hyväksymispäivä arvosana

arvostelija

Palveluiden haku palveluorientoituneessa tietojenkäsittelyssä

Toni Könnilä

Helsinki 15.12.2015

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Abstrakti

Service-oriented computing (SOC) on ajatusmalli, jossa yhden monoliittisen ohjelman sijaan keskitytään keskenään kommunikoiviin palveluihin. SOC nojaakin hyvin vahvasti palveluorientoituneeseen arkkitehtuurimalliin (myöhemmin SOA), joka ajaa tätä ideaa[1]. SOA ei kuitenkaan tarjoa itsessään vastauksia siihen miten turvallisuus, palveluiden koordinointi ja ohjelman muu arkkitehtuuri on rakennettu.

Tässä aineessa keskitytään palveluorientoituneiden arkkitehtuureiden yhteisiin piirteisiin ja erityisesti palveluidenhakuun (*service discovery*), ja siihen liittyen, palveluiden läpinäkyvyyteen (*location transparency*).

2 Johdanto

SOC:in 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:issa (*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 (olipa 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 haettavuuteen ja siihen miten se voidaan toteuttaa palveluorientoituneessa arkkitehtuurimallissa.

3 Palveluorientoitunut arkkitehtuuri

Perinteisesti SOC:in 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 yksittinen 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 on 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 1.1) 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 (viite) 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 [viite papazoglouhun]. Tästä syystä onkin kehitelty xSOA (extended SOA) pyramidi [oma liite tähän].

4 Palveluorientoituneiden arkkitehtuurien haittapuolet

5 Palveluiden haku

5.1 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.

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 [viite: Subset WSDL to access Subset Service for Analysis Animesh Chaturvedi], joka kertoo kutsuvalle taholle miten pyydettyä palvelua voidaan kutsua, millaisia parametreja se odottaa, missä muodossa vastaus tulee ja millaisissa tietorakenteissa vastauksen sisältö on [viite: http://www.w3.org/TR/wsdl20-primer]. 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 . 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 pal-

veluun. 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 2.1.

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 (citation needed). 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 (esimerkiksi Netflix ja EUREKA viite). Ratkaisut loppuviimein ovat hyvin samanlaisia, erona on ettei REST-palvelunhakua ole standardoitu vielä samalla tavalla kuin Web Service maailmassa.

5.2 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 palvelurajapin-

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

nan, 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*). (viite query by example setti). Vastaavasti pyynnöt esikäsitellään edellämainitulla tavalla, 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.

5.3 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 [https://lists.oasis-open.org/archives/uddi-spec/200807/msg00000.html]. 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.

5.4 Palveluiden haun haasteet

Web service -palveluhaun suurimpia ongelmia on, että se on kuin etsisi neulaa heinäsuovasta[5], varsinkin kun palveluiden määrä kasvaa. Oikean palvelun löytä-

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

minen käyttäjän antamien hakuehtojen mukaan on vieläkin perustavanlaatuinen tutkimusongelma SOC-maailmassa[6].

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.

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[7] 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.
- Monitulkintaiset XML-elementtien nimet:
- Ylimääräiset port-type määritykset:
- Pienen koheesion omaavat operaatiot samassa port-typessä:
- Suljettu datamalli:
- Ylimääräiset datamallit:
- Turhan laaja tyyppikonteksti datatyypille:
- Piilevät virheviestit standardiviesteissä:

Koostettu kuva edellisistä tämän tutkielman liitteenä.

Tarve paranneltuihin palvelumäärityksiin on huomattu muualla tiedeyhteisössäkin[4]. Todella paljon tutkimusta on tehty SOC:in palvelun haun ongelmien vastaamiseksi. Ne voidaan jakaa neljään eri luokkaan[8].

1. **Semanttiset lähestymistavat**: 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 -lähestymistavat**: Nämä lähestymistavat tutkivat semanttista suhdetta palvelua kutsuvan osapuolen pyyntöjen välillä sekä palvelun tarjoajan määrittämien termien välillä[9].

3. **Data mining -lähestymistavat**: Nämä lähestymistavat esittävät palvelun haun ongelman nk. "Constraint Satisfaction-ongelmana[10] (*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 lähestymistavat**: Linkittämiseen liittyville lähestymistavoille tyypillistä on, että niissä pyritään yhdistämään asiakkaan tarpeet tyydyttävä palvelu yhdeksi kokonaisuudeksi useista eri palveluista[11].

Haasteeksi perinteisessä WSDL:n, rekisteriin ja SOAP:aan keskittyvässä mallissa on tullut semanttisuuden puute[12], 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[3], varsinkin kun palveluiden määrä rekisterissä kasvaa.

Erilaisia lähestymistapoja asian korjaamiseksi on esitetty. Esittelyt löytyvät kohdasta "Semanttinen palveluiden haku".

5.5 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 web palveluiden 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[2].

6 Yhteenveto

Lähteet

2 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.

8 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`.

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

5 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.

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

6 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.

12 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`.

9 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.

1 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.

7 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.

11 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.

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

Liite 1. Malli ABC

Liitteet ovat tässä vain sisällysluettelon ja esitystavan mallina. Jokainen liite aloitetaan yleensä uudelta sivulta, jonka alkuun tulee liitteen numero ja nimi. Kunkin liitteen sivut numeroidaan erikseen.

Liite on paitsi dokumenttia täydentävä osuus myös itsenäinen kokonaisuus. Liite ei siten voi olla pelkästään kuva tai ohjelmanpätkä, vaan liitteessä on ilmaistava sen sisällön laatu ja tarkoitus.