hyväksymispäivä arvosana

arvostelija

Palveluiden haku palveluorientoituneessa tietojenkäsittelyssä

Toni Könnilä

Helsinki 18.12.2015

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Toni Könnilä

Työn nimi — Arbetets titel — Title

Palveluiden haku palveluorientoituneessa tietojenkäsittelyssä

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	18.12.2015	19 sivua + 1 liitesivua

Tiivistelmä — Referat — Abstract

Service-oriented computing (SOC) on ajatusmalli, jossa yhden monoliittisen ohjelman sijaan painotetaan keskenään kommunikoiviin palveluihin. SOC nojaakin hyvin vahvasti SOA-arkkitehtuurimalliin, joka ajaa tätä samaa ideaa. SOA ei kuitenkaan tarjoa itsessään vastauksia siihen miten turvallisuus, palveluiden koordinointi ja ohjelman keskinäinen arkkitehtuuri on rakennettu. Tässä aineessa keskitytään palveluorientoituneen arkkitehtuuriin kuuluvaan palvelun hakuun (*service discovery*).

ACM Computing Classification System (CCS):
A.1 [Introductory and Survey],
I.7.m [Document and text processing]

Avainsanat — Nyckelord — Keywords
ulkoasu, tiivistelmä, lähdeluettelo

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

SOC on ajatusmalli, joka painottaa eri palveluiden käyttöä palvelun koostamiseksi. Yleensä lopputuloksena on siis palvelu, jonka ideana on kommunikoida muiden palveluiden kanssa, jotka ovat itsessään alustariippumattomia ja väljästi yhteydessä toisiinsa (*loosely coupled).* Business-taustaisena ideana SOA:ssa (*Service-oriented architectures*) on tuottaa hyvin keskenään toimivia, mahdollisimman atomisia palveluita, jotka ovat uudelleenkäytettäviä, helposti korvattavissa ja ylläpidettävissä tuottaen nopeasti kustannustehokkaita hajautettuja palveluja lyhyellä kehitys-julkaisuikkunalla (*time-to-market*).

Yksittäisen palvelun tehtävä voi olla yksittäinen pyyntö, tai monimutkaisempi businessprosessi, ja tällaisten palveluiden tarjoajat (olivatpa ne sitten organisaatioita tai yksityishenkilöitä) mahdollistavat ohjelmallisen pääsyn palveluihinsa ja dataansa internetin yli käyttäen standardoituja välitystapoja, käyttäen yleensä XML-perustaisia tai REST-mallin mukaisia HTTP-protokollaan perustuvia rajapintoja käyttäen. Näiden palveluiden tarjoajien (*service providers*) vastuulla on antaa palvelukuvaus ja ylläpitää palvelua, ja antaa palveluun liittyvää teknistä sekä muuta tukea. Palvelua kutsuvat tahot voivat olla muita ohjelmia, tai yksittäisiä tahoja, esimerkiksi kotipalvelin joka pyytää resurssia REST-rajapinnan toteuttavalta palvelulta internetin yli. Tästä seuraa, että näillä näillä palveluilla on tiettyjä vaatimuksia ja edellytyksiä.

- *Teknologiariippumattomuus*: Palvelun ei kuulu olla liian teknologiariippuvainen, vaan sitä pitää pystyä kutsumaan alalla vallitsevilla työkaluilla ja alan standardien mukaisesti.
- *heikko linkittyneisyys (loosely coupled)*: palvelun tulee olla irti mahdollisesta kontekstista, ja toimia itsenäisesti (ainakin ulospäin) ilman tietoa muista palveluista tai komponenteista.
- *Palvelun haettavuus (service discovery) ja tuki sijaintiläpinäkyvyydelle (location transparency)*: Palveluiden määritykset ja varsinaiset sijaintitiedot tulee olla tallennettuna johonkin - esimerkiksi UDDI repositorioon, ja asiakkailla/-tahoilla pitää olla pääsy palveluun tietämättä missä se varsinaisesti sijaitsee.

Tässä aineessa keskitytään palveluiden hakuun, haettavuuteen ja siihen miten se voidaan toteuttaa palveluorientoituneessa arkkitehtuurimallissa.

2 Palveluorientoitunut arkkitehtuuri

Perinteisesti SOC palvelumalli johon nojataan, on ollut SOA. SOA pyrkii käyttämään erillisiä palveluja koostaakseen monimutkaisiakin ohjelmistoja. Näiden erillisten palvelujen on tarkoitus olla uudelleenkäytettäviä ja joustavia, ja siten nopeuttaa sekä halventaa ohjelmistokehitystä.

Erillinen yksittäinen palvelu eroaa ajatusmaailmaltaan monoliittisen arkkitehtuuripuolen mallista tietyissä osa-alueissa. Se on itsenäinen, väljästi linkitetty, uudelleenkäytettävä, itsensä kuvaava ja helposti siirrettävä toiselle alustalle. Monoliittisessa arkkitehtuurissa palvelut ovat yleensä tiukasti sidottu sovittuihin teknologioihin, ja edelleen sidottuina käytettävän teknologian vaatimuksiin ja rajotteisiin. Ohjelmat ovat tiiviisti linkattuina muihin kirjastoihin, ja yksittäisten luokkien ja palveluiden korvaaminen päätyy lopulta useiden moduulien muokkaamiseen tätä uutta korvaavaa moduulia tukevaksi.

Myös business-mielessä monoliittinen arkkitehtuuri kasvaa usein turhauttavaksi: yhä kasvavan ohjelman monimutkaisuus ja sisäinen linkittyneisyys muodostuu hidasteeksi uusille projektiin liittyville työntekijöille ja vaatii enemmän ja enemmän aikaa ohjelman kokonaisuuden ymmärtämiseksi. Ohjelman luokkien ja palveluiden linkittyneisyys muodostaa myös kokonaisuuden, jota on vaikeampi muokata, jolloin uudelta työntekijältä saattaa jäädä kooditasolla huomiotta tehdä muutokset kaikkiin paikkoihin joihin muutokseen liittyvä moduuli on kytköksissä. Tämä ongelma on usein itseään ruokkiva, ja ongelmat kasaantuvat entisestään, jos koodin laadusta ei pidetä tasaisesti huolta. Tätä palveluorientoituneisuus sen sijaan yrittää helpottaa rajaamalla palvelut selkeiksi itsenäisiksi kokonaisuuksiksi.

Monoliittisessa arkkitehtuurissa koko ohjelma voidaan nähdä isona siilona, kun taas SOA on yksinkertaisimmillaan infrastruktuuri, joka tukee erillisten palveluiden kommunikointia keskenään standardien protokollien avulla. Myös näiden palveluiden tulee olla löydettävissä alalla vallitsevien standardirajapintojen avulla. Kun nämä osat toimivat, ohjelmat pääsevät tarvittaessa helposti käsiksi näihin palveluihin ilman tarvetta monimutkaisille räätälöidyille kommunikointiprotokollille.

SOA on looginen tapa suunnitella ohjelmistoja joko yksittäisten käyttäjien tarpeeseen, tai rajapinnaksi muille ohjelmistoille. Perinteinen SOA erittelee (kts. Kuva 1) osapuolet palvelun jakajaan, palvelua kutsuvaan osapuoleen (asiakas) sekä palvelurekisteriin jonka avulla pyydetty palvelu löydetään.

Palveluntarjoajan vastuulla on antaa palvelukuvaus ja ylläpitää palvelua, ja antaa

palveluun liittyvää teknistä sekä muuta tukea. Palveluntarjoajan vastuulla on myös pitää huoli, että palvelu on löydettävissä palvelurekisterin avulla.

Palvelua kutsuvat tahot voivat olla muita ohjelmia, tai yksittäisiä tahoja, esimerkiksi kotipalvelin joka pyytää resurssia REST-rajapinnan toteuttavalta palvelulta internetin yli.

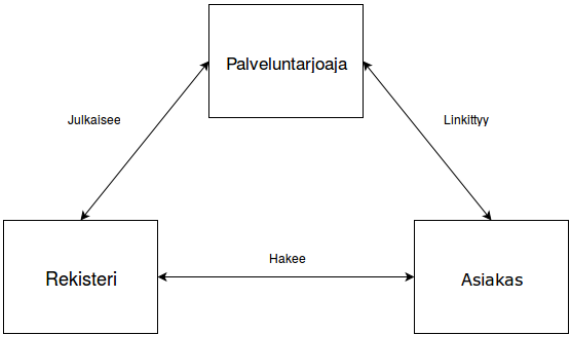
SOA:lle tyypillisessä skenaariossa palveluntarjoaja rakentaa toteutuksen palvelusta ja antaa sille palvelukuvauksen. Tämän jälkeen julkaisee sen palveluhakutaholle tai rekisterille, jota kautta palvelu on löydettävissä. Palvelua hakeva osapuoli pääsee käsiksi palveluun tällaisen tahon/rekisterin kautta (kuten esimerkiksi UDDI) ja linkittyy palvelukuvauksen avulla palveluntarjoajaan, jonka jälkeen kutsuva osapuoli pääsee kutsumaan palvelutoteutusta.

Palveluorientoituneet arkkitehtuurit toteuttavat SOC:in ideaa, mutteivat ota kantaa esimerkiksi turvallisuuteen, transaktioiden hallintaan tai koordinaatioon[Pap03].

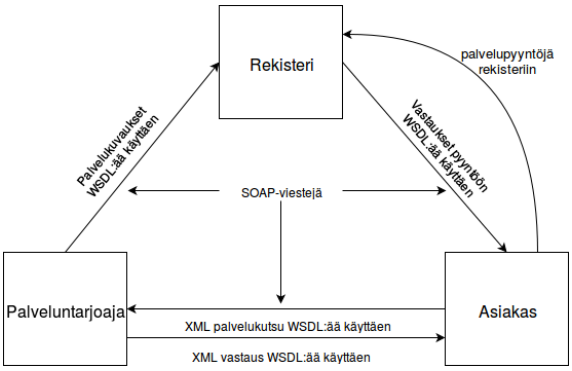
3 Palveluiden haku

3.1 Yleisesti

Palveluntarjoajilla on hallussaan toteutus palvelusta, jota he ylläpitävät ja he myös määrittävät palvelukuvauksen tarjoamalleen palvelulle. Palvelukuvausten tarkoituksena on ilmoittaa palvelun tarkoituksesta, käyttäytymisestä, laadusta ja sen tulee muodostaa alusta löydettävyydelle, linkittymiselle ja palvelukompositiolle. Palvelut voivat tarvittaessa olla yhteydessä toisiinsa monimutkaisemmissa prosesseissa, kuten vaikkapa luottokortin tarkistuksessa tai tuotteen tilauksessa. Koska nämä palvelut ovat luonteeltaan alustariippumattomia, luo se puitteet kompositiopalveluiden tuottamiselle liittämällä yhteen toiminnaltaan yksinkertaisia tai monimutkaisempia palveluita.



Kuva 1: Perinteinen SOA



Kuva 2: SOAP viestintä

Web service -malli koostuu kolmesta komponentista: asiakkaasta, palveluntarjoajasta sekä rekisteristä. Palveluiden haun kannalta nämä kaikki ovat oleellisia. Ilman rekisteriä kutsujaa ja tarjoajaa ei voida linkittää, koska yleensä asiakas ei tiedä missä palvelu sijaitsee. Palveluntarjoaja julkaisee palvelun rekisterissä luomalla WSDL (Web Service Description Language) määrityksen, joka on XML-pohjainen rajapintakuvaus[Cha14], joka kertoo kutsuvalle taholle miten pyydettyä palvelua voidaan kutsua, millaisia parametreja se odottaa, missä muodossa vastaus tulee ja millaisissa tietorakenteissa vastauksen sisältö on[W3C07b]. Perinteisesti nämä tahot ovat keskustelleet keskenään SOAP:n avulla. SOAP on protokolla, joka alkujaan oli akronyymi sanoista Simple Object Access Protocol. Myöhemmissä versioissa päätettiin ettei SOAP ole akronyymi ollenkaan.

SOAP luotiin toimimaan HTTP-protokollan yli, koska sitä tuetaan kaikissa internet selaimissa ja tarjoaa muodon, jolla XML viestit kääritään[W3C07a]. Näin kaikki palvelut, jotka tukevat SOAP:ia voivat keskustella keskenään, eikä ole väliä miten palvelut ovat teknisesti toteutettu. Rekisterissä on siis WSDL-kuvaus palvelusta. Web service -mallissa asiakastaho tekee pyynnön rekisteriin, jonka jälkeen rekisteri vastaa kutsujalle lähettämällä palan WSDL-määrittelystä. Tämän jälkeen, mikäli

palautettiin oikeanlaista tietoa, asiakkaalla on kaikki tarpeellinen tieto linkittyäkseen palveluun. Saamansa WSDL:n avulla asiakas tekee pyynnön palveluntarjoajalle, joka antaa edelleen WSDL-määritellyssä muodossa vastauksen asiakkaalle. Kaikki nämä pyynnöt palveluntarjoajan, rekisterin ja asiakkaan välillä ovat perinteisessä mallissa SOAP-muotoisia. Havainnollistus kuvassa 2.

Tällainen malli on hyvin perinteinen, oleellisina osina ovat WSDL, rekisterit ja SOAP viestit. On kehitetty myös muita tapoja keskustella palvelun ja asiakkaan välillä. Nykyään yhä useammat web-palvelut toteutetaan REST-tyyppisesti. Ongelmaksi palvelun haun kannalta muodostuu se, ettei REST:ä ole standardoitu, eikä siis ole olemassa standardoitua tapaa miten palveluiden haku REST-tyyppisille rajapinnoilla toteutetaan. Normaalisti REST-tyyppisen palvelun haku perustuu sen URI:n. Esimerkiksi osoitteessa http://yhtio.com/api/kayttajat/ voisi listata kyseisen palvelun käyttäjät. On täysin palveluntarjoajan vastuulla, että asiakkaat löytävät kyseiset resurssit. Hyvässä tapauksessa resurssien metadatasta löytyy tietoa siitä mitä voidaan tehdä seuraavaksi tai mitä muita resursseja voidaan hakea. REST ei siis varsinaisesti ota kantaa miten palvelun haku on toteutettu.

Erinäisiä lähestymistapoja yhdistää REST:iä traditionaaliseen malliin kuitenkin löytyy. Eräs ohjelmointi- ja palvelunkoostamistapa, jossa kaikki yksittäiset palvelut ovat täysin hajautettuja ja jotka keskustelevat keskenään RESTmäisesti, ilman keskinäisiä palvelumäärityksiä ja sopimuksia, on mikropalveluarkkitehtuuri. Mikropalveluarkkitehtuuri on nimitys, jonka puolestapuhuja esimerkiksi Martin Fowler on. Käytännössä mikropalvelut ovat SOA arkkitehtuurin mukaisia, mutta eräät kokivat SOA:n terminä liian laveaksi ja monimerkitykselliseksi. Mikropalveluihin ei oletuksena kuulu rekisteriä, joten nämä tarvitsevat oman tapansa linkittää palvelun kutsuja sekä palvelun tarjoaja. Tällaisia ratkaisuja voivat olla vaikkapa selaimen tai palvelinpuolen palveluiden haku. Joka tapauksessa nämäkin ratkaisut päätyvät usein rekisterin käyttöön[Net12]. Ratkaisut loppuviimein ovat hyvin samanlaisia, erona on ettei REST-palvelunhakua ole standardoitu vielä samalla tavalla kuin Web Service maailmassa.

3.2 WSDL

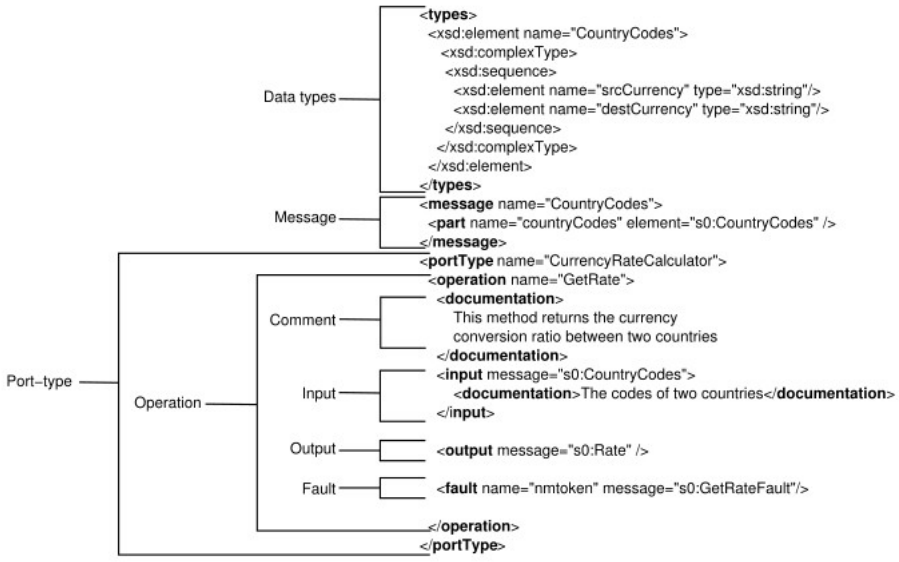
WSDL on kieli jonka avulla kehittäjät kuvaavat kaksi tärkeintä osaa palvelusta: palvelun toiminnallisuuden ja miten sitä kutsutaan, eli miten siihen pääsee käsiksi.

WSDL 1.1 määrittelyn [1] mukaan toiminnallinen kuvaus paljastaa palvelurajapinnan, joka tarjotaan palvelun kutsujalle. Myöhempi osa määrittää teknologiaan liittyvät osa-alueet, kuten viestintäprotokollan sekä verkko-osoitteen. Palvelua hakevat osapuolet tarvitsevat toiminnallisen määrityksen löytääkseen kolmannen osapuolen palvelun joka sopii käyttäjän tarpeisiin ja käyttäjä hyödyntää teknologiallista määritystä viestiäkseen palvelun kanssa.

WSDL dokumentti määrittelee palvelun toiminnallisuuden joukkona porttityyppejä (*port-types*), jotka asettava eri operaatiot joiden kutsuminen perustuu viestien vaihtoon. Viestit tässä yhteydessä tarkoittavat operaatioiden syöttö- ja ulostuloparametreja niitä sen enempää erottelematta. WSDL myös tarjoaa palvelun tarjoajalle mahdollisuuden määritellä virheviestejä (*faults*), joita palautetaan palvelun kutsujalle. WSDL elementtien, kuten porttityyppien, virheviestimääritelmien ja operaatioiden pitää olla yksikäsitteisesti nimetty. Vapaaehtoista on, lisätäänkö elementeille dokumentaatiota kommentteina. Dokumentaation lisääminen WSDL-dokumenttiin on kuitenkin suositeltua. Viestit muodostuvat osista, jotka kuljettavat tietoa palvelun tarjoajan ja palvelujen käyttäjien välillä sekä toisin päin. Tyypillisesti jokainen viesti on hierarkkiselta sisällöltään määriteltyjen datatyyppien mukainen. Nämä datatyypit on määritelty erikseen skeemassa. XSD-kieli (*XML Schema Definition*) on kehitetty kuvaamaan viestin osien hierarkiaa. XSD tarjoaa konstruktorit määrittämään yksinkertaiset tyypit (*simple types*) esimerkiksi integer, string ja totuusarvo. XSD tarjoaa myös työkalut rajoitusten määrittämiseen, kapseloimiseen sekä lisäosa mekanismien kehittämiseen monimutkaisempia elementtejä varten. Kuvassa 1 on esitelty tarvittava XML-muotoinen esitys monimutkaisen tyypin (*complex type*) - tässä "CountryCodes- toteutukseen.

Useimmat rekisterit ovat luonteeltaan syntaktisia, joten on onnistuneen palvelun haun kannalta oleellista, että WSDL-dokumentin elementit on nimetty ja kommentoitu kuvaavasti. Syntaktiset rekisterit esikäsittelevät WSDL-dokumentit tyypillisesti siten, että rekisterit keräävät joukon termejä. Termit valitaan porttityyppielementtien, sisä- ja ulostuloviestien sekä operaatioiden nimistä ja kommenteista. Kun termit on kerätty dokumentista, syntaktiset rekisterit erottelevat nimiyhdistelmät: "CountryCodes"erotetaan ja jää "Country"ja "Codes". Edelleen poistetaan ei-relevantit termit kuten "the", "for"ja vastaavat. Mahdollisesti poistetaan myös ylimääräiset päätteet jotta saadaan termin juuri. Tätä on kutsuttu juuristamiseksi (*stemming*)[CZC08]. Vastaavasti pyynnöt esikäsitellään edellämainitulla tavalla,

[1] WSDL 1.1 www.w3.org/TR/wsdl



Kuva 3: Esimerkki palvelumäärityksestä WSDL:a käyttäen

jonka jälkeen termejä vertaillaan tarjolla olevien palvelujen termejä vastaan. Jos julkaisija ei osaa määritellä WSDL-dokumenttiaan käyttäen järkeviä paradigmoja, kuten selkeitä nimeämiskäytäntöjä, on vertailu tarjolla oleviin palveluihin hyvin vaikea tehtävä.

3.3 UDDI

Perinteisen Web Service mallin palvelut nojaavat yleensä erilaisiin rekistereihin. OASIS kehitti UDDI:n vuonna 2000 siltä ideapohjalta, että palvelua käyttävät tahot linkitettäisiin palvelun tarjoajiin julkisen tai yksityisen rekisterin avulla. Tällaisessa visiossa kuka tahansa esimerkiksi luottokorttitunnistusta tarvitseva asiakas tekisi pyynnön palvelurekisteriä vastaan ja valitsisi oikean SOAP:ia tukevan palvelun. Tällaisessa maailmassa julkisesti ylläpidettävä UDDI-rekisteri olisi kriittisen tärkeä kaikille. UDDI ei kuitenkaan päätynyt niin yleiseen käyttöön kuin sen julkaisivat olisivat toivoneet ja sen kehittäminen loppui vuonna 2007[OAS07]. Se on kuitenkin edelleen osa Web Services Interoperability standardia ja tärkeä osa perinteistä Web Service palvelunhakumallia.

UDDI-rekisteri koostuu kolmesta osasta:

- *Valkoiset sivut* sisältävät tietoa palveluntarjoajasta. Tähän sisältyy esimerkiksi

julkaisijan/yrityksen nimi ja kuvaus, mahdollisesti monella kielellä, ja palvelu voi olla löydettävissä pelkästään tämän tiedon avulla. Valkoiset sivut saattavat sisältää myös palveluntarjoajan yhteystiedot kuten puhelinnumero ja osoite.

- *Keltaiset sivut* luokittelevat palvelut standardoitujen taksonomioiden mukaan.[2] Yhteen valkoiseen sivuun voi liittyä useampi keltainen sivu, eli kuvaukset useammalle palvelulle.

- *Vihreät sivut* sisältävät teknistä tietoa miten palveluun pääsee käsiksi. Sisältävät siis palvelun osoitteen, parametrit ja viitteet vaadittuihin rajapintoihin tai protokolliin joilla palvelu toimii.

UDDI on luonteeltaan syntaktinen rekisteri, eli se esikäsittelee termit ja suorittaa vertailun olemassaoleviin palveluihin niiden avulla.

3.4 Palveluiden haun haasteet

Web service -palveluhaun suurimpia ongelmia on, että se on kuin etsisi neulaa heinäsuovasta[GPST04], varsinkin kun palveluiden määrä kasvaa. Oikean palvelun löytäminen käyttäjän antamien hakuehtojen mukaan on vieläkin perustavanlaatuinen tutkimusongelma SOC-maailmassa[LLL+09].

Useat tekijät haastavat palvelun haun ongelmaa entisestään:

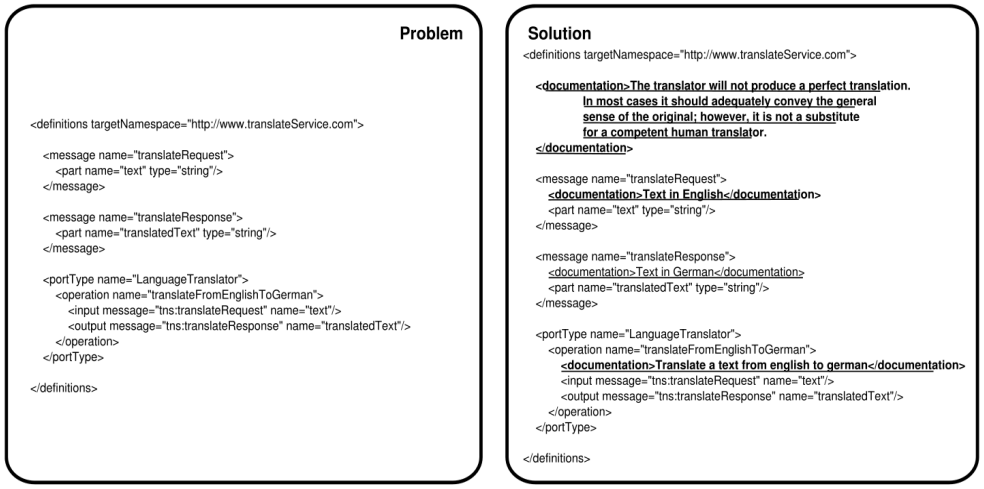
- Julkisten web palveluiden määrä kasvaa tasaisesti koko ajan, samalla kun pilvi- ja SaaS-palvelut ovat kasvattaneet suosiotaan.

- Avainsana-perustainen palvelun haku on ongelmallinen kahdessakin mielessä. Palveluntarjoajat julkaisevat usein puutteellisia palvelukuvauksia jotka eivät sovellu avainsana-hakuun. Lisäksi nykypäivän monimutkaiset liiketoimintatarpeet ylipäätään vaikeuttavat asiakkaan hakusanavalintaa.

- Haku on syntaksiperustainen, joka johtaa semanttisuuden puutteeseen palvelun haussa.

- palvelun tarjoajan välinpitämättömyys ohjelman toiminnan kannalta vähemmän oleellisiin asioihin, kuten palvelun laatuun ja hintaan.

[2]Standardoituja taksonomioita ovat esimerkiksi SIC (Standard Industrial Classification) ja NAICS (North American Industry Classification System)



Kuva 4: Puuttuvat tai virheelliset kommentit -puute ja esitetty korjaus, otettu lähteestä [RCZC10]

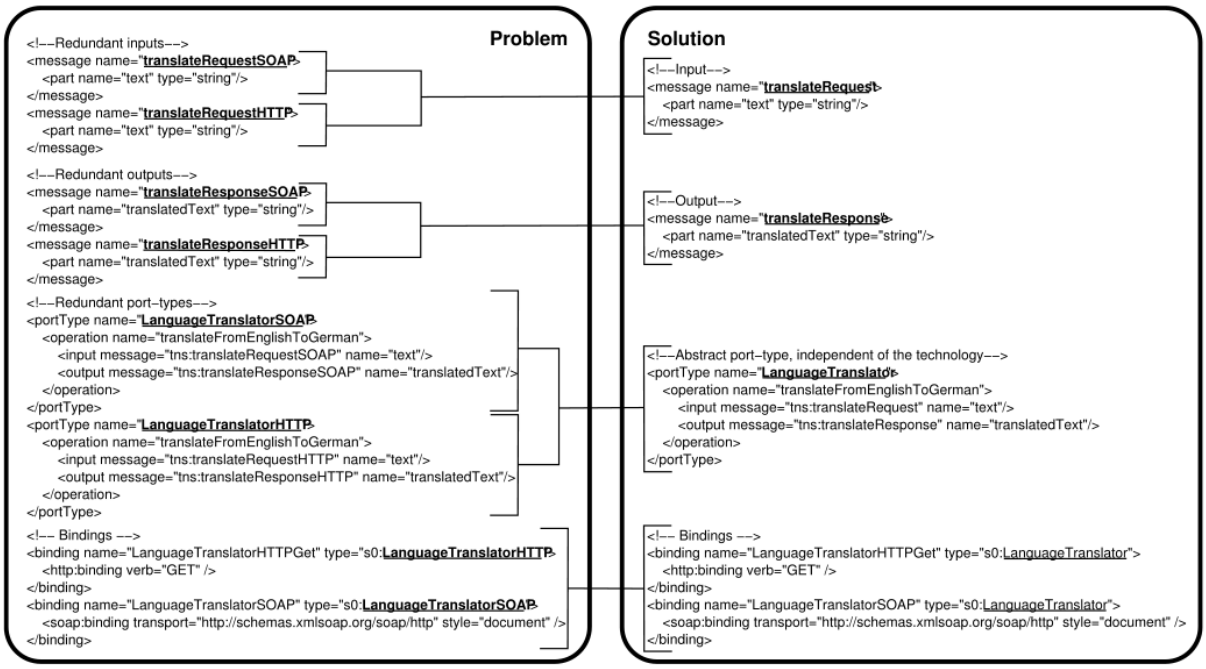
Perinteisen mallin syntaktisuuteen perustuva palvelun haku aiheuttaa helposti ongelmia. Palvelua määritellessään julkaisijat sortuvat helposti XSD sekä WSDL-pohjaisiin puutteisiin määrittelyssä, jotka heikentävät palvelun haun tehokkuutta. Lähteessä [RCZC10] mainitaan kahdeksan puutteellista määrittelytyyliä ja ratkaisuja niihin:

Puuttuvat tai virheelliset kommentit: Puute esiintyy dokumentaatiossa. Ongelma esiintyy silloin, kun WSDL-dokumentti ei sisällä kommentteja, tai ne ovat muodoltaan tai esitykseltään huonoja. Kommenttien tulisi olla itsensäselittäviä ja sijaita oikeassa kohtaa WSDL-dokumenttia. Kuvassa 4 on esitelty puute dokumentoinnissa ja sen vieressä ehdotettu korjaus. Kun tarvittava dokumentaatio on lisätty dokumenttiin, sen lisäksi että dokumentti on ilmaisultaan järkevämpi, kasvaa sen löytövoima samalla, sillä korjaus lisää dokumentista eroteltavien termien määrää. Esimerkiksi nyt saadaan juuret "translat", "term", "english"ja "german", joita syntaktiset rekisterit käyttävät.

Ylimääräiset port-type määritykset: Yleinen huono tapa on toistaa port-type määrityksiä. Turha port-type määritys on sellainen, joka sisältää saman joukon operaatioita kuin jokin toinen saman palvelun port-type määritys. Yleensä lähtökohtana kehittäjällä tämän tyyppisessä toteutuksessa on määritellä eri port-typet operaatioille jotka tarjoavat toimintaansa eri protokollan yli, esimerkiksi HTTP ja SOAP. Tämä ei ole kuitenkaan tarpeellista, sillä operaatiot voidaan abstrahoida, niinkuin on tehty kuvassa 5. Toisaalta, useammat (turhat) port-type määritykset voivat nostaa palvelua suotuisammaksi rekisterin valitessa asiakkaan kyselyn perusteella termejä. Jos termi sattuu osumaan palvelusta esikäsiteltyyn termiin, saa se korkeamman arvosanan. Jos sama termi esiintyy useampaan kertaan esimerkiksi port-type määrityksen toiston takia, saa se edelleen korkeamman arvosanan. Tällainen tilanne voidaan nähdä samanlaisena kuin Google pommi[Wik07].

Monitulkintaiset XML-elementtien nimet: Ilmenee kun monitulkintaisia tai kryptisiä nimiä esiintyy porttityyppi, operaatio tai viestielementeissä. WSDL-elementin tulee ilmentää hyvin tehtäväänsä, sillä nimi on uniikin tunnisteen lisäksi myös kuvauselementti, koska syntaktiset rekisterit esikäsittelevät myös nimielementit. Kuvaavat nimet eivät siis sisällä argumentteja kuten "arg0" tai "foo". Lisäksi syntaktiset rekisterit luottavat usein suosittuihin nimentäkonventioihin pilkkoessaan pitkiä nimiä[DHM+04]. Esimerkiksi nimi "*xmlelementinnimi*" tulee kirjoittaa muodossa "*xmlElementinNimi*". Nämä ovat myös helpommin luettavissa. Myös nimen pituus kannattaa ottaa huomioon. Nimen ei kannata olla liian lyhyt, eikä liian pitkä. Suosituspituus olisi kolmen ja viidentoista merkin välillä[KNB06].

Pienen koheesion omaavat operaatiot samassa port-typessä: Yleinen huonoa suunnittelutyyliä noudattava määrittelyvirhe on asettaa pienen koheesion omaavia operaatioita samaan porttimäärittelyyn. Koheesiolla tarkoitetaan tässä sitä, kuinka voimakkaasti operaatio on liitoksissa muihin saman porttityypin alla sijaitseviin operaatioihin. WSDL-dokumentit, joiden porttimäärittelyt noudattavat korkeaa koheesiota ilmaisevat termeillään lähinnä kyseessä olevan palvelun domainiin liittyviä toimintoja. Kun rekisteriin tehdään syntaktinen kysely, on näillä korkean koheesion omaavilla palveluilla suurempi todennäköisyys tulla löydetyksi. Jos palvelu omaa matalaa koheesiota, on tarpeen eritellä palvelun toiminnan kannalta vähemmän oleelliset operaatiot omiin porttityyppeihinsä, tai kokonaan erillisiksi web palveluiksi koheesion nostamiseksi.



Kuva 5: Ylimääräiset port-type määritykset -puute ja esitetty korjaus, otettu lähteestä [RCZC10]

Suljettu datamalli: Dataa esittävien suljettujen mallien määrittely erikseen jokaisessa palvelun kuvauksessa, jotka tarvitsevat niitä, on myös huonoa suunnittelutyyliä. Suljettu datamalli on datamalli joka sijaitsee WSDL-dokumentissa ja sitä voidaan käyttää vain operaatioissa, jotka on kuvattu kyseisessä WSDL-dokumentissa. Tuotu datamalli taas on malli, joka on eristyksissä palvelukuvauksesta omassa XSD-dokumentissa. WSDL-dokumentista voidaan viitata tähän XSD-dokumenttiin tarvittaessa. Näin tehtynä tulee palvelumäärittelystäkin helpommin ylläpidettävä, koska elementtien uudelleenkäytettävyys paranee ja toiston määrä pienenee.

Ylimääräiset datamallit: Saman datatyypin määrittely useaan kertaan on myös yleinen huonoa suunnittelutyyliä sisältävä tapa. Ylimääräisiä datamalleja esiintyy, jos WSDL dokumentissa vähintään kaksi datatyyppimäärittelyä ovat samaan käyttötarkoitukseen luotuja. Syntaktiselle rekisterille tarpeettomat datamallit voivat aiheuttaa samanlaisia vaikutuksia kuin tarpeettomat porttityypit. Samoin kuin ylimääräiset porttityyppimääritykset, tämä suunnitteluvirhe aiheuttaa tarpeettoman suuria ja vaikeaselkoisia WSDL-dokumentteja asiakkaan, eli palvelua kutsuvan osapuolen, kannalta. Ylimääräiset datamallit -suunnitteluvirhe muodostuu usein vaadittujen datatyyppien huonosta esitysmuodosta. Tämän korjaamiseksi on suotavaa korvata turhat ja ylimääräiset datatyypit yksittäiselle datatyypillä ja tehdä korjauk-

set niihin osiin jotka kutsuivat näitä datatyyppejä osoittamaan uuteen datatyyppiin. Korjatun version ei tulisi sisältää ylimääräistä XSD-koodia, jolloin palvelun kuvaus on ytimekkäämpi ja helpommin ymmärrettävissä myös kolmannen osapuolen palvelunhaulle. Huomioitavaa on myös, että korjattu datamalli kannattaa määritellä omassa XSD-dokumentissaan, jotta vältettäisiin sortumista aiemmin mainittuun suljettuun datamallityyliin.

Turhan laaja tyyppikonteksti datatyypille: WSDL tukee myös määrittelyelementtien nimeämistä yleiskäyttöisillä datatyypeillä. Näiden käyttö on kuitenkin usein turhaa tai erheellistä. Yleensä tällainen liian lavea määrittely esiintyy, jos esitetään datatyyppiä XSD:n "xsd:any"tai "xsd:anyAttribute"konstruktorilla. Alempana on esitetty XSD-määrittely datatyypille nimeltä "DataJoukko", jota voidaan käyttää siirtämään melkein mitä tahansa monimutkaisempaa XML datastruktuuria asiakkaan ja palveluntarjoajan välillä.

```
<xsd:element name="DataJoukko" nillable="true">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="s:schema" />
      <xsd:any />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Ilmentymä DataJoukosta voi sisältää mitä tahansa XML:a, se voi olla jopa tyhjä XML-tiedosto. Jotkin kehittäjät voisivat nähdä tämän vaihtoehtoisena tapana rakentaa laajennettavia operaatioita, se kuitenkin monimutkistaa ja vaikeuttaa XML-koodin ymmärrettävyyttä. Tällainen suunnittelu siis tekee operaatioista vaikeasti ymmärrettäviä myös kolmannen osapuolen palvelun hakijan kannalta. Esimerkiksi otetaan porttityyppimäärittely nimeltään "SatunnaisGeneraattori", joka sisältää yhden operaation jonka signeeraus on "generate():DataJoukko". Nyt tämän operaation signeeraus ja datatyyppi eivät ole yksikäsitteisiä, eikä palvelua hakeva henkilö pysty päättelemään mikä operaation vastaus on muodoltaan ja sisällöltään ennen kuin operaatiota kutsutaan. Tällaiset laajat tyypit eivät yleensä kerro sisällöstään eksplisiittisesti, eikä niistä silloin helposti esikätellä termejä oikein, joka taas vaikeuttaa löydettävyyttä syntaktisissa rekistereissä.
Sen sijaan, otetaan toinen operaatio, jonka signeeraus on "generate:SatunnainenNumero"ja jonka määrittely on alla.

```
<xsd:simpleType name="SatunnainenNumero">
  <xsd:restriction base="xsd:double">
    <xsd:minInclusive value="0" />
    <xsd:maxExclusive value="1" />
  </xsd:restriction>
<xsd:simpleType>
```

Tämän operaation signeerauksesta voidaan päätellä, että operaatio palauttaa yhden satunnaisen luvun. Lisäksi nähdään, että "SatunnainenNumero"laajentaa double-tyyppistä primitiivistä datatyyppiä ja rajoittaa luvun nollan ja ykkösen välille. Ihminenkin pystyy päättelemään (kutsumatta palvelua ensin), että operaatio palauttaa double-tyyppisen desimaaliluvun nollan ja ykkösen väliltä.

Piilevät virheviestit standardiviesteissä:

Ajonaikana ilmenneistä virheistä ilmoittaminen ulostuloviestissä on myös huonoa suunnittelutyyliä. Tällaisessa tyylissä määritellään ulostuloviesti jonka tehtävänä on siirtää tieto virheviestistä kun operaatio epäonnistuu, tai esimerkiksi statuskoodi kun operaatio on päättynyt. Tällaisen ulostuloviestin määrittely vaatii joustavan datatyypin. Tällainen sekä aiheuttaa helposti liian laajaa tyyppimäärittelyä (mainittu aikaisemmin) että monimutkaistaa operaation toimintalogiikkaa. Virheviestien lähettäminen ulostuloviesteissä voi vaikuttaa negatiivisesti syntaktisiin rekistereihin, jotka hyödyntävät XML-elementtien nimiä ja/tai viestiosien XML-rakennetta[CZC08]. Tämä kehno suunnittelutyyli esiintyy usein liian laaja datatyyppi -suunnittelutyylin kanssa. Toinen yleinen esiintymismuoto tälle tyylille vaatii viestimäärittelyn joka koostuu kolmesta osasta: ensimmäinen osa kertoo tapahtuiko virhettä, toinen osa kertoo mikä aiheutti virheen ja tietoja siitä ja kolmas kertoo operaation lopputuloksen. Seuraamalla tätä kolmen osan lähestymistapaa, viestimäärittely operaatiolle joka laskee parametrina annetun epänegatiivisen luvun kertoman (mukailtu lähteestä [RCZC10]) voisi olla seuraavanlainen:

```
<message name="laskeKertomanTulos">
  <part name="onkoVirheita" type="xsd:bool" />
  <part name="virheenAiheuttaja" type="xsd:string" />
  <part name="kertoma" type="xsd:long" />
</message>
```

onkoVirheita-osa kertoo tapahtuiko virhettä. Jos virhe tapahtui, niin viheenAiheuttaja-osa kuvaa virheen aiheuttajan. Jos virhettä ei tapahtunut niin virheenAiheuttaja-

osa on tyhjä. Toisaalta, jos virhe tapahtuu, niin kertoma-osa on tyhjä. Sen lisäksi, että ulostuloviesti lähettää pyydettyä tietoa, se myös yrittää kertoa vastaanottavalle taholle mitä pitäisi tehdä. Tämä voidaan nähdä erikoistapauksena kontrollikytkennästä, jota tulisi välttää strukturoidussa suunnittelutyylissä[YC79]. Palvelun haun kannalta edellämääritelty viesti on ongelmallinen, sillä se antaa syntaktisen rekisterin esikäsittelyn jälkeen ohjelman päätoiminnan kannalta epäoleellisia termejä kuten 'onko', 'virheita', 'virheen' ja 'aiheuttaja'. Ainoastaan 'kertoma' on haluttu termi kuvatessa ohjelman päätoimintoa, eli kertoman laskemista. Ratkaisuna tähän huonoon suunnittelumalliin on käyttää WSDL:n omaa tapaa käsitellä virheelliset tilanteet: virheviestit pitäisi siirtää omissa 'fault'-viesteissään. Yllämääritelty huonoa suunnittelutyyliä noudattava esimerkki voisi näyttää korjausten jälkeen seuraavanlaiselta:

```
<message name="maaritaKertomaVirhe">
  <part name="virheKuvaus" type="xsd:string" />
</message>
<portType name="KertomanLaskija">
  <operation name="laskeKertoma">
    <documentation>Palauttaa kertoman annetulle numerolle
    </documentation>
    <input message="tns:laskeKertomaPyynto" name="number" />
    <output message="tns:laskeKertomaVastaus" name="result" />
    <fault message="tns:laskeKertomaVirhe" name="virheenTiedot" />
  </operation>
</portType>
```

Korjatulla operaatiolla on kolme viestiä: sisääntulo-, ulostulo- ja virheviestit. Nyt sisääntulo- ja ulostuloviestit ainoastaan siirtävät asiakkaan ja palvelimen välillä ja fault-viestit on erikoistettu antamaan palvleun kutsujille kuvaus virheistä. Se voisi olla liitettynä myös monimutkaisempaan tyyppiin, jossa voitaisiin antaa enemmän tietoa virheestä, kuten ohjelman kutsupino (*stacktrace)*. Joka tapauksessa korjattu WSDL-dokumentti on vapaa liian laajoista tyyppimäärityksistä sekä kontrollikytkennästä.

Koostettu kuva suunnitteluvirheistä tämän tutkielman liitteenä.

Tarve paranneltuihin palvelumäärityksiin on huomattu muualla tiedeyhteisössäkin[KTSW08].
Todella paljon tutkimusta on tehty SOC:in palvelun haun ongelmien vastaamiseksi.
Ne voidaan jakaa neljään eri luokkaan[AN10].

1. **Semanttiset tavat**: Semanttisen palveluhaun mallissa kyse on yksinkertaisuudessaan palvelujen vertailusta. Semanttiset lähestymistavat pyrkivätkin lisäämään vertailun tehoa, yleensä rikastuttamalla palvelukuvauksia jollakin ontologialla, esimerkiksi OWL-S. Ontologialla rikastetut kuvaukset tuovat lisää ilmaisua ja parantavat vertailua käyttäjän tarpeen ja tarjolla olevien palvelujen välillä.

2. **Informaation haku tavat**: Nämä lähestymistavat tutkivat semanttista suhdetta palvelua kutsuvan osapuolen pyyntöjen välillä sekä palvelun tarjoajan määrittämien termien välillä[MCZ07].

3. **Data mining tavat**: Nämä lähestymistavat esittävät palvelun haun ongelman nk. Constraint Satisfaction -ongelmana[Tsa14] (*CSP*). Tehty pyyntö verrataan joukkoon saatavilla olevia palveluita ja reitinhakualgoritmeilla sekä klusterointimetodeilla pyritään ratkaisemaan CSP. Palveluiden suhdetta palveluiden välillä käytetään siis parantamaan käyttäjälle tarjottujen palveluiden tarkkuutta.

4. **Linkittämiseen liittyvät tavat**: Linkittämiseen liittyville lähestymistavoille tyypillistä on, että niissä pyritään yhdistämään asiakkaan tarpeet tyydyttävä palvelu yhdeksi kokonaisuudeksi useista eri palveluista[RS05].

Haasteeksi perinteisessä WSDL:n, rekisteriin ja SOAP:aan keskittyvässä mallissa on tullut semanttisuuden puute[MBM+07], sillä WSDL kertoo kyllä miten palvelua käytetään, mutta ei tarjoa ratkaisuja siihen mitä tapahtuu kun palvelua käytetään. Palvelua käyttävä ihminen lukee sen luonnollisella kielellä palvelukuvauksesta. Jos halutaan automatisoituja palveluja, jotka osaavat luotettavasti ratkaista ongelmia, tarvitaan tapa jättää ihminen välistä. Tarvitaan siis jokin ratkaisu tuottamaan semanttisia palvelukuvauksia. Palveluiden haku on luonteeltaan semanttista, mutta perinteinen UDDI,SOAP,WSDL perustuvan mallin rajoitteena on kertoa mitä kaikkea palvelu pystyy tekemään. Malli on luonteeltaan syntaksipohjainen, joka ei pysty itsessään ilmaisemaan koko kontekstia, jossa palvelut toimivat ja mitä ne pystyvät tekemään.

UDDI:n avulla kyselyt palauttavat usein vääriä palveluita[BCP05], varsinkin kun palveluiden määrä rekisterissä kasvaa.

Erilaisia lähestymistapoja asian korjaamiseksi on esitetty, josta lisää kohdassa "Semanttinen palveluiden haku".

3.5 Semanttinen palveluiden haku

Palveluiden haku myös semanttisen webin saralla on perustavanlaatuinen tutkimusongelma. Semanttiset palvelut ovat visio arkkitehtuurista, jossa käyttäjällä on mahdollisuus automaattisesti yhdistää tarvittavia palveluita toteuttaakseen jokin tietty tehtävä tai tehtäväkokonaisuus. Olkoon käyttäjä "Simo". Simo haluaa löytää halvan lennon määränpäähänsä josta haluaa jatkaa hotellille joka tyydyttää Simon tarpeet hinnaltaan ja sijainniltaan. Semanttisten webpalveluiden visiona on yhdellä pyynnöllä yhdistellä palveluita, jotka antavat Simolle vastauksen haluttuun ongelmaan. Simo voisi hyvinkin olla myös muu e-Business palvelu, joka yhdistelee palveluita saadakseen ratkaisuja ongelmiinsa.

WSDL, SOAP ja UDDI -standardit kärsivät semanttisen esityksen puutteesta, joten toive alan standardeilla koostettujen palveluiden automaattisesta integroimisesta jää täyttymättä. Jotta palveluista saataisiin kehitettyä semanttisia, on haasteita joihin täytyy löytää ratkaisuja. Ensinnäkin kun palveluiden määrä lisääntyy merkittävästi, ja oikeiden palveluiden löytäminen on tärkeää, palvelunhaun tehokkuus ja tarkkuus tulee olla merkittävästi parempi kuin nyt. Toisekseen, kun palvelu saadaan tehokkaasti löydettyä, pitäisi pystyä automaattisesti linkittämään se kyseistä palvelua kutsuvaan palveluun. Molemmat näistä haasteista ovat riippuvaisia palveluntarjoajan kyvystä kuvata tarjoamansa palveluiden toiminnot, sekä palveluita kutsuvien osapuolten kyvystä määritellä yksikäsitteisesti ja koneluettavasti tarpeensa palvelulta.

Eräs tapa lähestyä näitä haasteita, eli perinteisen mallin semanttisuuden lisäämistä on lisätä laajennus UDDI rekisteriin[AGDR03, SPS04] lisäämällä siihen OWL-S ontologia.

4 Yhteenveto

Tässä tutkielmassa ollaan keskitytty palveluiden haun toteutukseen palveluorientoituneissa arkkitehtuureissa. Tutkielmassa ollaan tutustuttu miten palveluiden haku on perinteisesti toteutettu käyttäen WSDL,SOAP,UDDI -teknologioita ja miten palveluiden haun tarkkuutta voidaan parantaa, ja millaisiin sudenkuoppiin voidaan palvelumäärityksissä astua. Sen lisäksi tutkielmassa mainitaan mitä aiheesta on tutkittu ja millaisia päätelmiä niistä voidaan tehdä.

Voidaan todeta, että yksinkertaisenkin palvelun määrittely voi vaikeuttaa perinteis-

ten syntaktisten rekistereiden löytämistä, jos palvelu on WSDL-määritelty huonosti. Syntaktisten rekistereiden toiminta on tärkeä olla tiedossa suunnitellessa helposti löydettäviä web-palveluita.

Lähteet

AGDR03 Akkiraju, R., Goodwin, R., Doshi, P. ja Roeder, S., A method for semantically enhancing the service discovery capabilities of uddi. *IIWeb*, 2003, sivut 87–92.

AN10 AbuJarour, M. ja Naumann, F., Dynamic tags for dynamic data web services. *Proceedings of the 5th International Workshop on Enhanced Web Service Technologies*, WEWST '10, New York, NY, USA, 2010, ACM, sivut 3–9, URL `http://doi.acm.org/10.1145/1883133.1883135`.

BCP05 Brogi, A., Corfini, S. ja Popescu, R., Composition-oriented service discovery. *Software Composition*. Springer, 2005, sivut 15–30.

Cha14 Chaturvedi, A., Subset wsdl to access subset service for analysis. *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, Dec 2014, sivut 688–691.

CZC08 Crasso, M., Zunino, A. ja Campo, M., Easy web service discovery: A query-by-example approach. *Science of Computer Programming*, 71,2(2008), sivut 144 – 164. URL `http://www.sciencedirect.com/science/article/pii/S0167642308000282`.

DHM+04 Dong, X., Halevy, A., Madhavan, J., Nemes, E. ja Zhang, J., Similarity search for web services. *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, sivut 372–383.

GPST04 Garofalakis, J., Panagis, Y., Sakkopoulos, E. ja Tsakalidis, A., Web service discovery mechanisms: Looking for a needle in a haystack. *International Workshop on Web Engineering*, osa 38, 2004.

KNB06 Kahan, D., Nowlan, M. ja Blake, M., Taming web services in the wild. *Web Services, 2006. ICWS '06. International Conference on*, Sept 2006, sivut 957–958.

KTSW08 Kuropka, D., Tröger, P., Staab, S. ja Weske, M., *Semantic service provisioning*. Springer Science & Business Media, 2008.

LLL+09 Li, Q., Liu, A., Liu, H., Lin, B., Huang, L. ja Gu, N., Web services provision: Solutions, challenges and opportunities. *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*. ACM, 2009, sivut 80–87.

MBM+07 Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D., Sirin, E. ja Srinivasan, N., Bringing semantics to web services with owl-s. *World Wide Web*, 10,3(2007), sivut 243–277. URL `http://dx.doi.org/10.1007/s11280-007-0033-x`.

MCZ07 Ma, J., Cao, J. ja Zhang, Y., A probabilistic semantic approach for discovering web services. *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, sivut 1221–1222.

Net12 Netflix, Netflix and EUREKA service discovery, `http://techblog.netflix.com/2012/09/eureka.html`, 2012. [Online].

OAS07 OASIS, Closure of OASIS UDDI Specification, `https://lists.oasis-open.org/archives/uddi-spec/200807/msg00000.html`, 2007. [Online].

Pap03 Papazoglou, M., Service-oriented computing: concepts, characteristics and directions. *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, Dec 2003, sivut 3–12.

RCZC10 Rodriguez, J. M., Crasso, M., Zunino, A. ja Campo, M., Improving web service descriptions for effective service discovery. *Science of Computer Programming*, 75,11(2010), sivut 1001 – 1021. URL `http://www.sciencedirect.com/science/article/pii/S0167642310000134`. Special Section on the Programming Languages Track at the 23rd {ACM} Symposium on Applied ComputingACM {SAC} 08.

RS05 Rao, J. ja Su, X., A survey of automated web service composition methods. Teoksessa *Semantic Web Services and Web Process Composition*, Springer, 2005, sivut 43–54.

SPS04 Srinivasan, N., Paolucci, M. ja Sycara, K., Adding owl-s to uddi, implementation and throughput. *Proceedings of Semantic Web Service and Web Process Composition*, 25.

Tsa14 Tsang, E., *Foundations of Constraint Satisfaction: The Classic Text*. BoD–Books on Demand, 2014.

W3C07a W3C, SOAP Version 1.2, `http://www.w3.org/TR/soap12/`, 2007. [Online].

W3C07b W3C, Web Services Description Language (WSDL) Version 2.0, `http://www.w3.org/TR/wsdl20-primer/`, 2007. [Online].

Wik07 Wikipedia, Google Pommi, `https://fi.wikipedia.org/wiki/Google-pommi`, 2007. [Online].

YC79 Yourdon, E. ja Constantine, L. L., *Structured design: Fundamentals of a discipline of computer program and systems design*. Prentice-Hall, Inc., 1979.

Liite 1. Yleiset WSDL-määrittelypuutteet

Yleisimmät WSDL-määrittelypuutteet. Otettu lähteestä [RCZC10].

Anti-pattern	Concern	Problem	Manifest	Suggested solution
Inappropriate or lacking comments	Documentation	Occurs when: (1) a WSDL document has no comments, or (2) comments are inappropriate and not explanatory.	(1) Evident, or (2) Not immediately apparent	Create explanatory comments and place them in the correct part of the WSDL document.
Ambiguous names [9]	Documentation	Occurs when ambiguous or meaningless names are used for denoting the main elements of a WSDL document.	Not immediately apparent	Replace ambiguous or meaningless names with representatives names.
Redundant port-types	Design	Occurs when different port-types offer the same set of operations.	Evident	Summarize redundant port-types into a new port-type.
Low cohesive operations in the same port-type	Design	Occurs when port-types have weak semantic cohesion.	Not immediately apparent	Divide non-cohesive operations into different port-types.
Enclosed data model	Representation	Occurs when the data-type definitions used for exchanging information are placed in WSDL documents rather than in separate XSD ones.	Evident	Move data-type definitions from WSDL documents to XSD files.
Redundant data models	Representation	Occurs when many data-types for representing the same objects of the problem domain coexist in a WSDL document.	Evident	Summarize redundant data-types into a new data-type.
Whatever types [23]	Representation	Occurs when a special data-type is used for representing any object of the problem domain.	Evident	Replace Whatever types with data-types that properly represent the required objects.
Undercover fault information within standard messages [24]	Design	Occurs when output messages are used to notify service errors.	Present in service implementation	Use fault messages for conveying error information.