

Esimerkkiotsikko

Eija Esimerkki

Seminaariraportti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 22. syyskuuta 2013

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	22. syyskuuta 2013	15	
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	3
2	Web-ontologiakieli OWL	4
2.1	URI, XML ja RDF	4
2.2	tarpeelliset OWL-konstruktio OWL-S:n ymmärtämiseen . . .	5
3	Webpalveluiden kuvauskieli OWL-S	5
3.1	Korkean abstraktiotason rakenne	5
3.2	Profiili	6
3.2.1	Ontologioiden tuonti ja viitteet prosessimalliin sekä palveluun	7
3.2.2	Tuottajainformaatio	7
3.2.3	Toiminnallinen kuvaus	7
3.2.4	Toimintaa kuvaavat ominaisuudet	8
3.3	Prosessi	8
3.3.1	Osapuolet	9
3.3.2	Syötteet ja tulosteet	9
3.3.3	Esiehdot ja tulokset	10
3.3.4	Ehdollisten palautusarvojen ja vaikutusten kuvaus . .	10
3.3.5	Komposiittiprosessit ja kontrollirakenteet	12
3.4	Maadoitus	13
	Lähteet	14

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[4].

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 esimerkiksi 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[4]. Jotta voidaan välttää ihmisen osallistumien web-palvelujen orkestrointiin, tulee olla *automaattisia ohjelmistoagentteja*, jotka etsivät oikeat palvelut ja käyttävät niitä automaattisesti, ilman ihmisen puuttumista asiaan[4].

Automaattinen ohjelmistoagentti mahdollistaisi esimerkiksi:

- *Presentaation pitäjä haluaa lähettää esitysmateriaalinsa kaikkien osanottajien käyttöön. Hän voisi tehdä sen yhdellä lähetyksenä painalluksella riippumatta siitä, mitä teknologioita ja protokollia vastaanottajat käyttävät*[4].
- *Kuluttaja haluaa löytää haluamansa tuotteen mahdollisimman halvalla jostain verkkokaupasta. Ohjelmistoagenttia hyödyntävä ohjelma hakee tuotteen hintaa ja saatavuutta useista verkkokaupoista, jotka ovat koodanneet kataloginsa jostain standardoitua sanastoa käyttäen. Sanastojen ei edes tarvitse olla samat eri verkkokaupoilla*[4].

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 soveltuvien osien OWL (Web Ontology Language) :ään ja sen konstruktioihin, joiden avulla web-palveluontologioita voidaan muodostaa.

2 Web-ontologiakieli OWL

World Wide Web Consortium on antanut OWL:lle suosituksen standardiksi vuonna 2004[?]. Uudempi suositus on OWL 2:lle vuodelta 2012. OWL kielenä perustuu muutamiin jo aiemmin määritettyihin 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 suhteita 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 kuvaila webissä olevaa tietoa kolmikoiden avulla[?]. RDF-kolmikon muodostavat subjekti, objekti ja predikaatti[?]. 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*. Tämänkaltaisen XML-kielinen notaatio on OWL:n (ja OWL-S:n) ontologioiden kuvaamisen

syntaktinen ydin. Pelkkä RDF -syntaksi ei kuitenkaan riitä, koska semanttikoiden kuvaamiseen tarvitaan myös monimutkaisempia abstraktioita kuten luokka, aliluokka, ilmentymä, suhde ja alisuhde.

2.2 tarpeelliset OWL- ja RDFS -konstruktiot OWL-S:n ymmärtämiseen

OWL on kehittynyt versio RDF Schemasta.

luokka
instanssi
suhde
aliluokka
alisuhde
import
union
disjointWith
risuaitaviittaukset

3 Webpalveluiden kuvauskieli OWL-S

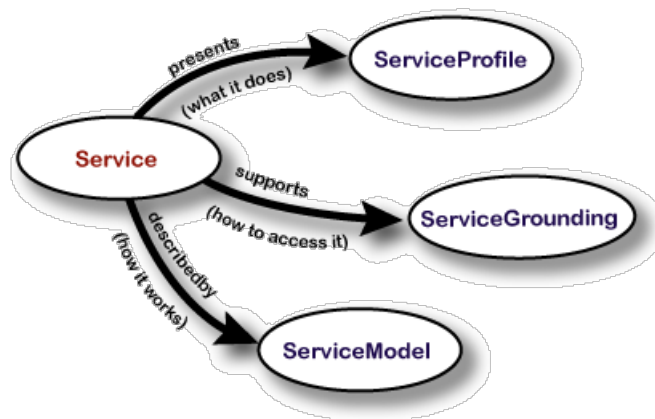
UDDI löytäminen, käyttö, monitorointi

3.1 Korkean abstraktiotason rakenne

Palveluontologian korkean abstraktiotason rakenne muodostuu kolmen tyyppisestä tiedosta ja ne vastaavat kolmeen eri kysymykseen [3]:

- *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** [3].
- *Kuinka palvelua käytetään?* Ontologian *prosessimalli* antaa vastauksen tähän ja se esitetään luokassa **ServiceModel**. Palvelun ja sen prosessimallin välillä on **describedBy** -suhde[3].
- *Miten palvelun kanssa kommunikoidaan?* Tähän antaa vastauksen ontologian *maadoitus*, jossa määritellään esimerkiksi tuki erilaisille viestiprotokollille. **Service** -luokalla on ominaisuus **supports**, joka viittaa **ServiceGrounding** -luokkaan[3].

Kuten kuvasta 1 voidaan nähdä, jokaista julkistettua palvelua kohden on yksi **Service**-luokan instanssi, joka edustaa **ServiceProfilea** suhteella **presents**, on **ServiceModelin** kuvailema suhteella **describedBy** ja **serviceGroundingin** tukema suhteella **supports**.



Kuva 1: OWL-S:llä kuvatun palveluontologian korkean taon rakenne [3]

3.2 Profili

Profiili siis kertoo mitä palvelu tekee ja kuka palvelun tarjoaa. Se mahdollistaa asiakkaita (agentteja) löytämään palvelun esimerkiksi keskitetyistä palvelurekistereistä kuten UDDI tai puhtaasti P2P:n puitteissa. Profiili tarjoaa kolmenlaista informaatiota [3]:

1. *Tuottajainformaatio* kertoo tietoja palvelun tuottajasta, esimerkiksi ylläpitäjän tai asiakasyhteyshenkilön yhteystiedot. Myös lyhyt tekstikuvaus palvelusta sekä yksikäsitteinen nimi palvelulle määritellään profiilissa [3].
2. *Toiminnallinen kuvaus* kuvaa (tautologia) palvelun käyttämät *syötteet*, sen tuottamat *tulosteet*, *esiehdot*, jotka tulee olla voimassa ennen määrättyjä prosesseja sekä *tilamuutoksia*, joita prosessien suorittaminen aiheuttaa [3]. ESIMERKKEJÄ?? 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. [3].

Seuraavassa käsitellään em. kolmea osa-aluetta sekä profiilitiedoston rakenne tarkemmin.

3.2.1 Ontologioiden tuonti ja viitteet prosessimalliin sekä palveluun

Profiilin(-tiedoston) alussa voidaan tuoda ontologian käyttöön muita jo määriteltyjä ontologioita tavallisilla owl:n `imports`-lauseilla. Esimerkissä tuodaan palvelun pääasiallinen määritelmä `BravoAirService` ontologian käyttöön[5]:

```
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.2/BravoAirService.owl"/>
```

Jokaista profilia edustaa palvelu. Viittaus palvelun määritelmään ilmaistaan `presentedBy`-suhteella[5]:

```
<service:presentedBy rdf:resource="http://www.daml.org/services/owl-s/1.2/BravoAirService.owl#BravoAirService"/>
```

3.2.2 Tuottajainformaatio

Palveluntarjoajan yhteystiedot on tarkoitettu pääasiassa ihmisten luettavaksi. Yhteyshenkilöitä voidaan luonnollisestikin määritellä useita, esimerkiksi ainoastaan yksi[5]:

```
<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`-tägin ja nimi `serviceName`-tägin sisään.

3.2.3 Toiminnallinen kuvaus

Toiminnallinen kuvaus ilmaisee mitä toimintoja palvelu tarjoaa ja minkä ehtojen puitteissa. OWL-S `Profile` ilmaisee kahdenlaista funktionaalisuutta: syötteet ja tulosteet, jotka voidaan ajatella informaatiovirtoina sekä esiehdot ja vaikutukset, jotka voidaan ajatella tilamuutoksina. Edellisiä vastaavat owl-ominaisuudet ovat[3]:

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 ”läh-tökenttä” (*departureAirport*), tuloste ”lentoja löytynyt” (*FlightsFound*) ja tilamuutos ”istumapaikka löytynyt” (*HaveSeatResult*)[5]:

```
<profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.2/BravoAirPro
<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 määrittelyksiä, todelliset määrittelyt voi katsoa liitteestä nnn.

3.2.4 Toimintaa kuvaavat ominaisuudet

Edellisessä aliluvussa luettelimme palvelun toiminnallisia ominaisuuksia.

Ontologiassa on myös mahdollista ilmoittaa palvelun luokitus jossain ontologiassa määriteltä, mahdollisesti ulkopuolista luokitusta tai taksonomiaa käyttäen. Luokitus ilmaistaan **serviceCategory**-tagien sisällä ja arvo on **ServiceCategory**-luokan ilmentymän. **ServiceCategory** luokalla on ominaisuuksia kategorian nimen, koodin jne ilmaisuun[3]. ESIMERKKI?

Profilissa voidaan myös ilmaista palvelun tarjoajan tärkeäksi kokemia vapaaehtoisia attribuutteja **serviceParameter**-ominaisuudella. Parametrille annetaan aina nimi, joka on datatyypin ominaisuus sekä arvo, joka on jonkin olion instanssi[3]. Esimerkiksi palvelulle voidaan määritellä ominaisuus ”BravoAir Geographic Radius” ja se saa arvoksen ontologiassa määritetyn alueen jenkkiä.

SERVICE CLASSIFICATION JA SERVICE PRODUCT?

3.3 Prosessi

Jotta voidaan tarjota yksityiskohtaista tietoa siitä, miten olla vuorovaikutuksessa palvelun kanssa, esitetään se OWL-S:ssä myös *prosessina*. On tärkeää tietää, että prosessimalli ei ole suoritettava ohjelma, vaan ainoastaan kuvaus siitä[3]. Se kertoo, kuinka asiakasohjelmisto voi kommunikoida palvelun kanssa.

Prosessikuvaukset jaetaan kolmeen kategoriaan[3]:

1. *atominen prosessi* on sellaisen prosessin kuvaus, joka ottaa vastaan yhden viestin ja palauttaa 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[3].

ESIMERKKI ATMICEISTA JA KOMPOSIITEISTA

Seuraavassa käydään läpi prosessin kuvausta jälleen BravoAir-esimerkin[5] avulla.

3.3.1 Osapuolet

Prosessilla on ainakin kaksi osapuolta (participant): asiakas (client) tai palvelin (server). Asiakas käyttää palvelimen tarjoamia palveluja[3]. Jos prosessilla on myös muita osapuolia, ne ilmaistaan **hasParticipant** suhteen avulla: ESIMERKKI? prosessi - hasParticipant - osapuoli

3.3.2 Syötteet ja tulosteet

Syötteet kuvaavat prosessin sisään tuleva tietoa[3]. Atomisilla prosesseilla syöte on asiakkaan antama kun taas komposiittiprosesseilla osa syötteistä on asiakkaalta ja osa prosessin edeltävän vaiheen tuottamaa.

Vaikka atomisille prosesseille sallitaan ainoastaan yksi syöte, se voi todellisuudessa koostua useasta syötteestä, koska syötteitä voi BUNDLATA yhdeksi [3]. Tällöin pitää ymmärtää käsite *viesti*, joka on usean syötteen BUNDLATU kokonaisuus. BUNDLAUS määrittellään palvelun maadoituksessa luvussa nnn.

BravoAirin prosessikuvauksessa on atominen prosessi **SelectAvailableFlight**, joka kuvaa prosessin, jossa asiakas valitsee haluamansa lennon saatavilla olevista vaihtoehdoista [5]. Sillä on on yksi syöte, lista tarjolla olevista lennoista (*SelectAvailableFlight_FlightsAvailable*), sekä yksi paluuarvo, valittu lento (*SelectAvailableFlight_SelectedFlight*). Syöte määrittellään *hasInput*-tägien sisällä ja vastaavasti paluuarvo *hasOutput* -tägien sisässä[5]:

```
<process:AtomicProcess rdf:ID="SelectAvailableFlight">
  <rdfs:label>SelectAvailableFlight (ATOMIC)</rdfs:label>
  <rdfs:comment>
    Get users preferred flight choice from available itineraries
  </rdfs:comment>
  <process:hasInput>
    <process:Input rdf:ID="SelectAvailableFlight_FlightsAvailable">
      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

```

        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/XMLSchema#string"
            http://www.daml.org/services/owl-s/1.2/Concepts.owl#FlightItineraryList>
        </process:parameterType>
    </process:Output>
</process:hasOutput>
</process:AtomicProcess>

```

Esimerkissä määritellään atominen prosessi tunnukseksi `SelectAvailableFlight`. Sillä on yksi syöte, joka on tyyppiltään XMLSchemassa määritelty URI ja arvoltaan lista lennoista (`Concepts.owl#FlightList`) sekä yksi paluuarvo, joka on samaa tyyppiä ja arvoltaan valittu lento (`Concepts.owl#FlightItineraryList`).

3.3.3 Esiehdot ja tulokset

Prosessia ei voi suorittaa, jos sille määrätty esiehto ei täyty[3]. Esiehto on prosessin ominaisuus ja se saa arvokseen lausekkeen (*expression*). ESIMERKKI?

Prosessin onnistuminen voi aiheuttaa muutoksen mailman tilassa tai sen, että palvelun kutsuja saa jotain informaatiota palvelulta. Tulos ilmaistaan termillä *Result* ja se liittyy prosessiin ominaisuudella *hasResult*. OWL-S:ssä ei sidota tulosta erikseen paluuarvoon tai muutokseen tilassa, vaan se määritellään *Resultin* sisällä[3].

Seuraavassa aliluvussa käydään läpi koko tuloksen muodostus.

3.3.4 Ehdollisten palautusarvojen ja vaikutusten kuvaus

Kun tulos (*Result*) on määritelty, prosessimalli voi kuvailla sitä neljällä eri ominaisuudella[3]: *inCondition*, *hasResultVar*, *withOutput* ja *hasEffect*[3].

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 *inConditionin* määrittelemään ehtoon. Näitä muuttujia käytetään myös tulosten muodostamiseen[3].

Seuraavassa esimerkissä esitetään tuloksen muodostuminen [5]. Esimerkki liittyy 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`-oliassa. Esimerkissä ylemmän `OutputBindingin` `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 määrittely (`toParam`) nimeltään "o2". Määritellään, että kyseisen paluuarvomuuttujan arvo saadaan prosessin (`fromProcess`) *p1* muuttujasta (`theVar`) "o1".

KUVA!!!

3.3.5 Komposiittiprosessit ja kontrollirakenteet

Komposiittiprosessi voidaan hajottaa atomisiksi prosesseiksi tai komposiittiprosesseiksi[3]. Tämä hajottaminen 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 kuvaam palvelu toimii vaan miten se *mahdollisesti* toimii[3].

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[3]. 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[3]. Prosessin saamat syötteet määräävät, koska se suoritetaan osana suurempaa kontrollirakennetta. Datavuohon palataan tarkemmin seuraavassa kappaleessa.

Komposiittirakennetta voidaan voidaan ajatella "black boxina", jolloin se voidaan nähdä abstraktina yksinkertaisena prosessina (*simple process*). Vastaavasti yksinkertainen prosessi voidaan nähdä komposiittirakenteena, "white boxina" [3].

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 samanaikaisesti.

- **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.

3.4 Maadoitus

Palvelun maadoitus kertoo (grounding) kuinka palvelu on saavutettavissa käytännön tasolla. Tärkeimmät maadoitusdokumentin ilmaisemat asiat ovat viestien muoto, viestiprotokollat, sarjallistaminen, osoitteet jne[3]. Voidaan ajatella, että palvelun käytön abstrakti kuvaus MÄPÄTÄÄN konkreettiselle, käytännön tasolle[3].

OWL-S:n maadoituksen tärkein tehtävä on kertoa kuinka atomisten prosessien syötteen ja paluuarvot muutetaan viesteiksi, jotka voidaan kuljettaa jollain menetelmällä, esim. html-protokollan avulla [3].

OWL-S nojaa viestien määrittelyssä vahvasti jo olemassa olevaan WSDL -kieleen, jonka puitteissa on tehty paljon kehitystyötä konkreettisen viestin määrittelyn 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[3]. 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 OWL-S:n ja WSDL:n välille. Silta toimii OWL-S:n maadoitusdokumentti. `Grounding` -luokan aliluokan `WSDLGrounding` ilmentymässä kuvataan `WSDLAtomicProcessGrounding` -ilmentymiä. Näissä em. ilmentymissä määritellään viitteet OWL-S:n prosesseista, syötteistä ja paluuarvoista WSDL:n vastaaviin, konkreettisiin määrittelyihin.

Seuraavat `WSDLAtomicProcessGrounding`in ominaisuudet MÄPPÄÄVÄT OWL-S:n konstruktioita WSDL:n vastaaviin (kaikkia ei esitelty):

- `wsdlDocument` kertoo käytössä olevan WSDL-dokumentin URIn. Kaikki seuraavat viittaukset wsdl-määrittelyihin viittaavat tähän dokument-

tiin.

- `wsdlOperation` viittaa wsdl-dokumentissa määriteltyyn operaatioon, joka vastaa OWL-S:n atomista prosessia. BravoAir-esimerkissä MÄPÄTÄÄN prosessikuvauksessa oleva *BravoAirProcess.owl#LogIn* -prosessi WSDL-määrittelyn vastaavaan operaatioon `#LogIn_operation`.

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

WSDL:n operaatiomäärittelyä ei voi sisällyttää tähän raporttiin joten niiden rakennetta voi tutkia lähteistä [5] ja [2].

- `wsdlInputMessage` on olio, johon on talletettu sen WSDL-määrittelyn URI, joka ilmaisee syötteitä kuljettavien viestien muodon.
- `wsdlInput` -oliassa MÄPÄTÄÄN jokainen määrittävän 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-määrittelyyn `#acctName`. Viittaus jälleen URI:n avulla.

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

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

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] 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/>.
- [4] 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>.
- [5] OWL-S-Coalition: *OWL-S 1.2 Release: Examples*, 2012. <http://www.ai.sri.com/daml/services/owl-s/1.2/examples.html>.