippumattomia eli REST palvelua voi kutsua millä tahansa kielellä. REST-pohjaiset web-rajapinnat ovat myös itsenäisiä ja riippumattomia, niitä ei sidota kutsuttavaan ohjelmaan mitenkään.

Tax Treatment Service on tehty Profit Softwaren sisäiseen käyttöön. Sisäisen käytön jälkeen palvelu on mahdollista tehdä myös sellaiseen muotoon, että se olisi myytävissä esimerkiksi muille vastaaville ohjelmistointegraattoreille kuin Profit Software.

Viimeinen visio Tax Treatment Servicestä voisi olla jopa sellainen, että tavalliset kansalaiset voisivat selvittää oman verokohtelunsa vakuutustuotteiden osalta.
Internetin käytön monipuolistuminen muutaman viime vuoden aikana on web-rajapintojen yleistymisten pääsyistä. Internet-sovelluksia ei enää kutsuta pelkästään tietokoneelta vaan esimerkiksi kännykällä erilaisten mobiilisisovellusten kautta. Web-rajapinnat mahdollistavat nykyaikaisen sovellusvetoisen liiketalouden.

Profit Softwaren bisneslogiikka kannattaa tulevaisuudessa toteuttaa yhä enemmän REST-pohjaisilla web-rajapinta -palveluilla Java-toteutuksen sijaan. Java –pohjaista toteutusta on vaikeata ylläpitää ja laajennettavuus on hankalaa. REST-rajapintoja on helppo käyttää niiden yksinkertaisuutensa ansiosta, joten suosittelen, että yritys käyttäisi näitä jatkossa myös muissakin projekteissa.

Lähteet

Kirjalliset:

Richardson, L., Amundsen, M., Foreword by Ruby, S., 2013. RESTful Web APIs. O´Reilly.

Sähköiset:

Fielding, Roy Thomas, Architectural Styles and the Design of Network-based Software Architectures. Kalifornian yliopisto, Irvine. 2000
https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Fredrich, T. RESTful Service Best Practices. Recommendations for creating Web Services.
http://www.restapitutorial.com/resources.html

SOAP Version 1.2 Part 0: Primer (Second edition). W3C. 2007
http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

Internet protocol suite. Verkkodokumentti. Wikipedia
http://en.wikipedia.org/wiki/Internet_protocol_suite

Introducing JSON. Verkkodokumentti
http://www.json.org/

Richardson, L. Justice Will Take Us Millions Of Intricate Moves. Act Three: The Maturity Heuristic. Verkkodokumentti
http://www.crummy.com/writing/speaking/2008-QCon/act3.html

Web Service Architecture. W3C. 2004
http://www.w3.org/TR/ws-arch/#id2260892

Winning in the API Economy. 3 Scale
http://www.3scale.net/wp-content/uploads/2013/10/Winning-in-the-API-Economy-eBook-3scale.pdf

Kuvat

Liitteet

Liite 1 Esimerkki Tax Treatment Servicen Test clientista ja Java-implementaatiosta

Test Client:

```
@Test
   public void getTaxTreatmentTest() {

     try {
         URL url = new  URL(http://localhost:8080/taxtreatment/taxTreatment?productType=
         pension&rolePHPerson=true&roleBenClaimPaidFrom=yield&savingsType=
         risk%20life%20share&roleBenLimitedTax=false&claimType=surrender&rolePHId=
         123&roleBenId=123&roleInsId=123");
         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
         conn.setRequestMethod("GET");
         conn.setRequestProperty("Accept", "application/json");

         if (conn.getResponseCode() != 200) {
            throw new RuntimeException("Failed :HTTP error code : " +
            conn.getResponseCode());
         }

         BufferedReader br = new BufferedReader(new InputStreamReader(
                  (conn.getInputStream())));

         String output;
         System.out.println("Output from Server (tst tax) .... \n");
         while ((output = br.readLine()) != null) {

           System.out.println(output);
         }

         conn.disconnect();

      } catch (MalformedURLException e) {

         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
```

Esimerkki java implementaatiosta:

```
@Path("/")
public class JSONTaxService {

   @GET
   @Path("/claimTypes")
   @Produces("application/json")
   public ClaimTypes getClaimTypesInJSON( @QueryParam("productType") String pts,
                                          @QueryParam("coverType") String cot ) {

      DecisionServiceKmodule ds = DecisionServiceKmodule.getInstance();
      Object outObject;
      List<Object> inList = new ArrayList<Object>();

      DecisionContext dc = new DecisionContextImpl();
```

```
        dc.setScope("decision");
        dc.setRuleFlow("claimTypes");
        inList.add( dc );

        ProductType pt = new ProductTypeImpl();
        pt.setName( pts );
        pt.setId( pts );
        inList.add( pt );

        CoverType covert = new CoverTypeImpl();
        covert.setName( cot );
        covert.setId( cot );
        inList.add( covert );

        FactFactory ff = new FactFactoryImpl();
        inList.add(ff);

        ClaimTypes claimts = new ClaimTypesImpl();   // here comes the results...
        inList.add( claimts );

        System.out.println("running rules...");
        ds.runProcessAndRules(inList);
        System.out.println("out of rules ...");

        return claimts;
    }
}
```