Esimerkkiotsikko

Eija Esimerkki

Seminaariraportti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 25. syyskuuta 2013

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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

Tekijä — Författare — Author
Eija Esimerkki

Työn nimi — Arbetets titel — Title
Esimerkkiotsikko

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Seminaariraportti	25. syyskuuta 2013	17

Tiivistelmä — Referat — Abstract

Tiivistelmä.

Avainsanat — Nyckelord — Keywords
avainsana 1, avainsana 2, avainsana 3

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

World Wide Web (WWW) tarjoaa staattisen informaation lisäksi palveluita. Web-palvelu voi yksinkertaisimmillaan palauttaa informaatiota pyynnön saatuaan tai monimutkaisemmassa tapauksessa voidaan tehdä esimerkiksi ostos, jossa veloitetaan luottokorttia ja muodostetaan toimitettava tilaus[5].

Ohjelmoijan on varsin helppo tuottaa ohjelma, joka käyttää web-palvelua. Tekniikoita palveluiden käyttämiseen on monia, mutta yleinen tapa on lähettää palvelulle viesti, joka on koodattu sovitulla tavalla, esimerkiksi XML:llä tai Jsonilla. Erilaiset datatyypit ja viestien (ja niiden osien) muodot voidaan selostaa esimerkikisi WSDL-dokumentissa [2]. Edellä mainittujen tekniikoiden toteuttaminen ei nykyaikaisilla ohjelmistokirjastiolle ja IDE:illä ole edes monimutkaista. Mutta olipa tekniikka mikä tahansa, tässä muodossa palvelun käyttö vaatii aina *ihmisen* puuttumisen asiaan. *ihminen* on se, joka hakee palvelun. *ihminen* on se, joka tutkii palvelun kuvauksen ja muokkaa oman ohjelmansa (tai palvelun) käyttämään sitä.

Mutta mitä *emme* voi löytää WSDL-kuvauksesta, on se, mitä palvelussa tapahtuu, kun sitä käytetään[5]. Jotta voidaan välttää ihmisen osallistumien web-palvelujen orkestrointiin, tulee olla *automaattisia ohjelmistoagentteja*, jotka etsivät oikeat palvelut ja käyttävät niitä automaattisesti, ilman ihmisen puuttumista asiaan[5].

Automaattinen ohjelmistoagentti mahdollistaisi esimerkiksi:

- *Presentaation pitäjä haluaa lähettää esitysmateriaalinsa kaikkien osanottajien käyttöön. Hän voisi tehdä sen yhdellä lähetysnapin painalluksella riippumatta siitä, mitä teknologioita ja protokollia vastaanottajat käyttävät*[5].

- *Kuluttaja haluaa löytää haluamansa tuotteen mahdollisimman halvalla jostain verkkokauasta. Ohjelmistoagenttia hyödyntävä ohjelma hakee tuotteen hintaa ja saatavuutta useista verkkokaupoista, jotka ovat koodanneet kataloginsa jotain standardoitua sanastoa käyttäen. Sanastojen ei edes tarvitse olla samat eri verkkokaupoilla*[5].

Ohjelmistoagenteille tulisi siis pystyä antamaan tarvittava informaatio muodossa, jota se ymmärtää. *Semanttinen web* voi tarjota ratkaisun tähän. Vuonna 2001 Tim Berners-Lee, James Hendler ja Ora Lassila julkaisivat artikkelin "the Semantic Web", jossa he luonnehtivat semanttista webiä seuraavasti:

> "Semanttinen web ei ole erillinen web vaan laajennos tämänhetkiseen webiin, jossa informaatiolle on annettu hyvin muotoiltu merkitys mahdollistaen koneiden ja ihmisten paremman yhteistyön.[suomennos kirjoittajan][1]

Ohjelmistoagenttien tulee pystyä *ymmärtämään mitä* palveluntarjoaja tarjoaa ja *miten* palvelu tuotetaan. Tunnetusti kone ei ymmärrä ihmisen ymmärtämää teksti-informaatiota. Koneelle pitää siis tarjota mahdollisuus ymmärtää käsitteitä, antaa mahdollisuus luoda uutta ymmärrystä käsitteiden ja käsitteiden välisten suhteiden pohjalta. Koneen tulee pystyä ymmärtämään *semantiikoita*.

Webissä semantiikoita ilmaistaa ontologioilla, jotka tietojenkäsittelytieteessä ymmärretään dokumentteina, joissa kerrotaan asioiden välisistä yhteyksistä[1]. Kun web-palveluita kuvataan ontologioiden avulla, tulee käytössä olla kuvaamiseen soveltuva *sanasto*, ontologia. Yksi mahdollinen tällainen sanasto on OWL-S, joka tulee sanoista Web Ontology Language for Services. OWL-S tarjoaa luokat ja ominaisuudet, joiden avulla web-palvelu voidaan kuvata koneen ymmärtämässä muodossa. Jotta voimme ymmärtää OWL-S:llä tuotettuja ontologioita, pitää ensin tutustua soveltuvin osin OWL (Web Ontology Language) :ään ja sen kostruktioihin, joiden avulla web-palveluontologioita voidaan muodostaa.

2 Web-ontologiakieli OWL

World Wide Web Consortium on antanut OWL:lle suosituksen standardiksi vuonna 2004[7]. Uudempi suositus on OWL 2:lle vuodelta 2012. OWL kielenä perustuu muutamiin jo aiemmin määriteltyihin merkintätapoihin ja konsepteihin, kuten XML:ään ja RDF:ään ja URI:in .

2.1 URI, XML ja RDF

Semanttisessa webissä määritettyjä luokkia, ilmentymiä ja niiden välisiä suhetita identifioidaan URI:en avulla[1]. URI (Uniform Resource Identifier) on standardi resurssien identifioimiseen. Useimmiten URI:na käytetään tavallista URL:ia (Uniform Resource Locator). Identifiointi on tärkeää, koska näin pystytään erottelemaan jo luodut resurssit itse luoduista resursseista.

XML-kieli on notaatio rakenteisen kielen esittämiseen. Useimmiten semanttisessa webissä tietoa kuvataan nimenomaan XML-tiedostoina, koska ne ovat helposti koneen tulkittavissa. XML:n nimiavaruudet helpottavat resurssien identifioimista URI:en avulla.

RDF (Resource Description Framework) on yksinkertainen tapa kuvailla webissä olevaa tietoa kolmikoiden avulla[3]. RDF-kolmikon muodostavat subjekti, objekti ja predikaatti[3]. Subjekti on asia, jota kuvataan, predikaatti on ominaisuus, jolla asiaa kuvataan ja objekti on ominaisuuden arvo. Esimerkiksi tieto ”Matin veli on Teppo” voidaan kuvata kolmikolla

ESIMERKKI

Esimerkistä voimme havaita, että kuvailtava asia on subjekti nimeltään *matti*, jota kuvataan suhteella *veli* joka saa arvokseen *teppo*. Tämänkaltainen XML-kielinen notaatio on OWL:n (ja OWL-S:n) ontologioiden kuvaamisen

syntaktinen ydin. Pelkkä RDF -syntaksi ei kuitenkaan riitä, koska ontologioiden (semantiikoiden) kuvaamiseen tarvitaan myös monimutkaisempia abstraktioita, kuten *luokka*, *aliluokka*, *ilmentymä*, *suhde* ja *alisuhde*. Nämä konstruktiot tarjoaa RDF:n semanttinen laajennos RDF Schema (RDFS) sekä OWL, joka käyttää suurta osaa RDFS:n konstruktioita lähes sellaisenaan. Seuraavassa kappaleessa käyn läpi pintapuolisesti RDFS:n ja OWL:n konstruktiot, jotka auttavat ymmärtämään OWL-S -ontologian rakenteen.

2.2 OWL -ontologiat

OWL(ja RDFS) mahdollistaa ontologioiden muodostamisen. Tässä kappaleessa käsitellään OWL:n peruskonstruktiot, joiden avulla OWL-S:n ontologiat ovat ymmärrettävissä. OWL käyttää useita RDFS:n konstruktioita, jotka tunnistaa etuliitteestä `rdfs:`.

OWL:lla on mahdollista määrittää resurssien joukkoja luokiksi (*class*). Luokalla voi olla aliluokkia(*subclass*) ja luokkien jäsenet ovat ilmentymiä (*instance*). Ilmentymiä voidaan kuvata *suhteilla*, joilla voi olla myös *alisuhteita*[7]. Seuraavassa polkupyörän valmistajaa kuvaavassa esimerkissä käydään läpi ontologian peruskonstruktioita.

Resurssi *Surly* kuuluu luokkaan *Polkupyöräpaja*. Luokkaan kuuluminen ilmaistaan `rdf:type` -konstruktiolla:

```
<owl:Thing rdf:ID="Surly"/>

<owl:Thing rdf:about="Surly">
    <rdf:type= rdf:resource="Polkupyorapaja"/>
</owl:Thing>
```

Polkupyorapaja on luokan *Liikeyritys* aliluokka ja ilmaistaan konstruktiolla `subClassOf`:

```
<owl:Class rdf:ID="Polkupyöräpaja">
    <rdfs:subClassOf rdf:resource="Liikeyritys"/>
</owl:Class>
```

Resursseja kuvataan *ominaisuuksilla*. Ominaisuuksilla voi olla *aliominaisuuksia*. Ominaisuus voi viitata joko olioon (luokkaominaisuus) tai arvoliteraalin (datatyyppiominaisuus). Ominaisuuden sovellusalue ilmaistaan `rdfs:domain` -konstruktiolla ja arvojoukko `rdfs:range` -konstruktiolla. Seuraavassa esimerkissä määritellään luokan *Polkupyorapaja* luokkaominaisuus *Sijaitsee*, joka voi saada arvoksi luokan *Kaupunki* ilmentymiä.

```
    <owl:ObjectProperty rdf:about="Sijaitsee">
        <rdfs:domain rdf:resource="Polkupyorapaja" />
        <rdfs:range rdf:resource="Kaupunki" />
    </owl:ObjectProperty>
```

*Polkupyöräpaja*lle voidaan määritellä myös datatyyppiominaisuus *postinumero*, joka voi saada arvokseen XMLSchemassa määritetyn arvon *integer*:

```
<owl:DataTypeProperty rdf:ID="Postinumero">
    <rdfs:domain rdf:resource="Polkupyorapaja"/>
    <rdfs:range  rdf:resource="&xsd;integer"/>
<owl:dataTypeProperty>
```

Samoin kuin luokalla voi olla aliluokka, ominaisuudella voi olla aliominaisuus. Esimerkiksi ominaisuus `Sijaitsee` voi olla ominaisuuden `Yritystieto` aliominaisuus. Aliominaisuus määritellään `subPropertyOf`-konstruktiolla:

```
<owl:ObjectProperty rdf:ID="Sijaitsee">
    <rdfs:subPropertyOf rdf:resource="yritystieto">
    . . .
<owl:ObjectTypeProperty>
```

Edellä on lyhyin esimerkein käyty läpi OWL:n peruskonstruktioita. OWL tarjoaa myös lukuisan joukon muita konstruktioita, jotka mahdollistavat monipuolisten ontologioiden muodostamisen. Seuraavassa listassa muutamia OWL-S -määrittelyissä vastaan tulevia:

- **kardinaalisuusrajoitteet**. Ominaisuudelle voidaan määrätä kardinaalisuusrajoitteita tietyissä sovelluskonteksteissa. Voidaan esimerkiki määritellä, että luokka *Polkupyörä* on joukko asioita, jolla on tasan kaksi *Renkaat* ominaisuutta.
- **kompleksiset luokat**. OWL mahdollistaa joukko-opilliset operaatiot luokkia muosostettaessa. Voidaan esimerkiksi määritellä, että luokka *Polkupyörä* on luokkien *maastopyörä* ja *maantiepyörä* edustajien yhdiste, unioni. Owl mahdollistaa myös ilmentymäjoukkojen leikkaukset ja komplementit.

3 Webpalveluiden kuvauskieli OWL-S

OWL-S on yksi tapa kuvata web-palveluita siten, että myös kone ymmärtää kuvauksen[4]. OWL-S:llä kuvattu palvelu mahdollistaa mm. automaattisen palveluiden löytämisen, automaattisen palveluiden käytön, sekä useiden palveluiden automaattisen yhdistämisen ja yhteiskäytön[4]. OWL-S tarjoaa luokka- ja ominaisuusrakenteen, jonka avulla palvelukuvaus voidaan muodostaa. Seuraavissa kappaleissä kaydään läpi ontologian karkean tason rakenne sekä sekä eri osa-alueet. Kun dokumentissa tästä eteenpäin käytetään käsitteitä *luokka*, *aliluokka*, *ominaisuus* jne., tarkoitetaan aina OWL:n vastaavia konstruktioita.

3.1 Ontologioiden muodostamisesta

Jotta ontologioita voidaan muodostaa, pitää käytössä olla sanastoja, jotka ovat itsekin ontologioita. Sanasto tarjoaa käsitteet, joiden avulla voidaan luoda omia ontologoita. OWL-S:ssä sanaston muodostaa karkeasti ottaen neljä OWL-ontologiaa: *Service.owl*, *Profile.owl*, *Process.owl* ja *Grounding.owl*.

Näissä ontologioissa on määriteltynä suuri osa luokista ja ominaisuuksista, joita OWL-S -ontologian määrittelemiseen tarvitaan. Nämä käsitteistöt sitten tuodaan oman määrityksen käyttöön OWL:n `import` -lauseilla. Em. ontologioiden seikkaperäinen läpikäynti ei ole tässä kohtaa järkevää, mutta suositeltavaa on tutustua niihin esimerkiksi *daml.org*:in esimerkkien avulla[6].

Tässä raportissa olevat esimerkit ovat *daml.org*:ista poimittuja ja käsittelevät kuvitteellista lentoyhtiötä ja sen lennonvarauspalvelua [6]. Lainatut esimerkit eivät muodosta yhtenäistä ja toimivaa ontologiaa, vaan ne on lainattu selkiyttämään raportin joitain kohtia.

3.2 Korkean abstraktiotason rakenne

Palveluontologian korkean abstraktiotason rakenne muodostuu kolmen tyyppisestä tiedosta ja ne vastaavat kolmeen eri kysymykseen [4]:

- *Mitä palvelu tarjoaa mahdolliselle asiakkaalle?* Tähän antaa vastauksen ontologian *profiili*, joka kertoo karkealla tasolla mitä palvelu tarjoaa. Profiilin avulla palveluntarjoaja voi mainostaa palveluaan potentiaalisille asiakkaille. Profiilissa kerrotaan myös, *kuka* palvelun tarjoaa. Jokainen `Service`-luokka edustaa yhtä `ServiceProfilea` [4].

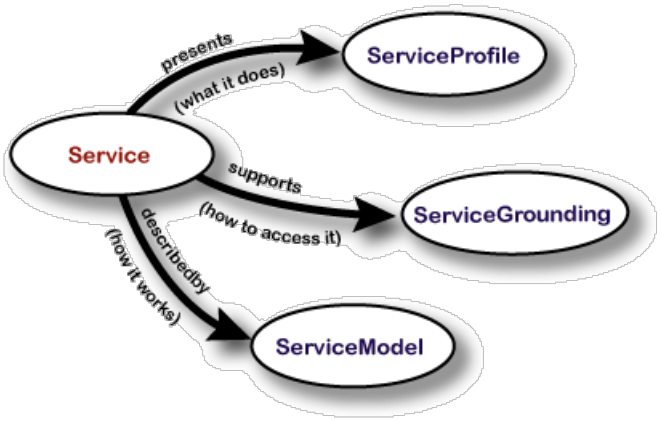
- *Kuinka palvelua käytetään?* Ontologian *prosessimalli* antaa vastauksen tähän ja se esitetään luokassa `ServiceModel`. Palvelun ja sen prosessimallin välillä on `describedBy` -ominaisuus[4].

- *Miten palvelun kanssa kommunikoidaan?* Tähän antaa vastauksen ontologian *maadoitus*, jossa määritellään esimerkiksi tuki erilaisille viestiprotokollille. `Service` -luokalla on ominaisuus `supports`, joka viittaa `ServiceGrounding` -luokkaan[4].

Kuten kuvasta 1 voidaan nähdä, jokaista julkistettua palvelua kohden on yksi `Service`-luokan ilmentymä, joka edustaa `ServiceProfile`-luokkaa ominaisuudella `presents`, on `ServiceModel`-luokan kuvailema ominaisuudella `describedBy` ja `serviceGrounding`-luokan tukema ominaisuudella `supports`.

3.3 Profiili

Profiili siis kertoo mitä palvelu tekee ja kuka palvelun tarjoaa. Se mahdollistaa asiakkaita (agentteja) löytämään palvelun parinlöytöalgoritmien



Kuva 1: OWL-S:llä kuvatun palveluontologian korkean taon rakenne [4]

(matchmaking-algorithm) avulla. Tämänhetkinen teollinen standardi palvelujen etsimiseen on ollut UDDI (Universal Description Discovery and Integration), mutta sen ongelmana on ollut palvelujen monipuoliseen kuvaamiseen kykenevän kielen puute. On kuitenkin mahdollista yhdistää UDDI:n rekisterit ja OWL-S -kuvaukset erityisillä integroiduilla parinlöytöalgoritmeilla[**?**].

Profiili tarjoaa kolmenlaista informaatiota [4]:

1. *Tuottajainformaatio* kertoo tietoja palvelun tuottajasta, esimerkiksi ylläpitäjän tai asiakasyhteyshenkilön yhteystiedot. Myös lyhyt tekstikuvaus palvelusta sekä yksikäsitteinen nimi palvelulle määritellään profiilissa[4] .

2. *Toiminnallin kuvauksessa* esitellään palvelun käyttämät *syötteet*, sen tuottamat *paluuarvot*, *esiehdot*, jotka tulee olla voimassa ennen määrättyjä prosesseja sekä *tilamuutokset*, joita prosessien suorittaminen aiheuttaa [4]. Nämä samat käsitellään myös prosessimallissa, mutta tarkemmalla tasolla. OWL-S ei aseta rajoitteita sen suhteen, onko profiili ja prosessimalli konsistentit toisiinsa nähden, mutta ollakseen totuudenmukainen palvelun tarjoamien todellisten palvelujen suhteen, tulee profiilin ilmaista palvelut yhtenevästi prosessimallin suhteen.

3. *Toimintaa kuvaavat ominaisuudet* Ensinnäkin palvelu voidaan luokitella jonkun tunnetun luokittelun, esimerkiksi UNSPSC:n [footnote] mukaan. Toiseksi, palvelun laatuluokitus voidaan ilmaista profiilissa. Profiilin lopussa on määrittelemätön määrä parametreja, joilla voidaan kertoa esimerkiksi palvelun maantieteellisestä saatavuudesta, arvioidusta vasteajasta jne. [4].

Seuraavissa kappaleissa käsitellään em. kolmea osa-aluetta tarkemmin.

3.3.1 Tuottajainformaatio

Palveluntarjoajan yhteystiedot on tarkoitettu pääasiassa ihmisten luettavaksi. Ne ilmaistaan *Profile.owl*:ssä määritetyn `contactInformation` -ominaisuuden avulla. Yhteyshenkilöitä voidaan luonnollisestikin määritellä useita, esimerkissä ainoastaan yksi[6]:

```
<profile:contactInformation>
  <actor:Actor rdf:ID="BravoAir-reservation">
  <actor:name>BravoAir Reservation department</actor:name>
  <actor:title>Reservation Representative</actor:title>
  <actor:phone>412 268 8780</actor:phone>
      . . .
  </actor:Actor>
</profile:contactInformation>
```

Yteystietoihin kirjataan usein ylläpitäjän ja/tai kaupallisen edustajan tietoja.

Palvelun tekstikuvaus kirjoitetaan `textDescription`-ominaisuuden ja nimi `serviceName`-ominaisuuden avulla.

3.3.2 Toiminnallinen kuvaus

Toiminnallinen kuvaus ilmaisee mitä toimintoja palvelu tarjoaa ja minkä ehtojen puitteissa. OWL-S:n profiilissa ilmaistaan kahdenlaista funktionaalisuutta: syötteet ja tulosteet, jotka voidaan ajatella informaatiovirtoina sekä esiehdot ja vaikutukset, jotka voidaan ajatella tilamuutoksina. Edellisiä vastaavat owl-ominaisuudet ovat[4]:

hasInput, joka saa arvokseen `Process`-ontologiassa määriteltyjä `Input`-luokan ilmentymiä.

hasOutput, joka saa arvokseen `Process`-ontologiassa määriteltyjä `Output`-luokan ilmentymiä

hasPrecondition, joka määrittelee jonkin esiehdon, joka on luokan `Precondition` ilmentymä

hasresult, joka ilmaisee minkä ehtojen puitteissa tuloksia generoidaan sekä ja mitä tilamuutoksia prosessien suoritus saa aikaan. Saa arvokseen `Result`-luokan ilmentymiä.

Alla olevassa esimerkissä on määritetty, että prosessilla on syöte ”lähtökenttä” (*departureAirport*), tuloste ”lentoja löytynyt” (*FlightsFound*) ja tilamuutos ”istumapaikka löytynyt” (*HaveSeatResult*)[6]:

```
<profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.2/BravoAirProc
<profile:hasOutput rdf:resource="http://www.daml.org/services/owl-s/1.2/BravoAirPro
<profile:hasResult rdf:resource="http://www.daml.org/services/owl-s/1.2/BravoAirPro
```

Edellisessä esimerkissä on poimittu ainoastaan muutamia toiminnallisia määrityksiä, todelliset määritykset voi katsoa alkuperäisestä lähteestä[6].

3.3.3 Toimintaa kuvaavat ominaisuudet

Ontologiassa on myös mahdollista ilmoittaa palvelun luokitus jossain ontologiassa määriteltyä, mahdollisesti ulkopuolista luokitusta tai taksonomiaa käyttäen. Luokitus ilmaistaan `serviceCategory`-ominaisuuden avulla ja arvo on `ServiceCategory`-luokan ilmentymän. `ServiceCategory` luokalla on ominaisuuksia kategorian nimen, koodin jne ilmaisuun[4].

Profiilissa voidaan myös ilmaista palvelun tarjoajan tärkeäksi kokemia vapaavalintaisia attribuutteja `serviceParameter` -ominaisuudella. Parametrille annetaan aina nimi, joka on datatyyppiominaisuus sekä arvo, joka on jonkin olion instanssi[4]. Esimerkiksi palvelulle voidaan määritellä ominaisuus "BravoAir Geographic Radius" ja se saa arvoksen ontologiassa määritetyn alueen `jenkkilä`.

SERVICE CLASsIFICATION JA SERVICE PRODUCT?

3.4 Prosessi

OWL-S:n prosessimalli kertoo kuinka palvelu toimii ja kuinka sen informaatiovirrat kulkevat. On tärkeää kuitenkin muistaa, että prosessimalli ei ole suoritettava ojelma, vaan ainoastaan kuvaus sen tiominnasta[4]. Se kertoo, kuinka asiakasohjelmisto voi kommunikoida palvelun kanssa.

Prosessikuvaukset jaetaan kolmeen kategoriaan[4]:

1. *atominen prosessi* on sellaisen prosessin kuvaus, joka ottaa vastaan yhden viestin ja palautta yhden viestin.

2. *komposiittiprosessi* kuvaa prosessia, joka koostuu useasta eri atomisesta prosessista, ja joka ylläpitää tilatietoa.

3. *yksinkertainen prosessi* ei ole suoritettavissa eikä se kytkeydy maadoitukseen. Sen rooli on on toimia abstraktiona atomisille prosesseille tai komposiittiprosessien yksinkertaistuksena.

Prosesseilla voi olla määräämätön määrä syötteitä ja tulosteita. Samoin prosesseilla voi olla määräämätön määrä esiehtoja, joiden täytyy toteutua, jotta prosessi voidaan suorittaa. Vastaavasti voi olla määräämätön määrä vaikutuksia, joita prosessi aiheuttaa[4].

ESIMERKKI ATMICEISTA JA KOMPOSIITEISTÄ

Seuraavassa käydään läpi prosessin kuvausta jälleen BravoAir-esimerkin[6] avulla.

3.4.1 Osapuolet

Prosessilla on ainakin kaksi osapuolta (participant): asiakas (client) tai tai palvelin (server). Asiakas käyttää palvelimen tarjoamia palveluja[4]. Jos prosessilla on myös muita osapuolia, ne ilmaistaan `hasParticipant` suhteen avulla: ESIMERKKI? prosessi - hasParticipant - osapuoli

3.4.2 Syötteet ja paluuarvot

Syötteet kuvaavat prosessin sisään tuleva tietoa[4]. Atomisilla prosesseilla syöte on asiakkaan antama kun taas komposiittiprosesseilla osa syötteistä on asiakkaalta ja osa proessin edeltävän vaiheen tuottamaa.

Vaikka atomisille prosesseille sallitaan ainoastaan yksi syöte, se voi todellisuudessa koostua useasta syötteestä, koska syötteitä voi BUNDLATA yhdeksi [4]. Tällöin pitää ymmärtää käsite *viesti*, joka on usean syötteen BUNDLATTU kokonaisuus. BUNDLAUS määritellään palvelun maadoituksessa luvussa nnn.

Syötteet ja paluuarvot ilmaistaan `hasInput`- ja `hasOutput` - ominaisuuksilla (jotka siis kuvaavat *prosessia*). Ne voivat saada arvokseen `input`- ja `output` -luokan ilmentymiä. Edellämainituilla on kummallakin datatyyppiominaisuus parametrin tyypille ja arvolle[4].

BravoAirin prosessikuvauksessa on atominen prosessi `SelectAvailableFlight`, joka kuvaa prosessin, jossa asiakas valitsee haluamansa lennon saatavilla olevista vaihtoehdoista [6]. Sillä on on yksi syöte, lista tarjolla olevista lennoista (*SelectAvailableFlight_FlightsAvailable*), sekä yksi paluuarvo, valittu lento (*SelectAvailableFlight_SelectedFlight*). Syöte määritellään *hasInput*-tägien sisällä ja vastaavasti paluuarvo *hasOutput* -tägien sisässä[6]:

```
<process:AtomicProcess rdf:ID="SelectAvailableFlight">
    <rdfs:label>SelectAvailableFlight (ATOMIC)</rdfs:label>
        <rdfs:comment>
            Get users prefered flight choice from available itineraries
        </rdfs:comment>
        <process:hasInput>
            <process:Input rdf:ID="SelectAvailableFlight_FlightsAvailable">
                <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchem
                    http://www.daml.org/services/owl-s/1.2/Concepts.owl#FlightList
                </process:parameterType>
            </process:Input>
        </process:hasInput>
        <process:hasOutput>
            <process:Output rdf:ID="SelectAvailableFlight_SelectedFlight">
                <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchem
                    http://www.daml.org/services/owl-s/1.2/Concepts.owl#FlightItiner
                </process:parameterType>
            </process:Output>
        </process:hasOutput>
</process:AtomicProcess>
```

Esimerkissä määritellään atominen prosessi tunnuksella `SelectAvailableFlight`. Sillä on yksi syöte, joka on tyypiltään XMLSchemassa määritelty URI ja arvol-

taan lista lennoista (`Concepts.owl#FlightList`) sekä yksi paluuarvo, joka on samaa tyyppiä ja arvoltaan valittu lento (`Concepts.owl#FlightItineraryList`).

3.4.3 Esiehdot ja tulokset

Prosessia ei voi suorittaa, jos sille määrätty esiehto ei täyty[4]. Esiehto on prosessin ominaisuus ja se saa arvokseen ehtolausekkeen (*Condition*). ESIMERKKI?

Prosessin onnistuminen voi aiheuttaa muutoksen mailman tilassa tai sen, että palvelun kutsuja saa jotain informaatiota palvelulta. Tuloksen olemassaolo ilmaistaan ominaisuudella `hasResult`, joka saa arvokseen *Result* -olion. OWL-S:ssä ei sidota tulosta erikseen paluuarvoon tai muutokseen tilassa, vaan se määritellään *Result* -olion sisällä[4].

Seuraavassa aliluvussa käydään läpi koko tuloksen muodostus.

3.4.4 Ehdollisten palautusarvojen ja vaikutusten kuvaus

Kun tulos (`Result`) on esitelty, prosessimalli voi kuvailla sitä neljällä eri ominaisuudella[4]: `inCondition`, `hasResultVar`, `withOutput` ja `hasEffect`[4]. Jokainen edellinen on siis `result` -oliota kuvaava ominaisuus.

`inCondition` kertoo ehdon jonka vallitessa joku tulos on mahdollinen (eikä joku toinen tulos).

`withOutput` ja `hasEffect` puolestaan määrittelevät mitä seuraa siitä, että ehto on tosi. `hasResultVar` esittelee muuttujat, jotka sidotaan `inCondition`in määrittelemään ehtoon. Näitä muuttujia käytetään myös tulosten muodostamiseen[4].

Seuraavassa esimerkissä esitetään tuloksen muodostuminen [6]. Esimerkki liitty prosessiin, jossa varataan lento. `hasResult` kuvaa tuloksen muodostuksen, jossa onnistuneen ostotapahtuman jälkeen palautetaan paluuarvona lentosuunnitelma (`#BookFlight_PreferredFlightItinerary"`) sekä varaustunnus (`#BookFlight_ReservationID`).

```
<process:hasResult>
    <process:Result>
        <process:inCondition rdf:resource=
        "http://www.daml.org/services/owl-s/1.2/generic/Expression.owl#AlwaysTrue"/>
            <process:withOutput>
                <process:OutputBinding>
                    <process:toParam rdf:resource=
                        "#BookFlight_PreferredFlightItinerary"/>
                     <process:valueSource>
                          <process:ValueOf>
                              <process:theVar rdf:resource=
                                  "#CompleteReservation_PreferredFlightItinerary"/>
                              <process:fromProcess rdf:resource=
```

```
                                    "#PerformCompleteReservation"/>
                            </process:ValueOf>
                        </process:valueSource>
                    </process:OutputBinding>
                </process:withOutput>
                <process:withOutput>
                    <process:OutputBinding>
                        <process:toParam rdf:resource="#BookFlight_ReservationID"/>
                        <process:valueSource>
                            <process:ValueOf>
                                <process:theVar rdf:resource=
                                    "#CompleteReservation_ReservationID"/>
                                <process:fromProcess rdf:resource=
                                    "#PerformCompleteReservation"/>
                            </process:ValueOf>
                        </process:valueSource>
                    </process:OutputBinding>
                </process:withOutput>
    </process:Result>
</process:hasResult>
```

Esimerkissä ilmaistaan ominaisuudella `inCondition` ehto, jolla kyseinen tulos muodostetaan. Tässä tapauksessa ehto on aina tosi, mikä tarkoittaa, että kyseisen prosessin suorituksessa tämä tulos muodostetaan aina. `withOutput`-ominaisuuksilla kuvataan kuinka paluuarvot sidotaan muuttujiin. Tämä tuo mukaan OWL-S:n datavuoaspektin.

OWL-S:ssä datavuo toimii periaatteella ”kuluttaja pyytää”, eli mikään prosessi (tuottaja) ei työnnä dataa toiselle prosessille (kuluttaja) vaan kuluttajaprosessi pyytää sitä kuten esimerkkitapauksessamme `OutputBinding` -oliossa. Esimerkissä ylemmän `OutputBinding`in `toParam` -ominaisuuden arvo kertoo, että sidomme palautettavan arvon paluuarvomuuttujaan *#BookFlight_PreferredFlightItinerary*. Muuttujan saama arvo määrätään tulevaksi prosessin `#PerformCompleteReservation` muuttujasta `#CompleteReservation_PreferredFlightItinerary`.

Yksinkertaistettuna datavuo prosessilta prosessille kulkee seuraavalla tavalla:

Prosessilla *p2* on paluuarvomuuttujan määritys (`toParam`) nimeltään ”o2”. Määritellään, että kyseisen paluuarvomuuttujan arvo saadaan prosessin (`fromProcess`) *p1* muuttujasta (`theVar`) ”*o1*”.

KUVA!!!

3.4.5 Komposiittiprosessit ja kontrollirakenteet

Komposiittiprosessi voidaan hajottaa atomisiksi prosesseiksi tai komposiittiprosesseiksi[4]. Tämä hajoittaminen voidaan ilmaista kontrollirakenteilla kuten `Sequence`

tai `If-Then-Else`. Vaikka kontrollirakenteen nimet muistuttavat ohjelmointikielistä tuttuja nimityksiä, on syytä muistaa, että ne *eivät kerro* kuinka niiden kuvaama palvelu toimii vaan miten se *mahdollisesti* toimii[4].

OWL-S määrittelee luokan `compositeProcess`, jolla on tasan yksi `composedOf` -ominaisuus, joka saa arvokseen `ControlConstruct` - luokan ilmentymän. Jokaisella `ControlConstruct`:lla on ominaisuus `components`, joka saa arvokseen toisia kontrollirakenteita[4]. Kontrollirakenteet muodostavat siis puumaisen rakenteen, jonka solmut, jotka eivät ole lehtiä, ovat kontrollirakenteita. Puun lehdet ovat prosessien suoritusta kuvaavia luokkia nimeltään `Perform`. Jokaisella `Perform` -luokalla on ominaisuus `process`, joka viittaa suoritettavaan prosessiin[4]. Prosessin saamat syötteet määräävät, koska se suoritetaan osana suurempaa kontrollirakennetta. Datavuohon palataan tarkemmin seuraavassa kappaleessa. PALATAANKO?

Komposiittirakennetta voidaan voidaan ajatella ”black boxina”, jolloin se voidaan nähdä abstraktina yksinkertaisena prosessina (*simple process*). Vastaavasti yksinkertainen prosessi voidaan nähdä komposiittirakenteena, ”white boxina” [4].

Seuraavassa luetellaan OWL-S:n kontrollirakenteita lyhyiden selostusten kera:

- **Sequence** on lista kontrollirakenteita, jotka suoritetaan sarjana.
- **Split** mahdollistaa prosessien suorituksen rinnakkaisesti. Rakenteet poimitaan kontrollirakennesäiliöstä, `ControlConstructBag`istä.
- **Split+Join** toimii samoin kuin Sequence, mutta prosessien suorituksen jälkeen suoritetaan puomisynkronointi. Myös tässä prosessit poimitaan säiliöstä.
- **Any-Order** -rakenne mahdollistaa prosessien suorituksen määräämättömässä järjestyksessä mutta ei samanaikaiseti.
- **Choice** toimii siten, että poimitaan joku yksi säiliön prosessi suoritettavaksi.
- **If-ThenElse** -luokassa on ominaisuus `ifCondition`, jolle annetaan ehto `Condition`. Jos ehto on tosi, suoritetaan `Else` -ominaisuuden arvona oleva kontrollirakenne tai prosessi, jos epätosi, suoritetaan `Else` -ominaisuuden arvona oleva rakenne tai prosessi.
- **Repeat-While ja Repeat-Until** luokat ovate luokan `Iterate` aliluokkia. Ne toimivat samaan tapaan kuin perinteisten proseduraalisten kielten while- ja do-while -rakenteet.

3.5 Maadoitus

Palvelun maadoitus (grounding) kertoo kuinka palvelu on saavutettavissa käytännön tasolla. Tärkeimmät maadoitusdokumentin ilmaisemat asiat ovat viestien muoto, viestiprotokollat, sarjallistaminen, osoitteet jne[4]. Voidaan ajatella, että palvelun käytön abstrakti kuvaus MÄPÄTÄÄN konkreettiselle, käytännön tasolle[4].

OWL-S:n maadoituksen tärkein tehtävä on kertoa kuinka atomisten prosessien syötteet ja paluuarvot muutetaan viesteiksi, jotka voidaan kuljettaa jollain menetelmällä, esim. html-protokollan avulla [4].

OWL-S nojaa viestien määrittelyssä vahvasti jo olemassa olevaan WSDL -kieleen, jonka puitteissa on tehty paljon kehitystyötä konkreettisen viestinmäärittelyn aikaansaamiseksi. WSDL tulee sanoista Web Services Description Language ja on XML-formaatti web-palveluiden kuvaamiseksi ENDPOINTEINA, jotka käsittelevät viestejä. WSDL:ssä ENDPOINTIT ja viestit määritellään abstraktilla tasolla, mutta MÄPÄTÄÄN konkreettisiksi viestiprotokolliksi ja viestiformaateiksi [2]

WSDL:n sopivuus perustuu siihen, että OWL-S:n maadoitus on melko yhdenmukainen WSDL:n *sitomisen* kanssa[4]. Kuitenkin molempia kieliä pitää käyttää toisiaan täydentävästi, koska kumpikaan ei yksinään tarjoa kaikkia ominaisuuksia.

Koska maadoituksen tarkoituksena on MÄPÄTÄ abstrakti kuvaus konkreettisiksi viestinkuljetusoperatioiksi, tulee muodostaa ikään kuin silta OWLS-n ja WSDL:n välille. Siltana toimii OWL-S:n maadoitusdokumentti. Siinä kuvataan pääasiassa `WSDLAtomicProcessGrounding` -ilmentymiä, koska WSDL on useimmiten viestinvaihdon konkreettinen toteutus. Niissä märitellään viitteet OWL-S:n prosesseista, syötteistä ja paluuarvoista WSDL:n vastaaviin, konkreettisiin määrityksin.

Seuraavat `WSDLAtomicProcessGrounding`in ominaisuudet mahdollistavat OWL-S:n konstruktioiden mäppäyksen WSDL:n vastaaviin (vain tärkeimät esitelty):

- `wsdlDocument` kertoo käytössä olevan WSDL-dokumentin URIn. Kaikki viittaukset wsdl-määrityksiin viittaavat tähän dokumenttiin.
- `wsdlOperation` viittaa wsdl-dokumentissa määriteltyyn operaatioon, joka vastaa OWL-S:n atomista prosessia. BravoAir-esimerkissä MÄPÄTÄÄN prosessikuvauksessa oleva *BravoAirProcess.owl#LogIn* -prosessi WSDL-määrityksen vastaavaan operaatioon `#LogIn_operation`:

  ```
  <grounding:owlsProcess rdf:resource="http://www.daml.org/services/owl-s/1.2/Bra
  <grounding:wsdlOperation rdf:resource="#LogIn_operation"/>
  ```

  WSDL:n operaatiomäärityksiä ei voi sisällyttää tähän raporttiin joten niiden rakennetta voi tutkia lähteistä [6] ja [2].

- `wsdlInputMessage` on olio, johon on talletettu sen WSDL-määrityksen URI, joka ilmaisee syötteitä kuljettavien viestien muodon.
- `wsdlInput` -oliossa MÄPÄTÄÄN jokainen mäpättävän prosessin syöte WSDL:ssä määritetyn viestin osaksi. Ideaalitapauksessa jokaista syötettä vastaa yksi WSDL:n viestin osa. Alla olevassa esimerkissä. MÄPÄTÄÄN BravoAirin prosessikuvauksen syöte `#LogIn_AcctName` WSDL-määritykseen `#acctName`. Viittaus jälleen URI:n avulla.

```
<grounding:wsdlInput>
    <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource="http://www.daml.org/services/owl
        <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSche
            http://www.daml.org/services/owl-s/1.2/BravoAirGrounding.wsdl#acctN
        </grounding:wsdlMessagePart>
    </grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
```

- `wsdlOutput` ja `wsdlOutputMessage` toimivat vastaavasti kuin edelliset, mutta MÄPPÄÄVÄT prosessin paluuarvoja.

4 Pohdintaa

versioasiat

menestynyt tutkimusmaailmassa. [5]

[5]OWL-S could still be improved in a number of ways. First, there is currently no way of describing desirable security and QoS properties for Web Services; there is a dire need for both in the community, and promising work is underway. Second, the process model lacks exception handling, which would allow for a closer alignment with WSDL faults. Third, we plan to update the grounding subontology so that it works with WSDL 2.0. Fourth, further work is needed on algorithms that effectively support discovery, selection, and composition of services in large-scale, complex settings. Fifth, to encourage the community to develop OWL-S extensions, we are contemplating the prospect of adopting an open-source style of code evolution. Finally, as in other areas of the Semantic Web, we need more domain-level ontologies; there is a need to develop specializations of OWL-S profiles for a variety of domains, and for broad categories of services; in addition, for services that sell goods, ontology modules are needed for specifying cost models, negotiations, contracts, and guarantees.

Lähteet

[1] Berners-Lee, Tim, Hendler, James ja Lassila, Ora: *The Semantic Web*. Scientific American, 284(5):34–43, 2001. http://www.nature.com/doifinder/10.1038/scientificamerican0501-34.

[2] Christensen, Erik, Curbera, Francisco, Meredith, Greg ja Weerawarana, Sanjiva: *Web Service Definition Language (WSDL)*, 2001. http://www.w3.org/TR/wsdl.

[3] Manola, Frank ja Miller, Eric: *RDF Primer*. W3C Recommendation, 10(February):1–107, 2004. http://www.w3.org/TR/rdf-primer/.

[4] Martin, David, Burstein, Mark, Hobbs, Jerry, Lassila, Ora, McDermott, Drew, McIlraith, Sheila, Narayanan, Srini, Paolucci, Massimo, Parsia, Bijan, Payne, Terry R ja al. et: *OWL-S: Semantic Markup for Web Services*. W3C Member Submission, 22(2008-01-07):2007–04, 2004. http://eprints.soton.ac.uk/262687/.

[5] Martin, David, Burstein, Mark, McDermott, Drew, McIlraith, Sheila, Paolucci, Massimo, Sycara, Katia, McGuinness, Deborah L, Sirin, Evren ja Srinivasan, Naveen: *Bringing Semantics to Web Services with OWL-S*. World Wide Web Internet And Web Information Systems, 10(3):243–277, 2007. http://www.springerlink.com/index/10.1007/s11280-007-0033-x.

[6] OWL-S-Coalition: *OWL-S 1.2 Release: Examples*, 2012. http://www.ai.sri.com/daml/services/owl-s/1.2/examples.html.

[7] Smith, Michael K, Welty, Chris ja McGuinness, Deborah L: *OWL Web Ontology Language Guide*. W3C Recommendation, 10(February):1–46, 2004. http://www.w3.org/TR/owl-guide/.