Matias Hirvonen

Avoimeen lähdekoodiin perustuvat Javan web-palveluiden ohjelmistokehykset

Tietojärjestelmätieteen
kandidaatintutkielma
5.6.2009

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Hirvonen, Matias Juhani
Tietojärjestelmätieteen kandidaatintutkielma / Matias Hirvonen
Jyväskylä: Jyväskylän yliopisto, 2009.
39 s.
Kandidaatintutkielma

Palvelukeskeisten arkkitehtuurien nauttiessa kasvavaa huomiota tietojärjestelmien suunnittelussa myös web-palveluiden tehokas toteuttaminen on noussut avainasemaan. Tutkielmassa kerrotaan lyhyesti mitä web-palvelut ovat ja luodaan katsaus niiden historiaan. Web-palvelun käsite yhdistetään tekniseen toteutukseen jaottelemalla palvelut karkeasti arkkitehtuurin ja toteutustavan perusteella. Aihealueen laajasta termistöstä selvitetään ohjelmistosuunnittelijan kannalta tärkeimmät termit, niiden merkitys ja keskinäiset suhteet.

Web-palveluiden roolia Java-ympäristössä käsitellään Javan ohjelmistoalustojen kautta ja paneudutaan tarkemmin Java Enterprise Editionin web-palvelurajapintaan. Lopuksi vertaillaan seitsemää keskenään varsin erilaista Java-kielellä toteutettua ohjelmistokehystä. Niiden hyviä ja huonoja puolia vertaillessa todetaan parhaan vaihtoehdon valinnan olevan tapauskohtaista. Kehysten ominaisuudet kootaan valintaa helpottavaan taulukkoon.

AVAINSANAT: web-palvelu, ohjelmistokehys, Java, SOA, REST, RPC

Ohjaaja: Pertti Hirvonen
Tietojenkäsittelytieteiden laitos
Jyväskylän Yliopisto

Tarkastaja: Jorma Kyppö
Tietojenkäsittelytieteiden laitos
Jyväskylän Yliopisto

SISÄLLYS

JOHDANTO

Ohjelmistoarkkitehdeille ja yritysten johtajille suunnatussa kirjassaan Agile-Path-yrityksen toimitusjohtaja Eric Marks esittää, että laajoja tietojärjestelmäarkkitehtuureja toteutettaessa palvelukeskeisten arkkitehtuurien (Service-oriented Architecture, SOA) asema on muuttunut vaihtoehdosta enemmänkin välttämättömyydeksi. Syyksi hän näkee erityisesti Internetin kasvaneen roolin yritysten sisäisen ja ulkoisen tiedonvälityksen kanavana. Marksin mukaan kaupalliset tarpeet ovat luoneet pohjaa verkon yli käytettävien web-palveluiden jatkuvaan kehittämiseen. Palvelulla tarkoitetaan tietoverkon välityksellä, koneen ymmärtämässä muodossa tarjottavaa tietolähdettä tai logiikkaa sisältävää komponenttia. (Marks, 2003, luku 1)

Palvelukeskeinen arkkitehtuuri ja sitä toteuttavat web-palvelut ovat varsin laaja ja moniulotteinen kokonaisuus. Tässä tutkielmassa keskitytään vain Java-ohjelmointikielellä, Java-ajoympäristöön toteutettuihin, avoimen lähdekoodin ohjelmistokehyksiin. Aluksi kerrotaan lyhyesti mitä web-palveluilla tarkoitetaan ja esitellään niihin liittyvää peruskäsitteistöä. Tutkielman pääaiheena selvitetään mitä maksuttomia vaihtoehtoja on saatavilla web-palvelun toteuttamiseen Java-ajoympäristössä ja arvioidaan niiden teknistä käyttöönottoa sovelluskehittäjän kannalta. Vertailussa arvioidaan käyttöönoton helppoutta ja pyritään luomaan arvio kehyksen tulevaisuuden näkymiin. Tekniseltä puolelta selvitetään mitä protokollia kehys käyttää ja mahdollisesti spesifikaatiot, jotka siinä toteutetaan.

Ilmestymisestään, 90-luvun loppupuolelta asti Java-ohjelmointikielen ja ajoympäristön suosio on kasvanut. Ohjelmistokehittäjien tarkkaa lukumäärää on vaikea arvioida, mutta monesta muuttujasta laskettavassa TIOBE Softwaren ylläpitämässä indeksissä Java nostetaan tällä hetkellä suosituimmaksi kieleksi. Ristiriitaisia tutkimustuloksia on mittaus- ja arviointitavasta riippuen lukuisia, mutta kiistatta voidaan sanoa, että Java on varsin suosittu teknologia suurten

verkkopohjaisten järjestelmien toteutuksessa. (TIOBE, 2009) (Sun Microsystems) (Java-Source.net [1])

Web-palveluita ja niiden soveltuvuutta palvelukeskeisen arkkitehtuurin toteuttamiseen on tutkittu etenkin viime vuosina enenevissä määrin. Tutkielman tekniseen näkökulmaan soveltuvaa aineistoa on saatavilla paljon etenkin kirjojen muodossa. Aihealueen nuoruuden vuoksi yleisesti hyväksyttyjä perusteoksia ei kuitenkaan ole tunnistettavissa. Tutkielmassa huomioidaan muodostuneet koulukuntaerot erityisesti web-palveluiden määrittelyssä ja valitaan mahdollisimman useita toteutusvaihtoehtoja kattava määritelmä.

Tutkielman kirjoittaja työskentelee ohjelmistosuunnittelijana Java-teknologioilla toteutettavissa projekteissa. Tämän tutkimuksen tuloksia käytetään hyödyksi valittaessa sopivaa ohjelmistokehystä web-palveluiden toteuttamiseen tulevissa projekteissa. Kandidaatin tutkielman jälkeen tehdään samasta aihealueesta pro-gradu tutkielma, jonka aihetta tämän tutkielman tulokset tarkentavat. Jatkuvasti kehittyvät teknologiat sekä tutkielman pituuteen nähden laaja aihealue eivät luo odotuksia tässä tutkielmassa esitettyjen arvioiden pidemmälle kantavasta akateemisesta hyödystä.

1 WEB-PALVELUISTA YLEISESTI JA NIIHIN LIITTYVÄÄ PERUSKÄSITTEISTÖÄ

1.1 Historiaa

Hajautettujen tietojärjestelmien historian voidaan katsoa ulottuvan ainakin 70-luvulle saakka, jolloin IETF (Internet Engineering Task Force) julkaisi ehdotelman standardista tiedon liikuttamiseksi maantieteellisesti hajautuneiden koneiden, sovellusten ja ihmisten välillä (IETF, 1976). Vaikka kolme vuosikymmentä sitten suunniteltujen hajautettujen järjestelmien toteutukset olivat yksinkertaisia ohjelmistojen etäkutsuja (Remote Procedure Call, RPC) yhdestä UNIX-käyttöjärjestelmästä toiseen, teknisestä näkökulmasta samoihin haasteisiin pyritään vastaamaan myös tämän päivän uusimmissa web-palvelujen toteutuksissa (Rosen, 2008, sivut 3-10). Web-palveluiden ilmestymiseen johtanutta kehitystä ei voidakaan ohjelmistoalalla pitää kiistatta mullistavana poikkeuksena, vaan pikemminkin luontaisena teknologisena evoluutiona.

Onnistuneita, Tuxedo-ohjelmistokehystä (Transactions for Unix, Extended for Distributed Operations) hyväksi käyttäneitä toteutuksia hajautetuista järjestelmistä nähtiin 80-luvun loppupuolella (Rosen, 2008, sivut 3-7). 90-luvulla päästiin alustariippuvuudesta muun muassa CORBA-standardin (Common Object Request Broker Architecture) mukaisilla toteutuksilla (Rosen, 2008, sivut 7-10). Kyseisen vuosikymmenen Microsoft kulki omaa polkuaan tiukasti omaan alustaansa sidotulla DCOM-spesifikaatiolla (Distributed Component Object Model). (Kalin, luku 7.1)

Vaikka tekniset tavoitteet ovat osittain samoja myös uudemmissa toteutuksissa, englannin kielen sanat ”web service” yhdistettiin teknologiseksi käsitteeksi vuonna 2000 Microsoftin toimesta. Yrityksen perustama työryhmä ehdotti useita standardeja yhdistettynä otettavaksi käyttöön toteuttaessa koneiden välistä, verkon yli tapahtuvaa kommunikaatiota. Ehdotettuja standardeja olivat muun

muassa jo aiemmin paljon käytetyt XML (Extensible Markup Language) ja HTTP (Hypertext Transfer Protocol). (Josuttis, 2007, sivut 209–211)

1.2 Määritelmä

Määritellessään termiä web-palvelu Josuttis (2007, sivut 209–211) lainaa luvussa 1.1 mainitun alkuperäisen web-palvelutyöryhmän jäsentä Adam Bosworthia, joka määrittelee web-palvelun olevan mikä tahansa ohjelmistojen välisen kommunikaation mahdollistava arkkitehtuuri. Hän kuitenkin tarkentaa tätä määritelmää viittaamalla web-palvelulla ohjelmistojen yhteentoimivuuden mahdollistavaan standardikokoelmaan. Kokoelmassa tulee olla normit alustariippumattomalle formaatille, jolla siirrettävä data esitetään, sekä protokollalle, jolla ohjelmistojen välinen kommunikaatio tapahtuu. Josuttis esittää myös web-palveluiden viestien olevan itseään kuvailevia ja helppoja liikuttaa eteenpäin tietoverkossa. Luvussa 1.1 mainitussa, Microsoftin ensimmäisessä standardikokoelmassa, HTTP olisi kommunikaatioon käytettävä standardi ja XML datan esittämiseen käytettävä standardi.

W3C (World Wide Web Consortium) puolestaan määrittelee web-palvelun seuraavasti:

> *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."* (W3C, 2004)

Ohjelmistoalalle tuttuun termistön sekavaan käyttöön ei voi olla törmäämättä myöskään web-palvelun määrittelyjä vertaillessa. Josuttisin käyttämän määritelmän mukaan web-palvelun lähettämien viestien täytyy olla itseään kuvailevia, kun taas W3C:n määritelmässä ei vielä vaadita semanttisia viestejä. W3C:n

määritelmää ei myöskään ole julkaistu standardina, vaan työryhmän huomautuksena (W3C Working Group Note). (Josuttis, 2007, sivut 209–211) (W3C, 2004)

Semantiikan vaatiminen web-palvelulta erottaa web-palvelut aiemmin käytetyistä teknologioista. Organisaation tiedonsiirron lisäksi niillä voidaan toteuttaa semanttinen tietoverkko, jossa tietyn tyyppisiä palveluita etsivä sovellus voi suorittaa hakuja tarjolla olevien palveluiden metatietoihin. (de Bruijn, 2008, sivut 17–20)

1.3 Karkea alajaottelu arkkitehtuurin perusteella

Web-palveluina tarjottavien toimintojen monimutkaisuus ja laajuus vaihtelevat tarpeen mukaan yksinkertaisesta resurssin tarjoamisesta monimutkaiseen ohjelmistologiikkaan. Palvelut voidaan jakaa toteutusarkkitehtuurin mukaan kahteen SOA-arkkitehtuuria noudattavaan alaryhmään, joita seuraavaksi kuvaillaan. (Richardson, 2007, sivut 13–21)

1.3.1 REST

Yksinkertaisia web-palveluja voidaan toteuttaa REST (Representational State Transfer)-arkkitehtuuria noudattaen. REST-palveluiden sisäinen tila on pääteltävissä suoraan kutsuttavasta osoitteesta ja yksi palvelu toteuttaa vain yhden toiminnon. Palvelun kutsu on tyypillisesti deklaratiivinen, sisältäen kutsuttavan metodin ja kaikki metodille välitettävät tiedot ihmisen luettavassa formaatissa. (Richardson, 2007, sivut 13–21)

HTTP-siirtoprotokollaa käyttävää REST-arkkitehtuuria noudattavaa web-palvelua voitaisiin kutsua esimerkiksi seuraavalla osoitteella:

http://www.web-palveluntarjoaja.org/kuvat/etsi?tagi=koira

Javalla web-sovelluksia tehneet huomaavat, että esimerkin kaltainen toteutus olisi yksinkertaisimmillaan *web-palveluntarjoaja.org*-palvelimella */kuvat/etsi-*

polusta vastaava HttpServlet-tyyppinen palvelinsovelma, jonka GET-metodille välitetään *tagi*-nimisessä parametrissa arvo *koira*. Voidaanko tällaista yksinkertaista toteutusta pitää edes web-palveluna, riippuu määrittelystä. Esimerkiksi Richardson (2007, sivu 13), joka käsittelee kirjassaan REST-arkkitehtuuria noudattavien palveluiden toteuttamista, esittää kaikkien staattisten web-sivujen olevan REST-arkkitehtuurin mukaisia web-palveluita. Luvussa 1.2 mainitun Adam Boswortin määritelmässä vaadittu web-palveluiden koneellinen löydettävyys voidaan toteuttaa REST-palveluille esimerkiksi tarjoamalla niitä UDDI-rekisterin (ks. luku 1.5.4) kautta. (Battle, 2007, sivut 63–66)

1.3.2 RPC

Monimutkaista ohjelmistologiikkaa sisältävät web-palvelut perustuvat tavallisesti RPC-arkkitehtuuriin. Esimerkiksi kaikki SOAP-viesteillä (ks. luku 2.4.2) kommunikoivat web-palvelut ovat RPC-arkkitehtuurin mukaisia. REST-tyyppisistä palveluista poiketen RPC-palveluilla voi olla käyttäjälle näkymätön sisäinen tila, jolloin erilliset kutsut palveluun voivat riippua toisistaan. Myöskään palvelukutsu ei ole REST-palveluiden tapaan itseään kuvaileva, vaan sekä parametrit että tieto palvelulta pyydettävästä toiminnosta, esimerkiksi metodista, lähetetään käyttäjältä piilossa. HTTP-protokollaa käyttävässä palvelussa tämä tarkoittaa tietojen lähettämistä POST-metodilla. (Richardson, 2007, sivut 13–21)

Edellisessä luvussa esitetty *koira*-tagattujen valokuvien etsiminen voisi RPC-arkkitehtuurin mukaista palvelua kutsuttaessa näyttää tältä:

http://www.web-palveluntarjoaja.org/SOAPViestinKasittelija

Kutsussa käy ilmi vain, miltä palvelimelta palvelu löytyy, mutta kaikki muu piilotetaan HTTP-viestin POST-parametreihin. Näitä voisivat olla REST-esimerkkiä vastaavasti *kategoria=kuvat*, *metodi=etsi*, *tagi=koira*.

1.4 Karkea alajaottelu toteutustavan perusteella

Web-palveluiden käyttökohteet vaihtelevat aina laajoista SOA-arkkitehtuurin mukaisista toteutuksista yksittäisen olemassa olevan sovelluksen logiikan tarjoamiseen web-palveluna. Vaikka palvelua käyttävän osapuolen näkökulmasta sen sisäinen toteutustapa on merkityksetön, voidaan sillä ratkaisevasti vaikuttaa käsiteltävän tapauksen ratkaisuun.

1.4.1 Dokumenttikeskeinen lähestymistapa

Kehitettäessä web-palveluita dokumenttikeskeisen lähestymistavan mukaisesti on suunnittelun painopiste normaalisti palvelua kuvaavalla WSDL-dokumentilla ja sitä vastaavalla XML-skeemalla. WSDL-dokumentissa kuvataan tekniseen toteutukseen tarvittavat tiedot, kuten tuetut viestityypit ja saatavilla olevat operaatiot. Dokumentti on riippumaton ohjelmointikielestä ja palvelurungon generointi toteutuskielelle jätetään ohjelmistokehyksen tehtäväksi. (Jayasinghe, 2008, sivut 137–139)

Dokumenttikeskeinen lähestyminen mahdollistaa web-palveluiden suunnittelun ilman minkään ohjelmointikielen tuntemusta ja auttaa myös tiettyjen teknisten haasteiden ratkaisemisessa. Esimerkiksi Java-kielelle ominaisten tietotyyppien sekä rekursiivisten oliorakenteiden esitteleminen rakenteisessa WSDL-dokumentissa on varsin haastavaa, mutta XML-rakennetta vastaavan oliomallin generointi ei vaadi kompromisseja. (Poutsma, 2007, sivut 4-6)

Dokumenttikeskeisen lähestymistavan eduista huolimatta tietyt käyttötapaukset, kuten olemassa olevan järjestelmän operaatioiden tarjoaminen web-palveluina, soveltuvat ratkaistaviksi koodikeskeisellä lähestymistavalla. (Jayasinghe, 2008, sivu 137)

1.4.2 Koodikeskeinen lähestymistapa

Koodikeskeisessä lähestymistavassa tehdään ensin palvelun rungon toteutus kohdekielellä. Java-kieltä käyttäessä kirjoitetaan ensin palvelulogiikan toteuttavat luokat ja luokkiin operaatiot toteuttavat metodit. Ohjelmistokehykselle voidaan kertoa annotaatioilla tai erillisellä XML-dokumentilla, minkä operaation metodi toteuttaa. Merkityn ohjelmakoodin pohjalta web-palvelun toteuttava ohjelmistokehys luo palvelun rajapinnan määrittävän WSDL-dokumentin. (Jayasinghe, 2008, sivut 137–139)

Dokumenttikeskeiseen lähestymistapaan verrattuna koodikeskeisen tavan selkeisiin etuihin lukeutuu mahdollisuus toteuttaa web-palveluita ilman syvällisempää WSDL-tuntemusta. Myös toistuvien muutosten toteuttaminen, esimerkiksi protoiluvaiheessa, on koodikeskeisellä lähestymistavalla helpompaa. (Jayasinghe, 2008, sivut 137–138)

1.5 Keskeistä termistöä

Koska kyseessä on uusi aihealue ja yleisesti hyväksytyt perusteokset ovat vasta vakiintumassa myös termien määritelmät vaihtelevat lähteestä riippuen. TAULUKOSSA 1 esitellään tärkeimpiä web-palveluiden yhteydessä käytettäviä termejä ja kerrotaan niiden rooli teknisestä näkökulmasta.

TAULUKKO 1: Web-palveluiden keskeisiä termejä ja rooleja

Termi	**Rooli**
SOA - Lähestymistapa organisaation rakenteen palvelukeskeiseen mallintamiseen	**Arkkitehtuuri, joka voidaan toteuttaa web-palveluilla**
SOAP - Viestiformaatti rakenteisen tiedon välittämiseen hajautetussa ympäristössä	**Välittää dataa operaatiolta ja operaatiolle WSDL:ssä määriteltyjen porttien kautta**
WSDL - Määrittää palvelun käyttämät portit, tarjoamat operaatiot, tuetut viestinvälityskanavat ja viestiformaatit	**Palvelun tarkka kuvaus** - Mitä operaatiota ja mistä portista tarjotaan? Millaisia viestejä voidaan välittää? Mitkä ovat operaatiolle mahdollisesti vietävien parametrien tietotyypit?
UDDI - Alustariippumaton tapa löytää ja kuvailla web-palveluita ja niiden tarjoajia	**Palveluiden ja niiden korkean tason kuvausten löytäminen** - Mitä palveluita on saatavilla?

1.5.1 SOA

SOA eli palvelukeskeinen arkkitehtuuri on terminä varsin monessa yhteydessä eri tavoilla määritelty ja käytetty. Tässä tutkielmassa ei pohdita palvelukeskeisen arkkitehtuurin määritelmää syvällisemmin vaan käytetään monipuolisen ohjelmistokehysvalikoiman löytämiseksi mahdollisimman avointa määritelmää.

Palvelukeskeinen arkkitehtuuri on lähestymistapana ollut yleisesti tunnustettuna 90-luvun loppupuolelta saakka. SOA-arkkitehtuuria toteuttavat järjestelmät tarjoavat toteutustavasta riippumattoman rajapinnan kautta muille järjestelmille pääsyn tarjoamaansa palveluun. (Josuttis, 2007, sivut 7-8)

Järjestelmäarkkitehti Nicolai Josuttis (2007, sivut 8-10) määrittelee palvelukeskeisen arkkitehtuurin perustuvan teknisessä mielessä kolmeen käsitteeseen: palveluun (service), palveluväylään (service bus) ja löyhään kytkentään (loose coupling). Tässä tutkielmassa SOA-käsite laajennetaan, muun muassa Apache Software Foundationin johtajan Sam Rubyn ja ohjelmointiasiantuntijan Leonard

Richardsonin tavoin sisältämään myös REST-arkkitehtuurin mukaiset palvelut. (Richardson, 2007)

1.5.2 SOAP

Web-palveluiden näkökulmasta SOAP on standardoitu tapa paketoida palveluiden välittämiä viestejä. Viestin kaikki data on XML-muotoista. Teknisesti SOAP-spesifikaatio koostuu yhteisesti sovituista säännöistä, joilla alusta- ja kieliriippuvaiset tietotyypit sekä ohjelmistologiikan tila, esimerkiksi virhetilanteen sattuessa, esitetään alustariippumattomassa XML-tiedostossa. (Tidwell, 2002, sivut 15–24)

Ohjelmistotalo MindStream Softwaren pääarkkitehti ja johtaja Robert Englander (2001, sivut 47–52) muistuttaa kirjassaan, ettei spesifikaatio sido viestien käyttömahdollisuuksia mihinkään tiettyyn teknologiaan eikä edes web-palveluihin. Standardin mukaisia viestejä käytetään muun muassa dokumenttien liikuttamiseen EDI-järjestelemissä (Electronic Document Interchange). Tässä tutkielmassa SOAP-viestit ovat kuitenkin RPC-arkkitehtuurin mukaisia metodikutsuja, jolloin viestin sisältämässä XML-tiedostossa välitetään myös metodien parametreja ja palautusarvoja. Viestejä voidaan kuljettaa esimerkiksi HTTP-protokollalla tavallisten nettisivujen tapaan, mutta vastaanottajana on selaimen asemesta SOAP-viestin lukija (Monson-Haefel, 2003, Luku 4).

Alun perin SOAP oli lyhenne sanoista Simple Object Access Protocol. SOAP ei kuitenkaan ole kovin yksinkertainen eikä sillä varsinaisesti ole olioiden kanssa mitään tekemistä. Määritelmän versiosta 1.2 alkaen SOAP onkin ollut itsenäinen termi. (Josuttis, 2007, sivut 217–218)

SOAP:n kaltaisia mutta huomattavasti yksinkertaisempia ja helppokäyttöisempiä viestiformaatteja ovat esimerkiksi XML-RPC ja JSON-RPC (JavaScript Object Notation RPC). Nimestään huolimatta JSON-tuki on saatavilla useille ohjelmointikielille, muun muassa Javalle. Kaikki ovat kuitenkin RPC-

arkkitehtuuriin tarkoitettuja viestiformaatteja eikä niiden yksityiskohtainen vertailu ole tarpeen tämän tutkielman puitteissa.

1.5.3 WSDL ja WADL

WSDL (Web Service Description Language) on alun perin Microsoftin, IBM:n ja Ariban ehdottama standardi web-palveluiden liittymien määrittämiseen. Versio 2.0 hyväksyttiin W3C:n standardiksi vuonna 2007 (Taylor, 2009, sivu 275). Selkeyden vuoksi mainittakoon, että versio 1.2 uudelleennimettiin suurten muutosten takia versioksi 2.0. Kyseessä on kuitenkin sama spesifikaatio. (W3C, 2003) (W3C, 2007)

WSDL-dokumentin voidaan katsoa koostuvan kahdesta osakokonaisuudesta: palvelun kuvauksesta sekä toteutuksen määrittelystä. Näistä ensimmäistä tarvitaan, kun asiakas (esimerkiksi toinen web-palvelu) on löytänyt palvelun ja haluaa tietää, millä tekniikoilla löydetty palvelu on valmis kommunikoimaan. Jälkimmäisessä, toteutuksen määrittelyssä, puolestaan kerrotaan palvelua tarjoavalle ohjelmistokehykselle, mihin ja miten tähän palveluun osoitetut viestit tulee ohjata. (Englander, 2002, sivut 170–179) (Taylor, 2009, sivut 275–281)

Kehittäjän näkökulmasta web-palvelujen määritteleminen WSDL-dokumentissa parantaa koneiden keskinäisen tiedonvaihdon lisäksi myös palveluiden virheensietokykyä. WSDL-dokumentti on vahvasti tyypitetty ja palvelun käyttöön ottava asiakas saa ilmoituksen mahdollisesta virhetilanteesta jo käyttäjäksi rekisteröityessään. (Pautasso, 2008)

Versioon 2.0 saakka WSDL-dokumenteilla ei voitu määritellä REST-arkkitehtuurin mukaisia web-palveluita. Puutteen korjatakseen Sun Microsystems julkaisi vuonna 2006 WADL-spesifikaation (Web Application Description Language), jolla REST-palveluiden liittymät voidaan määritellä. Seuraavana vuonna julkaistuun WSDL 2.0-spesifikaatioon lisättiin tuki REST-palveluiden määrittelylle. (Hadley, 2006) (Pautasso, 2008)

1.5.4 UDDI

UDDI (Universal Description, Discovery and Integration) on rekisteri, jossa palveluiden tarjoajat voivat julkaista tietoja itsestään ja tarjoamistaan palveluista. UDDI on itsessään web-palvelu, joka mahdollistaa rekisterissä esiteltyjen palvelujen koneellisen löydettävyyden, tarjoten muun muassa hakutoimintoja palveluiden etsimiseen metatietojen perusteella. Rekisterissä tarjottava palvelu esitellään UDDI-palvelimelle spesifikaation mukaisella XML-dokumentilla. Dokumentin määrittely ei rajoita rekisterin käyttöä vain web-palveluiden esittelyyn, mutta tässä tutkielmassa käsitellään UDDI:a vain web-palveluiden rekisterinä. (Cerami, 2002, sivut 135–144)

Kuten edellisessä luvussa todettiin, voidaan web-palveluiden tekniset metatiedot esittää WSDL-dokumentissa. Palvelujen etsijäkonetta kiinnostaa kuitenkin teknisten tietojen asemesta etsittävän palvelun semantiikka. Tällaisen palvelujen luokittelun UDDI-rekisteri pyrkii toteuttamaan rekisteröidyiltä palveluilta keräämillään metatiedoilla. Rekisterin tarkoitus onkin auttaa etsijäkonetta löytämään oikeanlainen palvelu ja antaa etsijälle tiedot mistä löydetyn palvelun tekniset metatiedot, esimerkiksi WSDL-dokumentti, löytyvät. Teknisen (WSDL) ja semanttisen (UDDI) metatiedon eriyttäminen mahdollistaa myös palvelun teknisten rajapintojen muuttamisen ilman palvelun uudelleenrekisteröintiä. (McGovern [1], 2003, luku 6)

Microsoftin, IBM:n ja Ariban yhteistyössä kehittämä UDDI-spesifikaatio julkaistiin vuonna 2000 (Cerami, 2002, sivut 135–136). Monesta muusta web-palveluteknologiasta poiketen UDDI ei ole W3C:n standardi, vaan spesifikaatiosta vastaa nykyään lähes kolmensadan yrityksen ja yhteisön yhteenliittymä OASIS (Organization for the Advancement of Structured Information Standards). (OASIS, 2009 [1], OASIS, 2009 [2])

1.6 XML-jäsentimet (DOM, SAX, StAX ja VTD)

Web-palveluiden tehokkaan käytön perustana voidaan pitää tehokasta viestinvälitystä ja -prosessointia. Suorituskyvyn kannalta tärkein yksittäinen osa XML-viestejä käsittelevän ohjelmistokehyksen arkkitehtuurissa on viestin prosessointi ja muuttaminen ohjelmointikielessä tarvittavaksi tietomalliksi. Vaikka tässä tutkielmassa ei tulla vertailemaan ohjelmistokehysten suorituskykyä eikä paneuduta kehysten sisäiseen toteutukseen, on hyvä tiedostaa eri lähestymistapojen erot XML-viestien prosessointiin. (Jayasinghe, 2008, sivut 19–21)

Luvussa esiteltävät jäsennintyypit ovat määriteltyjä rajapintoja, joissa määritellään jäsentimen ulospäin tarjoamat palvelut. Rajapinnat toteutetaan useissa eri jäsentimissä, joiden välillä voi olla suuriakin eroavaisuuksia suorituskyvyssä. (Professional XML, 2007, sivut 145–150, 185–190)

Perinteisesti viestien prosessoijat on jaettu kahteen pääryhmään: DOM (Document Object Model)- ja SAX (Simple API for XML)-jäsentimet. DOM-jäsentimet luovat XML-viestistä dokumentin rakennetta vastaavan oliopuurakenteen ja säilyttävät sen keskusmuistissa. SAX-jäsentimet käsittelevät viestiä tietovirtana ja ilmoittavat kutsujalleen esimerkiksi elementin alku- tai loppukohdan sen havaittuaan. SAX-mallissa rakenteen ja datan käsitteleminen tai muistiin tallentaminen jätetään jäsentäjän kutsujan vastuulle. (Atay, 2005)

SAX-jäsentimissä voidaan omaksi alaryhmäkseen erottaa StAX-jäsentimet (Streaming API for XML). Keskeisin eroavaisuus niiden välillä liittyy dokumentista löydetyn datan palauttamiseen kutsujalle. SAX palauttaa löytämänsä datan aina jäsentimen kutsujalle, kun taas StAX palauttaa ensin metatietoja löytämästään elementistä ja antaa sitten jäsennintä kutsuneen osapuolen päättää, luetaanko elementin sisältämää dataa vai ei. (Lam, 2008)

Kolmas lähestymistapa viestien prosessoinnissa on VTD (Virtual Token Descriptor), joka viestiä läpikäydessään kerää dokumentin rakenteen kokonaislu-

kuina taulukkoihin. VTD-jäsennin luo XML-dokumentista kaksi taulukkoa, joista ensimmäiseen tallennetaan elementtien ja attribuuttien sijaintitiedot alkuperäisessä dokumentissa. Toiseen taulukkoon merkitään elementtien keskinäiset rakenteet eli kunkin elementin isä-, lapsi- ja sisarelementtien sijainnit dokumentissa. (Lam, 2008)

XML-prosessointia käsittelevässä artikkelissaan Cisco Systemsin suorituskykyasiantuntijat Brian Lam ja Jianxun Jason Ding yhdessä Texas A&M:n yliopiston professorin Jyh-Charn Liun kanssa kuvailevat eri lähestymistapoja seuraavasti:

> *"Sekä DOM, että VTD ovat hyviä dokumentin edestakaiseen läpikäyntiin. VTD suoriutuu jäsentämisestä nopeammin ja on vähäisiä muutoksia tehtäessä DOM:a parempi. DOM puolestaan soveltuu paremmin monimutkaisten ja toistuvien muutospyyntöjen toteuttamiseen. SAX ja StAX selviävät vähäisellä muistinkäytöllä, mutta eivät sovellu hyvin dokumentin edestakaiseen läpikäyntiin tai muutoksiin."* (Lam, 2008)

2 JAVA EE WEB-PALVELUIDEN NÄKÖKULMASTA

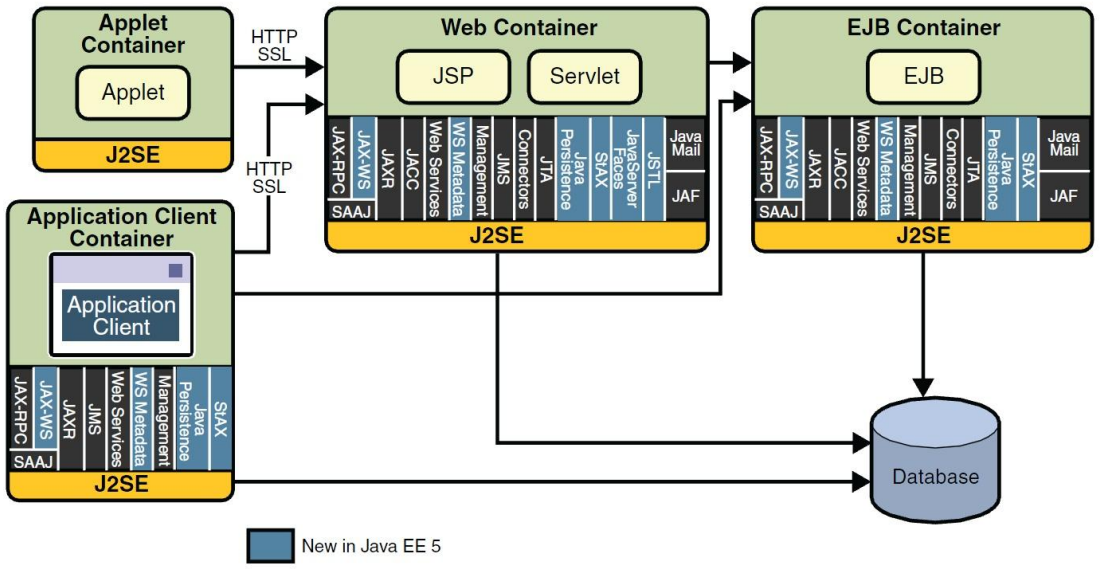
2.1 Java EE

Java-termiä käytettäessä voidaan tarkoittaa muun muassa Java-ajoympäristöä, Java-ohjelmointikieltä tai jotakin Java-ohjelmointikielen ohjelmistoalustaa. Ajoympäristö tarjoaa ohjelmistolle pääsyn alustajärjestelmän resursseihin, kuten keskusmuistiin ja oheislaitteisiin. Ohjelmistoalusta sisältää valmiita kirjastoja ohjelmistojen kehittäjille. Perinteisiä työpöytäohjelmia kehittäessä voidaan käyttää Java SE (Java Standard Edition)-standardin mukaista ohjelmistoalustaa. Alusta tarjoaa yleiskäyttöisiä kirjastoja muun muassa tiedostonhallintaan, laskentaan, tekstinkäsittelyyn sekä verkon käyttämiseen. (McGovern [2], 2003, sivut 3-5) (Sun Microsystems, 2004)

Java EE (Java Enterprise Edition) on Java SE:tä laajempi ohjelmistoalusta, joka tarjoaa valmiita kirjastoja laajojen, moduulirakenteisten järjestelmien kehittämiseen. Ohjelmistokehittäjälle Java EE tarjoaa normaalien työpöytäsovellusten lisäksi mahdollisuuksia web-sovellusten rakentamiseen sekä monia vaihtoehtoja sovellusten integroimiseen. Web-palveluita voidaan käyttää sekä Java EE-sovellusten keskinäiseen integraatioon että ulkoisten sovellusten kanssa kommunikoimiseen. (Jendrock, 2008, sivut 51–63)

2.2 Java EE:n rajapinta web-palveluille

Java EE-spesifikaatiossa määritellään web-palveluiden toteuttamiseen korkean tason JAX-WS (Java API for XML Web Services)-ohjelmointirajapinta, jolla voidaan tarjota järjestelmän moduuleita web-palveluina. KUVIOSSA 1 nähdään JAX-WS-rajapinnan olevan käytössä normaaleilla työpöytäsovelluksilla (Application Client Container), web-sovelluksilla (Web Container) sekä Java EE:n komponenttirajapinnan toteuttavilla olioilla (EJB Container). (Jendrock, 2008, sivut 51–63)



KUVIO 1: Java EE:n ohjelmointirajapinnat (Jendrock, 2008, sivu 57)

JAX-WS-standardin toteuttavan ohjelmistokehyksen käyttö web-palveluiden toteuttamiseen ei ole välttämätöntä eikä kaikissa tilanteissa edes suotavaa. Java EE on nimensä mukaisesti suunniteltu skaalautumaan suuriin, usean toteutusteknologian järjestelmiin, joka saattaa vähentää toteutuksen yksinkertaisuutta tai suorituskykyä. Standardi määrittää vain tietyt rajapinnat, jotka vaaditaan toteutettavaksi ohjelmistokehyksessä, mutta ei puutu varsinaiseen toteutukseen.

JAX-WS:n historia ulottuu vuoteen 2002 saakka, jolloin ensimmäinen versio JAX-RPC (Java API for XML-based RPC)-rajapinnasta julkaistiin. JAX-RPC:stä nähtiin seuraavana vuonna uudistettu versio 1.1, joka jäi viimeiseksi sillä nimellä julkaistuksi rajapinnaksi. REST-arkkitehtuurin mukaisten, resurssikeskeisten palvelujen nauttiessa kasvavaa huomiota haluttiin myös Javan web-palvelu rajapinnan RPC-keskeisyyttä vähentää. Spesifikaatio nimettiinkin uudelleen versiossa 2.0, jolloin lyhenteeksi tuli JAX-WS. (Hansen, 2007, sivut 1-7)

Aiempiin versioihin verrattuna JAX-WS:n suunnittelussa pääpaino on ohjelmistosuunnittelijan näkökulmasta ollut tuottavuudella ja helppokäyttöisyydellä.

Rajapinnassa määritelläänkin sekä koodikeskeinen että valmiisiin WSDL-määrityksiin perustuva dokumenttikeskeinen lähestymistapa. Toteutuksen helpottamiseksi JAX-WS mahdollistaa normaalin Java-luokan metodien tarjoamisen web-palveluina JAX-WS-annotaatiota käyttäen. (Hansen, 2007, sivut 36–42)

RPC-lyhenteen poistamisen myötä versiossa 2.0 määritelmään lisättiin tuki REST-arkkitehtuurin mukaisille web-palveluille. Myös REST-palvelujen luominen on mahdollistettu suoraan annotaatioilla eikä yksinkertaista web-palvelua toteuttaessa tarvitse välttämättä käyttää lainkaan normaaleiden Java EE-web-sovelluksien vaatimia XML-pohjaisia alustamäärityksiä (deployment descriptors). (Hansen, 2007, sivut 131–135)

JAX-WS on rajapinnan spesifikaatio, joka jättää sisäisen toteutuksen ohjelmistokehyksen toteuttavan osapuolen valittavaksi. Tutkielman luvussa 3 käsitellyistä web-palveluiden ohjelmistokehyksistä kolme (Axis2, CXF ja Service Component Architecture) toteuttaa JAX-WS-rajapinnassa määritellyt toiminnallisuudet. Yhteensopivuus rajapintaan mahdollistaa kehysten käyttämisen Java EE-sovelluspalvelimissa web-palveluita toteuttavana moduulina. JAX-WS:ssä määriteltyjen ominaisuuksien lisäksi mainitut kehykset sisältävät myös spesifikaatioon kuulumattomia laajennuksia. Pelkän JAX-WS-rajapinnan toteuttavia avoimen lähdekoodin ohjelmistokehyksiä on myös saatavilla. Sellainen on muun muassa referenssitoteutus JAX-WS RI.

3 WEB-PALVELUIDEN OHJELMISTOKEHYKSET JAVASSA

Muiden web-palveluiden kanssa yhteen toimivien web-palveluiden toteuttaminen on valitusta arkkitehtuurista riippumatta monivaiheinen prosessi. Palvelun täytyy vähintäänkin kommunikoida jotain viestinvälityskanavaa pitkin ja monimutkaisempaa logiikkaa sisältävissä palveluissa käytetään useimmiten myös jotain tekstimuotoista viestiformaattia, esimerkiksi SOAP:ia. Web-palvelua alusta asti Javalla toteuttaessa tämä tarkoittaisi muun muassa verkon tietovirtojen käsittelemistä sekä palvelun tekstimuotoisten viestien luomista ja jäsentämistä.

Web-palveluiden ohjelmistokehykset, muiden ohjelmistokehysten tapaan, tarjoavat toteuttajan tarvitsemia taustatoimintoja sekä ohjaavat toteutettavan ohjelmiston sisäistä tiedonkulkua. Web-palveluiden tapauksessa tämä voi tarkoittaa muun muassa tehokasta viestinvälitystä ja XML-viestien muuntamista oliorakenteeksi. Useimmissa kehyksissä pyritään abstraktiotasoa nostamalla luomaan toteuttajalle edellytykset keskittyä yksinomaan toteutettavan palvelun tarjoamaan toiminnallisuuteen.

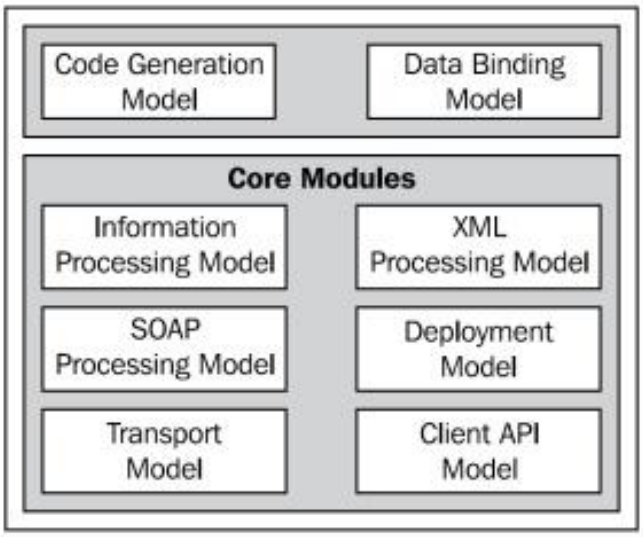
Seuraavaksi esitellään lähestymistavoiltaan ja arkkitehtuureiltaan varsin paljon toisistaan poikkeavia Java-kielellä toteutettuja web-palveluiden ohjelmistokehyksiä. Vertailuun on valittu verkossa eniten huomiota saaneet oman lähestymistapansa ja arkkitehtuurinsa ohjelmistokehykset. Kehyksiä vertaillessa on hyvä tiedostaa, ettei ole järkevää etsiä kaiken kattavaa vaihtoehtoa vaan tunnistaa tapauskohtaiset vaatimukset ja valita niiden täyttämiseen parhaiten soveltuva työkalu.

3.1 Apache Axis2

Apache Axis-projektin historia ulottuu ensimmäisiin web-palvelu toteutuksiin aina 2000 luvun alkuun saakka. Versio 1.0 versio julkaistiin vuonna 2002, jonka jälkeen Apache Axis nousi yhdeksi suosituimmista web-palveluiden ohjelmis-

tokehyksistä. Web-palvelujen käytön yleistyessä vaadittiin yhä tehokkaampia ja luotettavampia ohjelmistokehyksiä niiden toteuttamiseen. Ensimmäisen Axis-version ollessa jo laajalti käytössä merkittävien arkkitehtuurimuutosten toteuttaminen siihen olisi ollut ongelmallista. Vanhan muokkaamisen asemesta ASF:n (Apache Software Foundation) web-palvelutyöryhmä päätyikin kokonaan uuden ohjelmistokehyksen suunnitteluun. (Jayasinghe, 2008, sivut 14–16)

KUVIOSSA 2 esitetty Axis2:n moduulirakenteinen arkkitehtuuri jakautuu kahteen moduulikokoelmaan, joista ydinmoduulit sisältävät vain SOAP-viestien käsittelyyn liittyviä toimintoja. Kaikki kehyksen vastaanottamat viestit muutetaankin välitystasolla SOAP-viesteiksi, vaikka ne kehykseen saapuessaan olisivat muun tyyppisiä. (Jayasinghe, 2008, sivu 19)



KUVIO 2: Apache Axis 2 arkkitehtuuri (Jayasinghe, 2008, sivu 20)

Kehykseen on toteutettu monipuolinen sisäinen navigaatiologiikka (Flow), joka mahdollistaa ohjelmakoodin suorittamisen, esimerkiksi viestin saapumisen jälkeen, useassa eri vaiheessa. Logiikkaan voidaan määritellä suoritusvaiheita (Phase), joiden tehtävänä on ajaa määrätyt käsittelijät (Handler) määrätyssä järjestyksessä. Käsittelijä on kehyksessä määritelty rajapinta, joka itse kirjoitetun Java-luokan tulee toteuttaa. (Jayasinghe, 2008, sivut 45–58)

Yksi Axis2-arkkitehtuurin periaatteista on logiikan ja datan eriyttäminen. Käsittelijöiden välillä data kuljetetaan kehyksen määrittelemässä yhtä viestiä vastaavassa viestikontekstissa (MessageContext). Viestikonteksti yhdessä kehyksen neljän muun kontekstin kanssa (OperationContext, ServiceContext, ServiceGroupContext, ConfigurationContext) muodostaa puumallisen kontekstirakenteen, jossa alempana oleva konteksti näkee ylemmän kontekstin sisältämät tiedot mutta ei hierarkiassa samalla tasolla tai alempana olevien kontekstien sisältöjä. (Jayasinghe, 2008, sivut 71–82)

Axis2 toteuttaa JAX-WS-standardissa määritellyt rajapinnat ja tukee siten sekä RPC- että REST-arkkitehtuureita. (Jayasinghe, 2008, sivut 147–148) (Apache Axis2, 2008) JAX-WS-spesifikaation mukaisesti myös lähestymistavoista tuetaan sekä koodi- että dokumenttikeskeistä. (Jayasinghe, 2008, sivut 84–95)

Hyvää: Kattava paketti. Laajennettavissa. Paljon esimerkkejä ja materiaalia.

Huonoa: Monimutkainen sisäinen toteutus. Paljon konfiguroitavaa.

3.2 Apache CXF / FUSE Services framework

Toisen nykyään ASF:n alaisuudessa kehitettävän ohjelmistokehyksen tarina alkoi vuonna 2006, kun ObjectWebin kehittämä Celtix ja Codehausin suojissa ollut XFire yhdistettiin. Uuden, ASF:n alaisuudessa julkaistun projektin nimeksi annettiin CXF. (Apache, 2006)

CXF-projektin heikon dokumentaation vuoksi tutkielmassa käytetään myös Progress Software-yrityksen CXF-yhteensopivan, mutta kaupallisesti tuetun FUSE Services Framework (FUSE SF)-ohjelmistokehyksen dokumentteja.

Ohjelmistokehyksen suunnittelussa lähtökohtana on ollut keveys sekä yhteensopivuus muihin järjestelmiin. Muiden järjestelmien tehokkaan käytön voi havaita jo ennen ohjelmistokehyksen käyttöönottoa. XML-pohjaisesti palveluja

määritellessä luodaan Spring-ohjelmistokehyksen mukaisia XML-dokumentteja, jotka CXF käynnistyessään käsittelee. (Progress Software Corporation, 2008, sivut 11–14)

CXF sisältää toteutuksessaan tuen JAX-WS-mukaisten web-palvelujen toteuttamiselle. Muiden avoimeen lähdekoodiin perustuvien kehyksien kautta luvataan kommunikoinnin onnistuvan myös muun muassa CORBA- ja SCA-järjestelmien kanssa. (Apache CXF, 2009)

Hyvää: FUSE SF:n dokumentaatiot sopivat CXF:ään. Erittäin monipuoliset integraatiomahdollisuudet.

Huonoa: Osaa tehdä monta asiaa → paljon opeteltavaa

3.3 Restlet

Muista ohjelmistokehyksistä poiketen vuonna 2005 julkaistu Restlet perustuu puhtaasti REST-arkkitehtuurin tukemiseen. Kehyksen suunnittelussa pyrittiin toteuttamaan mahdollisimman tarkasti R. T. Fieldingin väitöskirjassaan esittelemät REST-arkkitehtuurin osat ja niiden keskinäiset riippuvuudet. Teknisessä suunnittelussa on pyritty yksinkertaisuuteen ja joustavuuteen. (Richardson, 2007, sivut 343–345) (Fielding, 2000, sivut 86–103)

Restletin perusarkkitehtuuri ei ota kantaa siihen, ollaanko toteuttamassa web-palvelua käyttävää asiakasohjelmistoa vai web-palvelun tarjoavaa ohjelmistoa. Kehyksellä toteutettu ohjelmisto voikin olla samaan aikaan sekä asiakas että palveluntarjoaja. Myöskään web-palvelusta palautettavaa dataa ei ole sidottu tiettyyn formaattiin, vaan kehyksellä voidaan toteuttaa yhtä lailla esimerkiksi XML-muotoisia vasteita antavia web-palveluja kuin HTML-merkittyjä, selaimella katseltavia sivuja palauttavia web-palveluja. (Richardson, 2007, sivut 344–346)

Hyvää: Selkeästi yhdellä tavalla ongelmia ratkaiseva. Ajattelutapaan tottumisen jälkeen myös web-sovellusten tekeminen samalla tavalla mahdollista. Laajennusosia esim. annotaatioperustaiseen palvelun määrittelyyn saatavilla.

Huonoa: Ei RPC-tukea.

3.4 Hessian

Toisen useimmista muista ohjelmistokehyksistä poikkeavan Hessianin lähtökohtana on siirrettävän datan sarjallistaminen binäärimuotoon. Palvelusta lähetettävää dataa ei muunneta SOAP-viestin vaatimaan XML-muotoon eikä paketoida palveluntarjoajan toimesta teknologiariippumattomaan kuoreen. Kehyksen palveluita ei myöskään määritellä WSDL-dokumenteilla eikä siten voida julkaista UDDI-rekistereissä. (Caucho Technology, 2008 [1])

Määritelmästä riippuen Hessian voitaisiinkin luokitella yhtä lailla sovelluksen sisäiseksi tiedonsiirtoprotokollaksi kuin web-palveluiden toteuttamiseen suunnitelluksi ohjelmistokehykseksi. Tässä tutkielmassa käytetyssä määritelmässä web-palveluilta ei kuitenkaan vaadita semantiikkaa eikä tietynlaista yhteentoimivuutta, joten Hessianin kaltaisen binääriprotokollan tutkiminen web-palvelun ohjelmistokehyksenä on mahdollista.

Web-palvelujen yhteentoimivuuden mahdollistamiseksi palvelun käyttäjän tulee tietää palvelun rajapinta. SOAP-viesteillä kommunikoivissa web-palveluissa rajapinta määritellään WSDL-dokumentissa, jonka palvelua käyttävä osapuoli lukee. Yhteentoimivuuden tuki Hessian-protokollassa on toteutettu lupaamalla yhteentoimivuus vain staattisesti tyypitettyjen, reflektiota tukevien olio-ohjelmointikielien välillä. Tällöin palvelun rajapinnan toteutus usealle eri kielelle voidaan tehdä mekaanisesti reflektiota käyttäen. Palvelun rajapintana käytetään suoraan palvelun toteutuskielen rajapintaa. Esimerkiksi Java-kielen Hessian-toteutuksella rakennetun web-palvelun rajapinnaksi tulisi suoraan palvelun määrittävä Java-kielen liittymä. (Caucho Technology, 2008 [2])

Hessian-protokollalle ominaisen sarjallistamisen ansiosta Hessian on voitu toteuttaa staattisesti tyypitettyjen kielten lisäksi useille dynaamisesti tyypitetyille kielille. Vaikka yhteentoimivuus luvataan jaetuilla rajapinnoilla staattisesti tyypitettyjen kielten välillä, merkitään sarjallistettaessa jokaiselle attribuutille myös sen tietotyyppi. Tällöin myös dynaamisesti tyypittävien kielten oliot voidaan sarjallistaa ja tarjota Hessianin web-palvelusta. (Caucho Technology, 2008 [2])

Protokollassa tuetaan tavallisten tietotyyppien, kuten kokonaisluvun ja merkkijonon lisäksi tietorakenteita, muun muassa listaa ja avainnettua kokoelmaa. Myös olioita sekä monimutkaisia olioverkkoja voidaan sarjallistaa. (Ferguson, 2007)

Hessian-toteutuksia on saatavilla ainakin seuraaville ohjelmointikielille: Java, C++, C#, Python, Ruby, PHP, Objective-C, Objective-C 2.0, D ja Erlang.

Hyvää: Erittäin yksinkertainen ottaa käyttöön, dataa ei tarvitse muuttaa tekstiksi ja takaisin oliokielen tietotyypiksi → varsin tehokas.

Huonoa: Vain olio-ohjelmointikielille. Vaatii Hessian-toteutuksen. Ei semantiikkaa.

3.5 XINS

Alun perin Alankomaisen Online Breedband B.V.-yrityksen kehittämälle XINS (XML Interface for Network Services)-ohjelmistokehykselle on ominaista poikkeuksellisen pitkälle viety dokumenttikeskeinen lähestymistapa. (De Haan, 2006, sivu 27)

Jokaisen XINS-kehyksellä toteutetun web-palvelun kehittäminen aloitetaan luomalla palvelua kuvaavat XML-tiedostot. Tiedostoissa määritellään palvelun yleiset tiedot, julkinen rajapinta, sisäiset lokikirjoittamissäännöt sekä generoitavassa rajapinnassa määriteltävät operaatiot. XML-tiedostojen pohjalta generoi-

daan palvelun dokumentaatio, WSDL-määrittelyt sekä Java-luokat yksikkötesteineen kullekin toteutettavalle operaatiolle. Dokumenteista voidaan generoida myös palvelua käyttävät Java-luokat asiakasohjelmistoa varten. (Goubard, 2008, sivut 5-11)

Dokumenttikeskeinen lähestymistapa mahdollistaa palvelujen tehokkaan uudelleenkäytön esimerkiksi dokumenttipohjia luomalla. Toisaalta monituiset XML-tiedostot ja niiden pohjalta koodin generoiminen monimutkaistavat yksinkertaisen web-palvelun toteuttamisprosessia. Koodin generointiin tutustuneiden ohjelmistokehittäjien tuntemat haasteet tulevat vastaan myös XINS-ohjelmistokehyksessä. Kaksisuuntaista generointia ei tueta, joten rajapinnan muuttaminen tehdään aina alkuperäiseen XML-dokumenttiin ja koodi generoidaan uudelleen. (De Haan, 2006, sivu 28–29)

XINS sisältää tuen sekä REST-arkkitehtuurin mukaiseen että RPC-tyyliseen palveluiden tarjoamiseen. Kehyksessä on valmiiksi toteutettuna SOAP-viestiformaatin lisäksi tuki muun muassa XML-RPC- ja JSON-RPC-viesteille. (Goubard, 2008, sivut 42–47)

Hyvää: Dokumenttikeskeinen → dokumentaatio on aina toteutuksen kanssa yhtenevä. Toimivan palvelun luominen ei vaadi laajaa ohjelmointiosaamista. Mahdollistaa tehokkaan uudelleenkäytön.

Huonoa: Yksinkertaisen web-palvelun toteuttamien vaatii monta XML-dokumenttia. Koodin generoimisen haasteet.

3.6 Spring-WS

Spring-WS (Spring Web Services) on yksi SpringSource-yrityksen avoimen lähdekoodin ohjelmistokehyksistä. Spring-WS:ssä korostetaan joustavuutta yksittäisten moduuleiden toteutusvaihtoehtojen välillä. Sen arkkitehtuuri perustuu IoC (Inversion of Control)-periaatteeseen, joka mahdollistaa muun muassa ke-

hyksen sisäisen moduulin vaihtamisen ohjelmakoodia muuttamatta. (Poutsma, 2007, sivu 2) Esimerkiksi XML-viestien käsittelemiseen voi halutessaan käyttää normaalien SAX- ja DOM-jäsentimien lisäksi suoraan Java-olioita luovia järjestelijöitä (marshallers). (Poutsma, 2007, sivut 30–33)

Palvelun logiikan toteutukseen Spring-WS tarjoaa varsin monipuolisia toteutusvaihtoehtoja. Sisäisenä, viestistä luotavana oliomallina voidaan käyttää esimerkiksi Axis 2-kehyksen AXIOM-oliorakennetta tai standardia SAAJ (SOAP with Attachments API for Java)-toteutusta. Itse kirjoitettua ohjelmakoodia voidaan määritellä ajettavaksi Axis 2:n tavoin useassa eri vaiheessa, kuitenkaan pakottamatta ennalta määrätyn sisäisen navigaatiologiikan noudattamiseen. (Poutsma, 2007, sivut 24–33)

Lähestymistavoista vain dokumenttikeskeinen on tuettu. XINS-kehyksen tapaisia metadokumentteja palvelun WSDL-määritysten luomiseen ei kuitenkaan käytetä vaan ohjelmakoodi generoidaan valmiin WSDL-määrityksen pohjalta. (Poutsma, 2007, sivu 4)

InfoQ-sivuston haastattelema Spring-WS-projektin perustaja Arjen Poutsma kertoo, että REST-tuki jätettiin tietoisesti pois kehyksen ensimmäisestä versiosta mutta suunniteltiin toteutettavaksi myöhemmissä versioissa. (Tilkov, S. 2007) Myöhemmin Poutsma kuitenkin kirjoittaa blogissaan REST-tuen toteuttamisesta toiseen SpringSourcen ohjelmistokehykseen mainiten samalla sen jäävän pois Spring-WS:stä. (Poutsma, 2009)

Hyvää: Yhteensopivuus sekä SpringSourcen että ulkopuolisten ohjelmistokehyksien kanssa. Täysi IoC-arkkitehtuuri → voi muokata juuri siellä ja sieltä mistä haluaa. Jatkuvuuden takeena menestyvä SpringSource.

Huonoa: Ei JAX-WS-standardin mukainen. Ei sisällä REST-tukea. Ei tue koodikeskeistä lähestymistapaa.

3.7 SCA ja SDO

SCA (Service Component Architecture) on SOA-toteutus, jonka yksittäinen komponentti voi olla esimerkiksi web-palvelu. Yrityksen liiketoiminnan näkökulmasta koko yrityksen ja tarvittavien sidosryhmien tiedon kulku voidaan mallintaa yhdessä SCA-kohdealueessa (domain), jossa määritetään XML-dokumenteilla kohdealueessa tarvittavat komponenttikokoelmat eli komposiitit. SCA-määrittely on alusta- ja kieliriippumaton. (Chappell, 2007, sivut 3-7) Avoimen lähdekoodin toteutuksia on saatavilla Java- ja C++-kielillä. (OSOA, 2008)

SDO:ta (Service Data Object) voisi kuvailla SCA:n komponenttien viestinnän kirjekuoreksi. SDO määrittää rajapinnan, jota SCA:n sisällä liikuteltavien viestien tulee toteuttaa. Käytännössä SDO on yhteinen rajapinta tiedon eri esitysmuodoille. SDO-tieto-olioita voidaan nykyisillä toteutuksilla tallettaa esimerkiksi relaatiokantaan tai XML-tiedostoon, mutta voidaan yhtä hyvin tarjota myös SOAP-viestiin sisällytettyinä web-palveluiden syötteinä ja vasteina. (Resende, 2007)

Web-palvelun luominen SCA-ympäristöön ei poikkea määrityksiltään muun tyyppisistä SCA:n tarjoamista palveluista. Komposiitti voi sisältää monta komponenttia, joista yksi voi olla määritelty olevan verkon yli kutsuttava web-palvelu. Palvelu voidaan toteuttaa dokumenttikeskeisesti, jolloin toteutettavan ohjelmakoodin tulee täyttää ennalta määritellyn WSDL-dokumentin vaatimukset. SCA tukee myös koodikeskeistä lähestymistapaa, jolloin jo toteutetun ohjelmakoodin pohjalta luodaan palvelun WSDL-dokumentti. Java-kielisessä SCA-toteutuksessa dokumentti voidaan luoda esimerkiksi SCA-spesifikaatiossa määriteltyjen annotaatioiden perusteella. (Holdsworth, 2007) (The Apache Software Foundation, 2008)

Vaikka joihinkin SCA-toteutuksiin on saatavilla kolmannen osapuolen REST-arkkitehtuurin mukaisia web-palveluja tukevia laajennuksia, spesifikaatiossa

tuetaan suoraan vain SOAP-viesteillä kommunikoivia RPC-arkkitehtuurin mukaisia web-palveluja. (Holdsworth, 2007)

SCA-järjestelmän yksittäinen web-palvelu komponentti voidaan määritellä käyttämään JAX-WS-rajapinnan mukaista toteutusta, mutta kaikki SCA-web-palvelut eivät kuitenkaan ole JAX-WS-yhteensopivia. (OSOA, 2007)

Hyvää: Monipuolinen ja laajennettavissa oleva palvelualusta. Yksittäisen palvelun lisääminen olemassa olevaan SCA-järjestelmään on varsin helppoa.

Huonoa: Laaja ja paneutumista vaativa kokonaisuus. SOAP-viestien sisältö SDO-muotoista XML-dataa (vastaanottajan tuettava SDO:ta).

4 YHTEENVETO

Web-palveluiden toteuttamiseen Java-kielellä kirjoitetussa ohjelmistossa on useita varsin pitkälle kehittyneitä vaihtoehtoja. Uuden alueen ollessa kyseessä teknologiakenttä kehittyy jatkuvasti ja uusia monipuolisuuden sekä helppokäyttöisyyden yhdistäviä ohjelmistokehyksiä pyritään kehittämään. Web-sovelluksia Javalla tehneet ovat todennäköisesti huomanneet tyrmistyttävän laajan ohjelmistokehysvalikoiman. Java-Source.net (Java-Source.net [1]) listaa pelkästään avoimen lähdekoodin ohjelmistokehyksistä yli 50 vaihtoehtoa Java-kehittäjälle. Web-palveluiden osalta vastaavien vaihtoehtojen määrä on kuitenkin vielä kohtuullisempi: alle 20 ohjelmistokehystä (Java-Source.net [2]). Vaihtoehtoisten toteutusten määrä ei takaa ohjelmistokehysten korkeaa laatua, mutta siitä voi päätellä web-palveluiden toteuttamisen Javalla nauttivan vähintään kohtalaista suosiota.

Tutkielman tarkoituksena oli kartoittaa Java-kielellä kirjoitettuja avoimen lähdekoodin web-palveluiden ohjelmistokehyksiä. Keskeisin tavoite oli hahmottaa eri lähestymistapojen erot ja luoda ohjelmistosuunnittelijan näkökulmasta kokonaiskuva vaihtoehtoisista ohjelmistokehyksistä. Vertailtavia ohjelmistokehyksiä valittaessa käytettiin varsin avointa web-palvelun määritelmää, joka mahdollisti laajan lähestymistapojen kirjon. Käytännön kokeiluja ei tutkielman puitteissa toteutettu, joten arviot suorituskyvystä sekä helppokäyttöisyydestä tehtiin kunkin ohjelmistokehyksen dokumentaation pohjalta.

Tärkeänä osana onnistunutta ohjelmistosuunnittelua voidaan pitää oikean työkalun valitsemista kunkin tehtävän toteuttamiseen. Sama pätee myös web-palveluita toteuttaessa. Yhtenä tärkeimmistä ohjelmistokehyksen valintaan vaikuttavista asioista voidaan pitää web-palveluna tarjottavan ohjelmiston elinkaaren vaihetta. Jos palveluna tarjottava ohjelmistologiikka on vielä suunnitteluasteella tai suuretkin arkkitehtuurimuutokset mahdollisia toteuttaa, saattavat dokumenttikeskeistä lähestymistapaa tukevat ohjelmistokehykset tarjota yllä-

tyksettömämmän ja ylläpidettävämmän ratkaisun. Jos puolestaan vaatimuksena on jo olemassa olevan järjestelmän ohjelmistologiikan tarjoaminen web-palveluna eikä ohjelmistoa voida tai haluta muuttaa, saattaa koodikeskeistä lähestymistapaa tukeva kehys olla parempi tai jopa ainoa vaihtoehto.

Web-palvelua käyttävän osapuolen näkökulmasta olennaisin asia on palvelun rajapinta ja tuetut viestiformaatit. Palvelua toteuttavan ohjelmistosuunnittelijan on hyvä ottaa huomioon REST- ja RPC-arkkitehtuurien erot sekä valinnasta aiheutuvat rajoitteet. Yksinkertaista web-palveluita tukevaa web-sovellusta kehittäessä kannattaa varmastikin perehtyä vertailun ainoaan puhtaasti REST-arkkitehtuurin mukaiseen Restlet-ohjelmistokehykseen. Jos toisaalta jo suunnitteluvaiheessa tiedetään web-palveluiden tulevan osaksi yrityksen liiketoimintaa mallintavaa SOA-arkkitehtuuria, ohjelmistokehyksen laajennettavuus ja integroitavuus nousevat keskeiseen asemaan. Mahdollisimman monipuolista rajapintaa tavoitellessa on luontevaa tarkastella lähemmin molempia arkkitehtuureja tukevia ohjelmistokehyksiä.

Päätös web-palveluiden toteuttamiseen valittavasta ohjelmistokehyksestä kannattaa tehdä tapauskohtaisesti huomioiden ainakin toimintaympäristön asettamat vaatimukset, välittömän tarpeen, johon web-palvelua käytetään sekä suunnitelmat mahdollisesta jatkokehityksestä. TAULUKOSSA 2 esitellään kootusti ohjelmistokehysten tukemat arkkitehtuurit (Luku 1.3), lähestymistavat (Luku 1.4) sekä soveltuvuus Java EE:n määrittämän web-palvelurajapinnan toteutukseksi (Luku 2.2).

TAULUKKO 2: Tuetut arkkitehtuurit ja lähestymistavat

	Arkkitehtuurit		Lähestymistavat		JAX-WS
	REST	RPC	Dokumenttikeskeinen	Koodikeskeinen	rajapinnan toteutus
Apache Axis2	X	X	X	X	X
Apache CXF	X	X	X	X	X
Restlet		X		X	
Hessian				X	
XINS	X	X	X		
Spring WS		X	X		
SCA / SDO	X	X	X	X	X

LÄHTEET

Apache Axis2. 2008. JAX-WS Guide. [viitattu 2.4.2009] <http://ws.apache.org/axis2/1_4/jaxws-guide.html>

Apache CXF. 2009. [viitattu 02.04.2009] Apache CXF: An Open Source Service Framework <http://cxf.apache.org/>

Apache. 2006. Celtix Xfire Proposal. [viitattu 2.4.2009] <http://wiki.apache.org/incubator/CeltiXfireProposal>

Atay, M., Chebotko, A., Liu, D., Lu, S., Fotouhi, F. 2005. Efficient schema-based XML-to-Relational data mapping. Department of Computer Science, Wayne State University, Detroit.

Battle, R., Benson, E. 2007. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). Web Semantics: Science, Services and Agents on the World Wide Web 6 (2008) 61–69.

Berners-Lee, T. 2002. The World Wide Web, Past Present and Future: Exploring Universality. [viitattu 23.1.2009] <http://www.w3.org/2002/04/Japan/Lecture.html>

Birbeck, M., Kay, M., Livingstone, S., Mohr, S., Pinnock, S., Loesgen, B., Livingston, S., Martin, D., Ozu, N., Seabourne, M., Baliles, D. 2007. Professional XML. Indianapolis: Wiley Publishing, Inc.

Caucho Technology, Inc. 2008 [1]. Hessian Overview. [viitattu 23.4.2009] <http://hessian.caucho.com/doc/hessian-overview.xtp>

Caucho Technology, Inc. 2008 [2]. Hessian Overview. [viitattu 23.4.2009] <http://hessian.caucho.com/doc/metaprotocol-taxonomy.xtp>

Cerami, E. 2002. Web Services Essentials. O'Reilly.

Chappell, D. 2007. Introducing SCA. Chappell & Associates. [viitattu 27.4.2009] <http://www.davidchappell.com/articles/Introducing_SCA.pdf>

de Bruijn, J., Fensel, D., Kerrigan, M., Keller, U., Lausen, H., Scicluna, J. 2008. Modeling Semantic Web Services. Berlin: Springer.

De Haan, E. 2006. Frameworks and XINS. [viitattu 24.4.2009] <http://xins.sourceforge.net/presentations/frameworks_and_xins.pdf>

Englander, R. 2002. Java and SOAP. Sebastobol: O'Reilly & Associates, Inc.

Ferguson, S. Ong, Emil. 2007. Hessian 2.0 Serialization Protocol. [viitattu 23.4.2009] <http://hessian.caucho.com/doc/hessian-serialization.html>

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Irvine: University of California

Goubard, A. 2008. XINS User Guide. [viitattu 24.4.2009] <http://xins.sourceforge.net/docs/XINSGuide.pdf>

Hadley, M. 2006. Web Application Description Language (WADL). Santa Clara: Sun Microsystems, Inc. <http://research.sun.com/techrep/2006/smli_tr-2006-153.pdf>

Hansen, M. D. 2007. SOA Using Java Web Services. Upper Saddle River: Pearson Education, Inc.

Holdsworth, S. Ielceanu, S. Karmarkar, A. Little, M. Patil, S. Rowley, M. 2007. SCA Web Service Binding Specification V1.00. OSOA. [viitattu 27.4.2009] <http://www.osoa.org/download/attachments/35/SCA_WebServiceBinding_V100.pdf?version=2>

IETF. 1976. A High-Level Framework for Network-Based Resource Sharing. [viitattu 23.1.2009] <http://tools.ietf.org/html/rfc707>

Java-Source.net [1]. Open Source Web Frameworks in Java. [viitattu 22.1.2009] <http://java-source.net/open-source/web-frameworks>

Java-Source.net [2]. Open Source Web Services Tools in Java. [viitattu 22.1.2009] <http://java-source.net/open-source/web-services-tools>

Jayasinghe, D. 2008. Quickstart Apache Axis2. Birmingham: Packt Publishing Ltd.

Jendrock, E. Ball, J. Carson, D. Evans, I. Fordin, S. Haase, K. 2003. The Java EE 5 Tutorial. 2008. Sun Microsystems.

Josuttis, N. M. 2007. SOA in Practice. Sebastopol: O'Reilly Media, Inc.

Kalin, M. 2009. Java Web Services: Up and Running. Sebastopol: O'Reilly Media, Inc.

Lam, T., Ding, J., Liu, J. 2008. XML Document Parsing: Operational and Performance Characteristics. IEEE Computer Society, SEP 2008 30-37.

Marks, E. A. 2003. Executives Guide To Web Services. Hoboken: John Wiley & Sons, Inc.

McGovern [1], J. Tyagi, S. Stevens, M. Matthew, S. 2003. Java Web Services Architecture. San Francisco: Morgan Kaufmann Publishers.

McGovern [2], J. Adatia, R. Fain, Y. Gordon, J. Henry, E. Hurst, W. Jain, A. Little, M. Nagarajan, V. Oak, H. Phillips, L. A. 2003. Java™ 2 Enterprise Edition 1.4 Bible. Indianapolis: Wiley Publishing, Inc.

Monson-Haefel, R. 2003. J2EE(TM) Web Services: Addison-Wesley.

OASIS. 2009 [1]. OASIS Sponsors. [viitattu 26.2.2009] <http://www.oasis-open.org/about/index.php>

OASIS. 2009 [2]. OASIS Contributors. [viitattu 26.2.2009] <http://www.oasis-open.org/about/contributors.php>

OSOA. 2007. JAX-WS Services Integration. [viitattu 27.4.2009] <http://www.osoa.org/display/Main/JAX-WS+Services+Integration>

OSOA. 2008. Implementation Examples and Tools. [viitattu 27.4.2009] <http://www.osoa.org/display/Main/Implementation+Examples+and+Tools>

Pautasso, C., Zimmermann, O., Leymann, F. 2008 RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. Peking: WWW 2008 / Refereed Track: Web Engineering - Web Service Deployment.

Poutsma, A. Evans, R. Rabbo, A. T. 2007. Spring Web Services - Reference Documentation 1.5.6. [viitattu 25.4.2009] <http://static.springframework.org/spring-ws/sites/1.5/reference/pdf/spring-ws-reference.pdf>

Poutsma, A. 2009. [viitattu 26.4.2009] <http://blog.springsource.com/2009/03/08/rest-in-spring-3-mvc/>

Progress Software Corporation. 2008. Configuring and Deploying FUSE™ Services Framework Endpoints. [viitattu 28.4.2009] <http://fusesource.com/docs/framework/2.2/deploy_guide/deploy_guide.pdf>

Resende, L. Feng, R. 2007. Handling Heterogeneous Data Sources in a SOA Environment with Service Data Objects (SDO). SIGMOD'07, June 12–14, 2007, Beijing, China.

Richardson, L., Ruby, S. 2007. RESTful Web Services. Sebastopol: O'Reilly Media, Inc.

Rosen, M., Lublisnky, B., Smith, K. T., Balcer, M. J. 2008. Applied SOA: Service-Oriented Architecture and Design Strategies. Indianapolis: Wiley Publishing, Inc.

Sun Microsystems. Java EE Compatibility. [viitattu 22.1.2009] <http://java.sun.com/javaee/overview/compatibility.jsp>

Sun Microsystems. 2004. Java(TM) 2 Platform Standard Ed. 5.0. [viitattu 6.3.2009] <http://java.sun.com/j2se/1.5.0/docs/api/>

Taylor, I., Harrison, A. 2009. From P2P and Grids to Services on the Web: Evolving Distributed Communities. London: Springer-Verlag.

The Apache Software Foundation. 2008. Build your first Web Services with Tuscany. [viitattu 27.4.2009] <http://tuscany.apache.org/build-your-first-web-services-with-tuscany.html>

Tidwell, D. Snell, J., Kulchenko, P. 2001. Programming Web Services with SOAP. Sebastopol: O'Reilly & Associates, Inc.

Tilkov, S. 2007. Arjen Poutsma on Spring Web Services. [viitattu 26.4.2009] <http://www.infoq.com/articles/arjen-poutsma-spring-ws>

TIOBE. Programming Community Index for January 2009. [viitattu 22.1.2009] <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

World Wide Web Consortium. 2003. Web Services Description Language (WSDL) Version 1.2. [viitattu 20.2.2009] <http://www.w3.org/TR/wsdl12>

World Wide Web Consortium. 2004. Web Services Architecture. [viitattu 5.2.2009] <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

World Wide Web Consortium. 2007. Web Services Description Language (WSDL) Version 2.0. [viitattu 20.2.2009] <http://www.w3.org/TR/2007/REC-wsdl20-20070626>