Juha Riihola

Web-palvelut ja sen perusteknologiat

Tietojärjestelmätieteen
kandidaatintutkielma
5.4.2005

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Riihola, Juha Markus Olavi
Tietojärjestelmätieteen kandidaatintutkielma
Jyväskylä: Jyväskylän yliopisto, 2005.
40 s.

Yhteentoimivuus erilaisten järjestelmien välillä on ollut heterogeenisen ohjelmistoteollisuuden yksi keskeisiä ongelmia. Web-palvelut (Web Services) -teknologia pyrkii ratkaisemaan tätä ongelmaa tarjoamalla toimittajariippumattoman menetelmän, jolla eri ohjelmointikielillä toteutetut ja eri alustoilla olevat sovellukset kykenevät keskustelemaan keskenään.

XML toimii keskeisimpänä Web-palveluiden perusteknologiana. XML tarjoaa standardin, joustavan ja laajennettavan tavan kommunikointiin ja toimi avaintekijänä useiden eri tekniikoiden yhteistyössä. Itse Web-palvelu on erilaisten sovellusten yhteistoimintaan suunniteltu sovelluskomponentti, joka kommunikoi käyttämällä SOAP-protokollaa avoimien verkkoprotokollien avulla. Web-palvelu on kuvattu WSDL-kieltä käyttäen ja sen sanomarakenne on määritelty XML-skeema–kielellä. Web-palvelut voidaan julkaista julkisella tai yksityisellä UDDI-rekisterillä.

Web-palvelujen ympärille on kehitetty suuri joukko erilaisia käytäntöjä ja protokollia, jotka pitävät huolta palvelujen luotettavuudesta, turvallisuudesta, hallinnasta jne. Keskeisimmät protokollat ovat laajalti hyväksyttyjä ja käyttöönotettuja, mutta yhä suuri osa näistä protokollista on joko keskeneräisiä tai vasta julkaistuja. Web-palvelut ovat siis kohdanneet uusille teknologioille tyypillisesti myös paljon kritiikkiä. Kritiikki on kohdistunut pääasiassa tietoturvaan, luotettavuuteen, prosessin ja transaktion hallintaan ja suorituskykyyn.

AVAINSANAT: Web Services, Web-palvelut, XML, SOAP, WSDL, UDDI, SOA

SISÄLLYSLUETTELO

1 JOHDANTO

Elektroninen liiketoiminta on yhä merkittävämpi tekijä globaalille taloudelle. Kasvu on ollut huimaa varsinkin 1990-luvun alusta alkaen ja kehitykselle ei näy loppua. Keskeisessä roolissa sähköisessä kaupankäynnissä on yritysten välinen kaupankäynti, joten yritysten tietojärjestelmien välinen integrointi on ollut suuren haasteen edessä.

David S. Linthicum jakaa kirjassaan *Next Generation Application Integration* (2004) tietojärjestelmien integroinnin kahteen pääluokkaan: EAI (Enterprise Application Integration) eli yrityksen sisäinen tietojärjestelmien integrointi ja B2B (Business to Business) eli yritysten välinen tietojärjestelmien integrointi. Integroinnin lähestymistavat jaetaan niin ikään neljään pääluokkaan: informaatio-orientoitunut (Information Oriented), liiketoimintaorientoitunut (Business Process Oriented), palveluorientoitunut (Service Oriented) ja portaali-orientoitunut (Portal Oriented). Linthicumin (2004) mukaan selkeänä trendinä on ollut siirtyminen yksinkertaisesta ja edullisesta informaatio-orientoituneesta integroinnista palveluorientoituneeseen integrointiin.

Palveluorientoituneessa tietojärjestelmien integroinnissa (SOAI, Service Oriented Application Integration) tietojärjestelmät jakavat yhteisiä liiketoimintalogiikoita ja metodeja keskenään. Uusia sovelluksia muodostetaan koostamalla verkossa olevia palveluita. Lähestymistapa ei ole uusi, sillä eri hajautetut objektiteknologiat, kuten esimerkiksi DCOM ja CORBA, ovat olleet laajalti käytössä jo pitkään. Dan Gisolfin (2001) mukaan näiden tekniikoiden menestystä on kuitenkin rajoittanut niiden toimittajariippuvaisuus, varsinkin DCOM on täysin riippuvainen Windows-alustasta. CORBA on toimittajariippumaton tiettyyn pisteeseen saakka, mutta esimerkiksi turvallisuus ja transaktioiden hallinta vaatii Gisolfin (2001) mukaan toimittajariippuvaisia keinoja. Uusimpana, täysin toimittajariippumattomana, ratkaisuna tarjotaan Web-palveluita, joka käyttää tehokkaasti hyväkseen olemassa olevia internetin standardeja. Teknologia on

saanut monen suuren valmistajan, kuten Microsoftin, IBM:n, Sunin, HP:n ja BEA:n, vahvan tuen, joten kyse ei ole todennäköisesti kovinkaan lyhytaikaisesta muoti-ilmiöstä.

Tässä tutkielmassa kartoitetaan, mitkä ovat keskeisimmät Web-palvelujen sisältämät teknologiat. Teknologiat käydään läpi havainnollistaen niitä yksinkertaisilla kuvilla ja XML-muotoisilla esimerkeillä. Tutkielmassa käsitellään Web-palveluita myös kriittisestä näkökulmasta esitellen yleisimmät kirjallisuudesta löytyneet teknologiaa kohdanneet haasteet ja niille mahdollisesti esitetyt ratkaisut. Tutkielmassa pyritään antamaan yleiskuva myös siitä, mikä on Web-palvelujen kypsyys tutkielman kirjoitushetkellä vuoden 2005 alkupuolella.

2 XML – ARKKITEHTUURIN ESITYSTAPA

Keskeisin web-palveluiden perusteknologioista on XML, jota käytetään kauttaaltaan koko arkkitehtuurin perusesitystapana. XML tarjoaa standardin, joustavan ja laajennettavan tavan kommunikointiin, ja toimii avaintekijänä useiden eri tekniikoiden yhteistyössä. Boothin, Haasin, McCaben ym. (2004) mukaan Web-palveluille XML:n piirteistä tärkeimmät osa-alueet ovat itse XML:n syntaksi, XML-nimiavaruudet (XML Namespaces), XML Infosetit ja XML-skeemat (XML Schema). XML:n parhaita puolia on myös sen siirrettävyys. XML-protokollaa käyttämällä ohjelmistosuunnittelijat pystyivät helposti esittämään yrityksen sisäisten järjestelmien datan siirrettävässä muodossa. Kun siirrettävä data on tällä tavoin standardisoitu, on myös yritysten tietojärjestelmien välinen kommunikointi huomattavasti tehokkaampaa.

2.1 XML-nimiavaruudet

XML-nimiavaruudet (XML Namespaces) tarjoaa yksinkertaisen tavan XML-dokumenteissa esiintyvien elementtien ja attribuuttien määrittelyyn. Nimiavaruudet liitetään XML-dokumentteihin URI-viittauksilla (Uniform Recource Identifier). XML-nimiavaruus on kokoelma elementtien tyyppejä ja attribuutteja, jotka ovat identifioitu URI-viitteillä. Määriteltyjä nimiä kutsutaan sovelluksen sanastoksi (vocabulary). (Bray, Hollander & Layman, 1999)

Bray ym. (1999) mukaan nimiavaruuksien sisältämät nimet esiintyvät XML-dokumenteissa kahdessa perusmuodossa:

- Kvalifioituna nimenä (qualified name), joka koostuu kahdesta ”:”-merkillä erotetusta osasta: nimiavaruuden prefiksistä ja lokaaliosasta. Prefiksin kiinnittäminen haluttuun nimiavaruuteen suoritetaan xmlns-attribuutin avulla (esimerkiksi xmlns:ex=[nimiavaruuden nimi]). Kuvassa 1 olevassa esimerkissä valitaan nimiavaruus ”http://example.org/test/” ja sille prefiksi ex.

```
<ex:EXAMPLE xmlns:ex=”http://example.org/test/”>
<ex:CONTENT ex:ATTR=” myös attribuutilla on nimiavaruus”>
     CONTENT-elementin nimiavaruus määritelty ex-prefiksin
avulla
   </ex:CONTENT>
</ex:EXAMPLE>
```

Kuva 1. Kvalifioidun nimen käyttö

- Lokaalina nimenä (kvalifioimattomana) nimenä, joka koostuu vain nimen lokaaliosasta. Tässä tapauksessa nimi sidotaan oletusnimiavaruuteen, se perii vanhempiensa nimiavaruuden tai se ei kuulu mihinkään nimiavaruuteen. Oletusnimiavaruus määritellään nimiavaruusjulistuksen (namespace declaration) avulla, joka tehdään XML-dokumentin esiintymäosassa (ks. Kuva 2). Elementti CONTENT perii oletusnimiavaruuden. Attribuutti ATTR ei peri oletusnimiavaruutta johtuen sovituista perintäsäännöistä. Attribuuttien nimiavaruus määrätään aina prefiksin avulla.

```
<EX xmlns=”http://example.org/test/”>
   <CONTENT ATTR=”Attribuutilla ei oletuksena nimiavaruutta”>
     Elementin lapset perivät elementin oletusnimiavaruuden
   </CONTENT>
</EX>
```

Kuva 2. Oletusnimiavaruuden määritys ja käyttö

Nimiavaruuksien määrittelyssä on otettava huomioon, että xml-alkuiset nimet on varattu World Wide Web Consortiumin (W3C) omaan käyttöön (nimiavaruuden URI: http://www.w3c.org /XML/1998/namespace)

2.2 XML Information Set

XML Information Set (Infoset) määrittelee abstraktin datajoukon, joka sisältää osan XML-dokumentista saatavasta informaatiosta ja niiden suhteista toisiinsa.

Infoset ei ota kantaa dokumentissa olevaan XML-syntaksiin vaan paremminkin informaatioon, jota dokumentti sisältää. (Cowan & Tobin, 2004)

XML Information Set kertoo mm. seuraavat asiat:

- millaisia lapsia elementillä on
- mihin elementtiin tai nimiavaruuteen määrite kuuluu
- mikä määritteen tyyppi on.

2.3 XML-skeema

Termillä *skeema* (schema) tarkoitetaan informaation rakenteen kuvaamista. Norman Walshin (1999) mukaan XML:n kontekstissa skeemalla määritellään malli kokonaiselle luokalle XML-dokumentteja. Malli kuvaa sitä noudattavan validin XML-dokumentin sisällön mahdollisen järjestyksen. XML-skeema voidaan käsittää myös sopimuksena yhteisestä sanastosta tietyn sovelluksen yhteydessä.

XML-skeema on siis metakieli XML-dokumenttien rakenteen määrittelemiseen. Käytännössä XML-skeema on kokoelma XML-dokumentteja, yksittäinen XML-dokumentti, tai osa toista XML-dokumenttia, skeema käyttää siis XML:n omaa kielioppia. Skeema muodostuu säännöistä, jotka määrittävät halutunlaisen XML-dokumentin rakenteen, elementit, attribuutit, attribuuttien tietotyypit ja tarvittaessa myös attribuuttien arvoille sallitut arvojoukot. Kaikki XML-dokumentit, jotka täyttävät tietyn XML-skeeman määrittelemät validisuusehdot, kuuluvat kyseisen XML-skeeman määrittämään dokumenttiluokkaan. XML-skeemaa voidaan käyttää validioimaan luotu XML-dokumentti, dokumentti voidaan myös muodostaa dynaamisesti XML-skeeman perusteella.

Hieman XML-skeemaa vastaava teknologia on SGML:n (Standard Generalized Markup Language) yhteydessä yleisesti käytetty DTD (Document Type Definition), josta on määritelty versiot myös XML:lle ja XML-skeemalle. DTD sisältää

kuitenkin rajoitteita, jotka tekevät siitä helposti isokokoisen ja vaikeasti hallittavan. Lisäksi DTD:ssä käytetään myös eri kielioppia kuin XML:ssä ja se ei tue nimiavaruuksien käyttöä. Norman Walshin (1999) mukaan XML-skeema on huomattavasti parempi ja ilmaisuvoimaisempi kuvaustapa XML-dokumenteille.

On olemassa myös muitakin XML:ään pohjautuvia skeemoja, kuten XML Data Reduced ja Microsoft Biztalk.

3 TEKNOLOGIOIDEN KUVAUS

3.1 Yleistä

Aaron Skonnard (2002) määrittelee web-palvelun sovelluskomponentiksi, joka

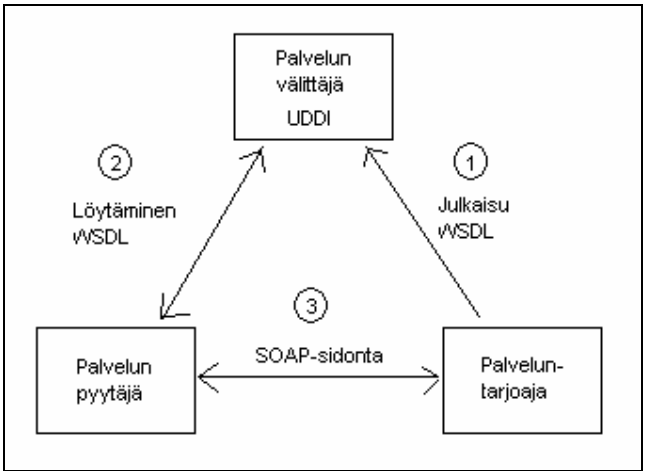
- Kommunikoi XML-viesteillä käyttäen SOAP:ia
- Kommunikoi avoimia verkkoprotokollia käyttäen
- Kuvaa viestit XML-skeemaa käyttäen
- On kuvattu WSDL-kielellä
- Voidaan löytää UDDI-rekistereitä käyttäen.

World Wide Web Consortiumin (W3C) uusin *Web Services Architecture* -dokumentaatio (WSA) (Booth, Haas, McCabe ym. 2004) mukailee Skonnardin määritelmää:

> 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 conveyded using HTTP with an XML serialization in conjunction with other Web-related standards.

Uusimmassa WSA-versiossa puhutaan palvelujen teknisen kuvauksen rinnalla myös web-palvelujen semanttisesta kuvauksesta (Booth ym. 2004). Rajaan kuitenkin selkeyden vuoksi komponenttien käyttäytymisen merkitykseen liittyvän keskustelun pois tästä tutkielmasta.

Hyvin yleinen tapa kuvata web-palveluita on kuvata se kolmen roolin yhteistoimintana (kuva 3): palvelun pyytäjä (Service Requester), palvelun tarjoaja (service provider) ja palvelun välittäminen (service broker), eli palvelurekisteri (service register). Arkkitehtuuria voidaan katsoa joko liiketoiminnallisesta tai arkkitehtuurisesta näkökulmasta. (Kreger, 2001)



Kuva 3. Yksinkertaistettu Web-palveluiden arkkitehtuuri Kregeriä (2001) mukaille

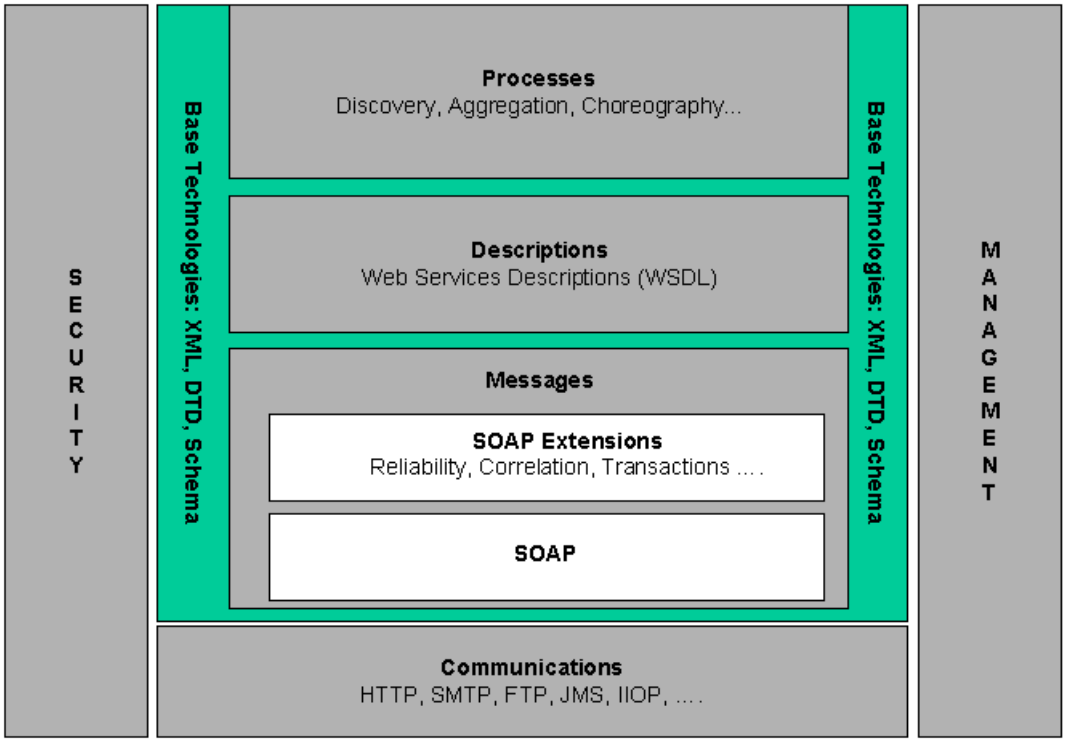
Palveluntarjoaja on liiketoiminnallisesta näkökulmasta palvelun omistava yritys. Arkkitehtuurisesta näkökulmasta palveluntarjoaja on alusta (platform), joka ylläpitää web-palvelua. W3C:n dokumentoinnissa (Booth ym. 2004) korostetaan, että Web-palvelu itsessään on vain abstrakti kohde (entity), ja se on toteutettava konkreettisesti ns. agentilla. Käytännössä tämä tarkoittaa sitä, että vaikka itse Web-palvelu säilyy muuttumattomana, voidaan sen toteuttava agentti korvata toisella ohjelmointikielellä toteutetulla optimaalisemmalla agentilla, joka toteuttaa täysin samat toiminnallisuudet.

Palveluntarjoaja voi julkaista kehittämänsä Web-palvelun UDDI-rekisterillä (kuva 3, (1)). Käytännössä tämä tarkoittaa palvelukuvauksen, eli WSDL-dokumentin, toimittamista rekisteriin. Lisäksi palveluntarjoaja toimittaa rekisteriin tiettyjä metatietoja kuten esimerkiksi organisaation yhteystiedot ja palvelun toimialan.

Palvelun pyytäjä on liiketoiminnallisesta näkökulmasta jokin liiketoiminta, joka tarvitsee jotain palvelua. Arkkitehtuurisesta näkökulmasta palvelun pyytäjä on sovellus, joka etsii ja käyttää web-palvelun palveluja. Palvelun pyytäjä voi hakea tarpeisiinsa sopivaa Web-palvelua UDDI-rekisteriltä, jossa julkaistut palvelut ovat luokiteltu mm. liiketoiminnan ja toimittajan perusteella (kuva 3, (2)).

Löydettyään sopivan Web-palvelun, palvelun pyytäjä siirtää rekisteriltä palvelukuvauksen (WSDL-dokumentin) itselleen, joka sisältää tiedot palvelun rajapinnasta ja käytöstä SOAP-sidonnan avulla (kuva 3, (3)).

Palvelurekisteri on rekisteri, josta voidaan hakea palveluntarjoajien julkaisemia Web-palveluiden kuvauksia (WSD, Web Service Description). Web-palvelujen yhteydessä käytetään yleisesti UDDI-rekisteriä (Universal Description, Discovery and Integration). Rekistereitä voidaan käyttää joko suunnittelun tai ajon aikana. Palvelurekisterin olemassaolo ei ole välttämättömyys, vaan kuvaukset voidaan toimittaa yhteistyökumppaneille myös esimerkiksi sähköpostilla, FTP:llä jne. (Kreger, 2001)



Kuva 4. Web-palveluiden arkkitehtuuripino (Booth ym. 2004)

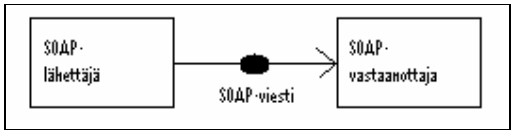
Kuvassa 4 kuvataan W3C:n (Booth ym. 2004) määrittelemä Web-palveluiden arkkitehtuuripino. Kaiken perustana toimii kommunikaatio (Communications), jota käsitellään luvussa 3.2. Viestien käsittelyä (Messages) käsitellään luvussa

3.3. Web-palveluiden kuvauksia (Descriptions) käsitellään luvussa 3.4. W3C:n arkkitehtuuripinossa esitellään ylimpänä osa-alueena prosessit, joista esittelen Web-palveluiden paikallistamisen (Discovery). Arkkitehtuuripinon XML-kieleen liittyvät perusteknologiat (Base Technologies) käsiteltiin luvussa 2. Viimeisimpänä tässä tutkielmassa käsitellään tietoturvaa (Security) ja hallintaa (Management) luvussa 4.

3.2 Kommunikointi

Web-palvelujen perustana toimii tietoverkko, joten palvelujen tulee olla saavutettavissa verkon kautta. Varsinkin julkisessa verkossa tarjolla olevat web-palvelut käyttävät yleisimpiä verkkoprotokollia, erityisen vakaan aseman on saavuttanut HTTP-protokolla. Web-palvelut tukevat myös muita internet-protokollia kuten esimerkiksi SMTP ja FTP. Yritysten sisäisissä verkoissa toimivat web-palvelut taas voivat käyttää harvinaisempia ja samalla luotettavampia protokollia, kuten esimerkiksi CORBA, MSSeries jne. (Kreger, 2001)

3.3 Viestintä — SOAP



Kuva 5. Yksinkertainen SOAP-viesti Skonnardia (2003a) mukaillen

Web-palveluiden viestintä on toteutettu SOAP-protokollan avulla. SOAP toimii yleisesti XML-muotoisen viestinnän perusstandardina (Kreger, 2001)

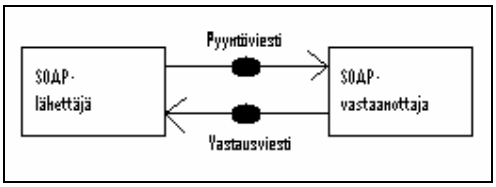
SOAP-protokolla määrittelee tavan jolla siirretään XML-muotoisia viestejä sovellusten välillä tietoverkon kautta. Tämä tapahtuu XML-pohjaisessa viestikehyksessä, jonka keskeisimmät piirteet ovat laajennettavuus, käytettävyys lukuisten verkkoprotokollien päällä, sekä riippumattomuus eri ohjelmointikielistä. (Skonnard, 2003a)

SOAP ei myöskään määrittele uutta verkkoprotokollaa vaan toimii jo olemassa olevien protokollien päällä. (Curbera ym. 2002)

SOAP-protokolla on hyvin yksinkertainen protokolla: se ei esimerkiksi sisällä tiettyjä hajautetuille tekniikoille ominaisia piirteitä, kuten esimerkiksi turvallisuutta (security) ja luotettavuutta (reliability). SOAP:in kommunikointikehys sitä vastoin määritelty niin, että tällaiset ominaisuudet voidaan lisätä siihen laajennuksina (SOAP Extensions). Suuret ohjelmistotalot, kuten Microsoft ja IBM, ovat kehitelleet yleistä laajennusosaa, joka sisältäisi keskeisimmät ohjelmistokehittäjien tarvitsemat laajennukset. (Skonnard, 2003a)

Menetelmää, jonka avulla SOAP-sanoma välitetään yhdeltä solmulta toiselle, kutsutaan SOAP-sidonnaksi (SOAP binding). Sidonta on toteutettu eri verkkoprotokollien avulla. SOAP:ia voidaan käyttää minkä tahansa verkkoprotokollan päällä, mukaan luettuna sovelluskerroksen protokollat (application layer protocol): TCP, HTTP, SMTP, MSMQ jne. (Skonnard, 2003a; Mitra, 2003). Jokainen eri protokolla-sidonta (protocol binding) tarjoaa oman tapansa sarjallistaa (serialize) SOAP-viesti, eli muuttaa viesti tietovirraksi, välittää se seuraavalle solmulle ja purkaa tietovirta takaisin ”luettavan muotoon”. Verkkoprotokollat tarjoavat eri ominaisuuksia SOAP-sidonnan käyttöön. Esimerkiksi tilanteessa, jossa työskennellään hitaan modeemiyhteyden kanssa, voidaan valita pakkauksen mahdollistava protokolla, toisessa tapauksessa taas voidaan valita salauksen mahdollistava protokolla (esim. http & SSL). Protokollan ominaisuuksiin voi kuulua myös esimerkiksi vastauksen yhdistäminen oikeaan pyyntöön (correlate a request with a response) ja luotettavuuden varmistaminen (reliable delivery channel). Näitä ominaisuuksia voidaan tarvittaessa toteuttaa myös SOAP:in otsikkokentän laajennuksina. Varsinkin jos viestinnässä käytetään useita välisolmuja (intermediates) useiden eri verkkoprotokollien kanssa, voi olla parempi käyttää SOAP-kirjekuoren laajennuksia protokollan ominaisuuksien asemesta. (Mitra, 2003)

SOAP:ia ei myöskään pidä sekoittaa hajautettujen objektien yhteydessä yleisesti käytettyyn RPC-tekniikkaan. RPC (Remote Procedure Call) tarkoittaa Skonnardin (2003a) mukaan käsitteenä ohjelmointimallia, joka antaa ohjelmistokehittäjille mahdollisuuden työskennellä metodikutsujen kanssa. RPC:n toteuttaminen SOAP:illa on kuitenkin mahdollista. SOAP on paremminkin perinteisen viestintätavan kaltainen (kuten MSMQ), sillä protokolla määrittää tavan toteuttaa yksittäisiä ja yksisuuntaisia viestejä. Kaksisuuntainen viestintä saadaan aikaiseksi yhdistämällä useita viestejä samaan keskusteluun.

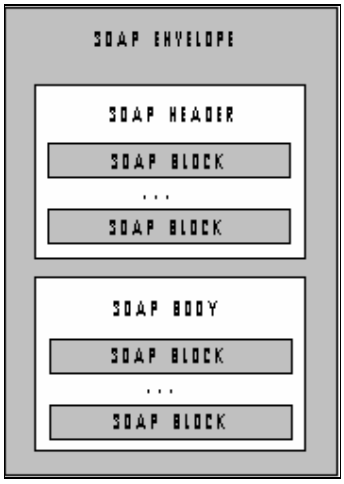


Kuva 6. Pyyntö/vastaus–viestintää SOAP:issa Skonnardia (2003a) mukaillen

SOAP antaa mahdollisuuden luoda vapaasti erilaisia viestintätapoja (Message Exchange Pattern MEP), joista pyyntö/vastaus (request/response) on vain yksi. Muita esimerkkejä ovat solicit/response (pyyntö/vastaus-tavan vastakohta), tiedonannot (notification) ja pitkäkestoiset peer-to-peer–keskustelut. (Skonnard, 2003a)

3.3.1 SOAP-viestikehys

SOAP-viestikehyksessä (SOAP Messaging Framework) määritellään XML-elementit, joiden avulla viestintä eri järjestelmien välillä tapahtuu. Itse viestikehys on määritelty XML-Skeema–kielellä, viittaus käytettyyn viestikehykseen (ks. liite 1: SOAP 1.2 XML Schema Definition) on löydyttävä jokaisen SOAP-viestin alusta. Viittaus viestikehykseen voidaan käsittää myös nimiavaruuden valintana. Jokainen XML-dokumentti, joka toteuttaa SOAP:in XML-skeeman, on validi SOAP-viesti. (Skonnard, 2003a)



Kuva 7. Yksinkertaistettu SOAP-kehys Elwiniä ym. (2002) mukaillen

Kirjekuori (Envelope) on SOAP-viestin peruselementti, josta sovellusten on helppo huomata viestin olevan juuri SOAP-viesti.

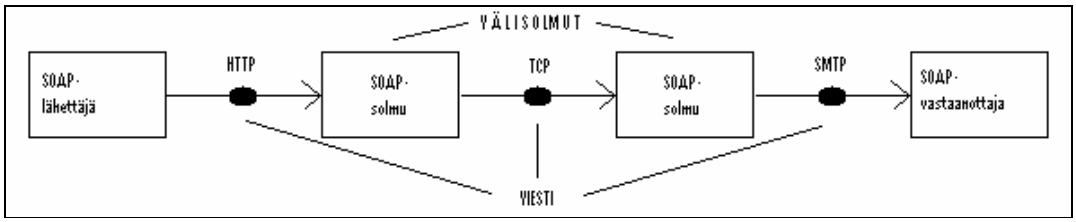
```
<soap:Envelope
   xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”
   <soap:Header> <!—optional -->
      <!-- Header blocks go here… -->
   </soap:Header>
   <soap:Body>
      <!--payload or Fault element goes here… -->
   </soap:Body>
</soap:Envelope>
```

Kuva 8. Yksinkertainen SOAP-viesti

Kirjekuori–elementti sisältää vaihtoehtoisen otsikko–elementin (header), jonka jälkeen tulee pakollinen runko–elementti (body). Runko sisältää viestin ns. hyötykuorman. Runko-elementti on geneerinen, se voi sisältää vapaasti elementtejä mistä tahansa nimiavaruudesta (Skonnard, 2003a)

SOAP:in laajennettavuus on toteutettu otsikko-elementin avulla. Elementti tarjoaa keinon tallentaa meta-informaatiota, jota ei ole tarkoitettu kommunikoiville sovelluksille. Tällainen ”kontrolli-informaatio” sisältää esimerkiksi viestin prosessointiin liittyvää tietoa.

Otsikko-osa on suunniteltu ennakoiden monenlaista SOAP:in käyttöä. Esimerkiksi, SOAP-viestin lähettäjän (sender) ja lopullisen vastaanottajien (ultimate receiver) välillä voi olla välisolmuja (intermediates), jotka jokainen käsittelevät viestiä hieman eri tavalla. (Mitra, 2003).



Kuva 9. SOAP-välisolmut Skonnardia (2003a) mukaillen

Välisolmu toimii sekä vastaanottajana että lähettäjänä. Välisolmut mahdollistavat mielenkiintoisien verkkoarkkitehtuurien suunnittelun. Esimerkiksi kuvassa 9 on esitetty tilanne, jossa käytetään kolmea eri verkkoprotokollaa ja kahta välisolmua. Välisolmujen toimintaan voidaan vaikuttaa SOAP-viestin otsikkoelementin sisällöllä. (Skonnard, 2003a)

Lähettäjäsolmu, välisolmut ja vastaanottajasolmu ovat solmujen *rooleja*. Solmua ei välttämättä ole sidottu kiinteästi mihinkään rooliin, vaan se tunnistaa roolinsa saapuvan viestin perusteella. SOAP-viestin otsikon sisältämä data voidaan osoittaa eri rooleissa toimiville solmuille erikseen. Kun solmu on tunnistanut roolinsa, on sen suoritettava kaikki pakolliset otsikko-osan elementit. Pakolliset otsikko-osat tunnistetaan elementtiin liitetyn attribuutin *mustUnderstand* arvolla 1. Solmu voi suorittaa myös vapaaehtoiset otsikko-osat (mustUnderstand = 0).

SOAP 1.2:ssa määritellään kolme eri roolia:

- next
    - Jokaisen SOAP välisolmun ja lopullisen vastaanottajan **on toimittava** tässä roolissa ja **voi lisäksi toimia** muissa rooleissa
- none
    - Mikään SOAP-solmu **ei toimi** tässä roolissa

- ultimateReceiver
    - Jotta solmu tunnistetaan lopulliseksi vastaanottajaksi, on solmun toimittava tässä roolissa. Välisolmut eivät saa toimia tässä roolissa.

Lisäksi solmu voi toimia myös vapaasti muokattavassa (customed) roolissa (Skonnard, 2003a)

3.3.2 RPC-käytäntö

SOAP määrittelee dokumenttityylisen viestinnän lisäksi käytännön myös RPC-kutsujen (Remote Procedure Call) hyödyntämiseen. RPC:llä tarkoitetaan Skonnardin (2003a) mukaan ohjelmointimallia, jolla annetaan mahdollisuus työskennellä metodikutsujen kanssa verkon yli. SOAP-standardissa on määritelty standardi tapa muuttaa RPC-kutsut SOAP-viesteiksi, joka mahdollistaa reaaliaikaisen metodien tulkkaamisen SOAP-viesteiksi ja päinvastoin.

Metodin esittely muokataan yksinkertaisiksi pyyntö/vastaus-rakenteeksi, joka voidaan sitten esittää XML:nä (ks. Kuva 12). Käytännössä metodin esittely muutetaan tietueeksi, joka sisältää muuttujan jokaiselle in tai in/out–parametrille. Metodin vastaus muotoillaan myös tietueeksi (ks. Kuva 11).

```
double add(ref doublex, double y
```

Kuva 10. Alkuperäinen C#-kielinen metodikutsu

```
struct Add {
   double x;
   double y;
}

--------------------------------

struct AddResponse {
   double result;
   double x;
}
```

Kuva 11. Metodikutsu käännettynä pyyntö- ja vastaustietueiksi

```
<add>
   <x>33</x>
   <y>44</y>
</add>

-------------------------------

<addResponse>
   <result>77</result>
   <x>33</x>
</addResponse>
```

Kuva 12. SOAP RPC pyyntö- ja vastausviestit

3.3.3 SOAP-viestin koodaus

SOAP-viestien koodauksella (SOAP encoding rules) tarkoitetaan tapaa muuntaa sovelluksessa sijaitseva data XML-muotoiseksi dataksi sekä päinvastoin.

Frank Cohenin (2003) mukaan seuraavat kolme koodaustapaa ovat tulleet suosituimmiksi:

- RPC-koodaus, joka on määritelty SOAP 1.1 määrittelyssä. Tämä on yksinkertaisin koodaustapa, jossa ei tarvitse välittää ulkoisista WSDL- tai XML-skeema -määrittelyistä. Metodien koodaus tehdään suoraan SOAP 1.1 määrittelyn mukaan.

- RPC-kirjaimellinen koodaus (SOAP RPC-literal), joka tarkoittaa RPC-kutsua siten, että välitettävä data on jo valmiiksi XML-muodossa. Käytännössä kyseessä on siis RPC-viesti yhdellä parametrilla.

- Dokumenttityyppinen (document-style) koodaus, joka tunnetaan myös viestityylisenä (message-style) tai document-literal–tyyppisenä viestintänä. Tässä tyylissä palvelimelle lähetetään kokonainen SOAP viesti vaatimatta vastausta lähetykseen. Cohenin (2003) mukaan tämä tapa on vaikein, sillä sovelluskehittäjän vastuulle jää erittäin paljon rutiineja. Skon-

nardin (2003) mukaan tämä tapa on kuitenkin yleisimmin käytössä useimmissa kehitysympäristöissä ja on ylivoimainen muihin koodaustapoihin nähden.

Lisäksi on kehitetty käytäntöjä esimerkiksi binäärisen data koodaukseen, mutta Cohenin (2003) mukaan nämä tavat eivät ole vielä yleistyneet.

3.3.4 SOAP HTTP-sidonta

Heather Kregin (2001) mukaan HTTP on sen yleisyyden vuoksi ns. de facto –verkkoprotokolla SOAP-protokollan yhteydessä. HTTP:n etuihin kuuluu esimerkiksi kätevä tapa yhdistää saapunut vastaus lähetettyyn pyyntöön: SOAP:ia HTTP:n kautta käyttävä sovellus voi päätellä suoraan HTTP-vastausviestin otsikko-osasta (body), mille pyyntöviestille vastaus oli tarkoitettu. (Mitra, 2003)

HTTP mahdollistaa ns. web-metodien (Web-method) käytön, rajoittaen sen GET- tai POST-metodeihin. HTTP tarjoaa kaksi viestintätapaa (MEP, Message Exchange Pattern) (Mitra, 2003):

- SOAP pyyntö-vastaus-viestitapa (SOAP request-response message exchange pattern), eli HTTP:n post-metodin käyttö SOAP viestien siirtämiseen HTTP-pyyntöjen ja –vastausten rungoissa (body). Tätä käytetään varsinkin RPC:n mallintamiseen (ks. Kuvat 13, 14 & 15).

- SOAP-vastaus-viestitapa (SOAP response message exchange pattern), eli HTTP:n GET-metodin käyttö HTTP-pyynnön tekemiseen ja SOAP viestin palauttamiseen HTTP-vastauksen rungossa (ks. Kuvat 16 & 17).

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/transaction"
           env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true" >5</t:transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
     env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
          xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
    <o:creditCard xmlns:o="http://mycompany.example.com/financial">
     <n:name xmlns:n="http://mycompany.example.com/employees">
           Åke Jógvan Øyvind
     </n:name>
     <o:number>123456789099999</o:number>
     <o:expiration>2005-02</o:expiration>
    </o:creditCard>
   </m:chargeReservation>
  </env:Body>
</env:Envelope>
```

Kuva 13. RPC-pyyntöviesti HTTP POST -pyynnössä

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
       ...
       ...
 </env:Header>
 <env:Body>
       ...
       ...
 </env:Body>
</env:Envelope>
```

Kuva 14. Onnistuneen tapauksen RPC-vastausviesti HTTP POST – vastauksessa

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
     <env:Code>
       <env:Value>env:Sender</env:Value>
       <env:Subcode>
        <env:Value>rpc:BadArguments</env:Value>
       </env:Subcode>
     </env:Code>
     <env:Reason>
      <env:Text xml:lang="en-US">Processing error</env:Text>
      <env:Text xml:lang="cs">Chyba zpracování</env:Text>
     </env:Reason>
     <env:Detail>
      <e:myFaultDetails
        xmlns:e="http://travelcompany.example.org/faults" >
        <e:message>Name does not match card number</e:message>
        <e:errorcode>999</e:errorcode>
      </e:myFaultDetails>
     </env:Detail>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

Kuva 15. Epäonnistuneen tapauksen RPC-vastausviesti HTTP POST – vastauksessa

```
GET /travelcompany.example.org/reservations?code=FT35ZBQ http/1.1
Host: travelcompany.eample.org
Accept: text/html;q=0.5, application/soap+xml
```

Kuva 16. HTTP GET –pyyntö

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-30T16:25:00.000-05:00</m:dateAndTime>
  </m:reservation>
 </env:Header>
 <env:Body>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:x="http://travelcompany.example.org/vocab#"
     env:encodingStyle="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <x:ReservationRequest
 rdf:about="http://travelcompany.example.org/reservations?code=FT35ZBQ">
      <x:passenger>Åke Jógvan Øyvind</x:passenger>
      <x:outbound>
       <x:TravelRequest>
        <x:to>LAX</x:to>
        <x:from>LGA</x:from>
        <x:date>2001-12-14</x:date>
       </x:TravelRequest>
      </x:outbound>
      <x:return>
       <x:TravelRequest>
        <x:to>JFK</x:to>
        <x:from>LAX</x:from>
        <x:date>2001-12-20</x:date>
       </x:TravelRequest>
      </x:return>
    </x:ReservationRequest>
  </rdf:RDF>
 </env:Body>
</env:Envelope>
```

Kuva 17. SOAP-viesti palautettuna HTTP GET pyynnön vastausviestissä

3.4 Palvelukuvaukset — WSDL

SOAP-protokolla tarjoaa Web-palveluille tavan XML-pohjaiseen kommunikointiin, mutta se ei kuvaa sitä, millaisilla viesteillä kommunikointi tietyn komponentin kanssa tapahtuu. WSDL on XML-muotoinen kuvauskieli, joka kuvaa Web-palvelut kokoelmina kommunikoinnin päätepisteitä (collections of communication end points), jotka voivat vaihtaa tietynmuotoisia viestejä. Toisin sanoen, WSDL-dokumentti kuvaa Web-palvelun rajapinnan ja tarjoaa sen käyttäjälle tavan komponenttiin liittymiseen (Curbera ym. 2002).

WSDL tarjoaa määrämuotoisen kuvauksen asiakas–palvelin-vuorovaikutuksesta. Kehittäjät voivat käyttää WSDL:ää luodessaan asiakas-koodia, joka vastaa palvelimen vaatimuksiin. WSDL:ää voidaan käyttää joko staattisessa mielessä, jolloin sen tarjoama tieto kovakoodataan ohjelmakoodiin, tai sitten dynaamisten välipalvelimien (proxy) kanssa, jotka generoivat palvelinkutsuja tosiaikaisesti. (Curbera ym. 2002)

WSDL on itse asiassa XML-muotoinen dokumentti, jonka juurielementissä on viittaus WSDL 1.0:n XML-skeema muotoiseen nimiavaruuteen. (Skonnard, 2003b)

Aaron Skonnardin (2003b) mukaan Microsoftin .NET ympäristön mukana tulee komentopohjainen wsdl.exe-sovellus, joka luo luokkia WSDL-kuvauksesta. Sovellus osaa luoda kuvauksesta sekä asiakas-, että palvelinpään luokkia. Apachen valmistama vastaava sovellus on olemassa myös Java-luokkien luomiseksi. Luokat, jotka ovat luotu käyttäen samaa WSDL-kuvausta, kykenevät kommunikoimaan keskenään riippumatta käytetystä ohjelmointikielestä.

Täydellinen WSDL muotoinen palvelukuvaus (A Complete WSDL Definition) sisältää kahdenlaista informaatiota: sovellustason palvelukuvaus (application-level service description) eli abstraktin rajapintakuvauksen ja verkkoprotokollaspesifisen yksityiskohdat (specific protocol-dependent details), eli konkreettisen sidontainformaation (Concrete Binding Information). Tämän jaon myötä WDSL erottaa komponentin yleisen toiminnallisuuden erilaisista toteutuksista (Curbera ym. 2002). Skonnardin (2003b) mukaan täydellinen kuvaus sisältää kaiken tarvittavan informaation kuvatun Web-palvelun käyttöönottamiseen.

Tässä tutkielmassa tarkastelen WSDL:n vanhempaa versiota 1.0. WSDL:stä on ilmestynyt uusi 2.0, mutta tällä hetkellä versio on edelleen luonnosasteella.

3.4.1 Abstrakti kuvaus

Web-palvelun abstraktin kuvauksen elementtejä ovat **types, message ja portType**.

Types-elementti sisältää XML Schema -muotoiset tyypinmäärittelyt. (Skonnard 2003b)

```
<types>
  <xs:schema
    targetNamespace="http://example.org/math/types/"
    xmlns="http://example.org/math/types/"
  >
   <xs:complexType name="MathInput">
     <xs:sequence>
    <xs:element name="x" type="xs:double"/>
    <xs:element name="y" type="xs:double"/>
     </xs:sequence>
   </xs:complexType>
   <xs:complexType name="MathOutput">
     <xs:sequence>
       <xs:element name="result" type="xs:double"/>
     </xs:sequence>
   </xs:complexType>
   <xs:element name="Add" type="MathInput"/>
   <xs:element name="AddResponse" type="MathOutput"/>
   <xs:element name="Subtract" type="MathInput"/>
   <xs:element name="SubtractResponse" type="MathOutput"/>
   <xs:element name="Multiply" type="MathInput"/>
   <xs:element name="MultiplyResponse" type="MathOutput"/>
   <xs:element name="Divide" type="MathInput"/>
   <xs:element name="DivideResponse" type="MathOutput"/>
  </xs:schema>
</types>
```

Kuva 18. Esimerkki types-elementin sisällöstä

Message-elementti määrittelee abstraktin viestin, joka voi toimia input- tai output-viestinä tietyssä operaatiossa. Message-elementti sisältää yhden tai useamman part-elementin, joista kustakin on viittaukset element:iin (SOAP:in dokumenttityyppi) tai type:en (SOAP:in RPC-kutsu). (Skonnard, 2003b)

Jos ollaan määrittelemässä RPC-tyylistä palvelua, message:n part-elementit edustavat RPC-metodin parametreja. Dokumenttityylisessä palvelussa, part-elementit viittaavat XML-elementteihin, jotka ovat sijoitettu runko-osaan (body)

```
<message name="AddMessage">
   <part name="parameter" element="ns:Add"/>
</message>
<message name="AddResponseMessage">
   <part name="parameter" element="ns:AddResponse"/>
</message>
<message name="SubtractMessage">
   <part name="parameter" element="ns:Subtract"/>
</message>
<message name="SubtractResponseMessage">
   <part name="parameter" element="ns:SubtractResponse"/>
</message>
```

Kuva 19. Esimerkki messages-elementtien käytöstä

PortType-elementti määrittelee ryhmän operaatioita, joita kutsutaan myös rajapinnoiksi. Jokainen elementti sisältää yhdistelmiä input- ja output-elementeistä, mukana voi olla myös fault-elementti. Kustakin elementistä on viittaus tiettyyn message-elementtiin. PortType-elementin sisältämien elementtien sisäinen järjestys määrittelee kyseisen operaation viestintätavan (MEP, Message Exchange Pattern). (Skonnard, 2003b)

```
<portType name="MathInterface">
   <operation name="Add">
      <input message="y:AddMessage"/>
      <output message="y:AddResponseMessage"/>
   </operation>
   <operation name="Subtract">
      <input message="y:SubtractMessage"/>
      <output message="y:SubtractResponseMessage"/>
   </operation>
   <operation name="Multiply">
      <input message="y:MultiplyMessage"/>
      <output message="y:MultiplyResponseMessage"/>
   </operation>
   <operation name="Divide">
      <input message="y:DivideMessage"/>
      <output message="y:DivideResponseMessage"/>
   </operation>
</portType>
```

Kuva 20. Esimerkki portType-elementistä

3.4.2 Konkreettinen sidontainformaatio

Konkreettisen sidontainformaation elementtejä ovat **binding** ja **services**

Binding-elementti kuvaa konkreettiset yksityiskohdat tietyn portType:n käytöstä tietyllä verkkoprotokollalla. Binding-elementin attribuutti type määritte-

lee, mitä portType-elementtiä ollaan käsittelemässä. Binding-elementti sisältää operation-elementin jokaiselle operaatiolle, mitä tietyssä portType-elementissä on määritelty. Binding-elementin alla olevan soap:binding-elementti määrittelee, että kyseessä on SOAP 1.1-sidonta. Elementin style-attribuutti määrittelee, käytettävän palvelutyypin (RPC tai Document-type) ja transport-attribuutti määrittelee käytetyn verkkoprotokollan. Käytettävä koodaustyyli (literal tai encoded) määritellään soap:body–elementin use-attribuuttin avulla. (Skonnard, 2003b)

```
    <binding name="MathSoapHttpBinding" type="y:MathInterface">
     <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
     <operation name="Add">
        <soap:operation
                  soapAction="http://example.org/math/#Add"/>
        <input>
           <soap:body use="literal"/>
        </input>
        <output>
           <soap:body use="literal"/>
        </output>
     </operation>
          ...
  </binding>
```

Kuva 21. Esimerkki binding-elementistä

Service-elementti määrittelee kokoelman portteja (päätepisteistä / endpoints), jotka ilmentävät yksityiskohtaisen sidonnan. Käytännössä tämä tarkoittaa esimerkiksi URI:a, missä palvelu on käytettävissä. Service-elementti pitää sitoa tiettyyn binding-elementtiin.

```
  <service name="MathService">
      <port name="MathEndpoint" binding="y:MathSoapHttpBinding">
        <soap:address
            location="http://localhost/math/math.asmx"/>
      </port>
  </service>
```

Kuva 22. Esimerkki service-elementistä

3.5 Paikallistaminen — UDDI

UDDI:a (Universal Description, Discovery and Integration) voi kuvata Web-palveluiden puhelinluetteloksi, sillä se tarjoaa keskitetyn ja systemaattisen ta-

van löytää web-palveluja verkosta. UDDI-rekisterin rajapinta on toteutettu SOAP-standardin mukaisesti. (Curbera ym. 2002)

Skonnardin (2002) mukaan UDDI mahdollistaa sovelluskehittäjille Web-palveluiden hakemisen niin teknisten yksityiskohtien, kuten myös liiketoiminta-orientoituneen tiedon ja luokittelun perusteella. Esimerkiksi voidaan löytää kaikki maksuttomat Web-palvelut, jotka liittyvät pörssikurssi-informaatioon, ja käyttävät verkkoprotokollana HTTP:tä.

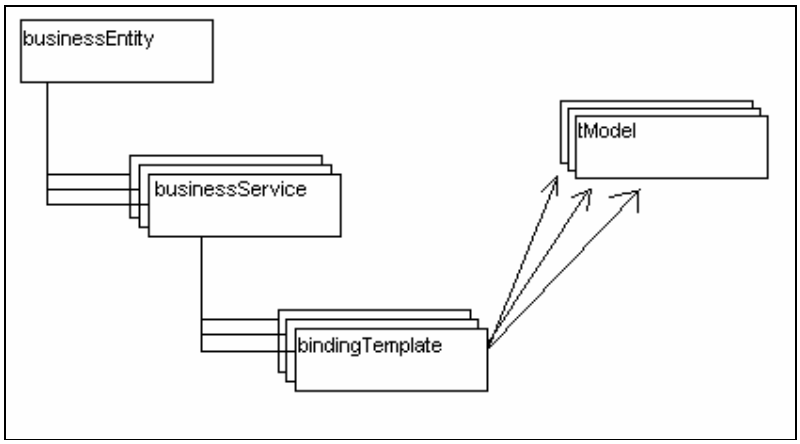
Karsten Januszewskin (2002) mukaan ilmeisesti tärkein UDDI:n ominaisuus on kategorisointi. Mahdollisuus liittää metadataa UDDI:in rekisteröityihin palveluihin ja sitten ajaa kyselyjä tähän metadataan perustuen on hänen mielestä ehdottomasti UDDI:n keskeisin merkitys sekä suunnittelun, että ajon aikana. Toisaalta Skonnard toteaa artikkelissaan (2002), että tuskin kukaan ohjelmistokehittäjä tai organisaatio antaa pienen koodin pätkän valita bisnespartnereita dynaamisesti.

Virallisen OASIS:en julkaiseman määritelmän (Bellwood, 2002) mukaan UDDI:n muodostavat joukko web-perusteisia rekistereitä, jotka sisältävät informaatiota palveluista ja niiden teknisistä rajapinnoista. Rekistereitä voidaan käyttää joko tiedon hakemiseen tai tiedon päivitykseen, käyttö on ilmaista. Web-palveluista rekisteröidään tietoa, joka antaa vastauksia yksinkertaisiin kysymyksiin: kuka, mitä, missä ja milloin:

- **Kuka:** Yksinkertainen informaatio palveluntarjoajasta ja palvelusta, kuten nimi ja yhteystiedot
- **Mitä:** Kategorisoitu informaatio palveluista, sisältäen *toimialan* ja *palvelun muiden ominaisuuksien* perusteella tapahtuva luokittelu.
- **Missä:** URL, sähköpostiosoite tms. mitä kautta palvelu on saatavissa

- **Kuinka:** Viitteitä rajapintakuvauksiin (WSDL-dokumentteihin) ja muihin kyseisen palvelun teknisiin ominaisuuksiin. Näitä viitteitä kutsutaan UDDI:n dokumentaatiossa tModelel:ksi. (Bellwood, 2002)

UDDI 1.0 määrityksen mukaan rekisteri sisältää neljä eri elementtityyppiä, jotka vastaavat edellä esitettyä jakoa: businessEntity, businessService, bindingTemplate ja tModel. (Kreger, 2002)



Kuva 23. UDDI:n perusdatarakenne

businessEntity-elementti tarjoaa informaatiota yrityksestä ja tarjoamista palveluista sekä niiden tyypeistä.

businessService-elementit kuvaavat businessEntity:n tarjoamat palvelut tai liiketoimintaprosessit. Kutakin businessEntity:n alle julkaistua Web-palvelua kohden vastaa yksi businessService-elementti.

bindingTemplate kuvaa tietyn Web-palvelun tekniset piirteet ja sidontainformaation (esimerkiksi URI:n, jossa komponentti on käytettävissä). Lisäksi sisältää viittauksia tModel-komponentteihin.

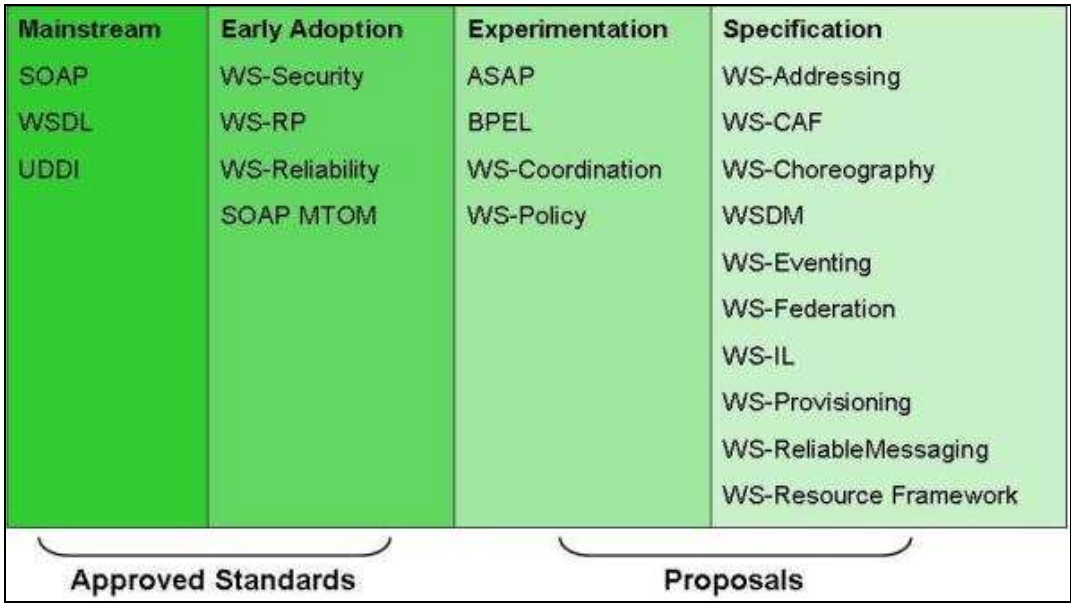
tModel sisältää esimerkiksi tarkat tekniset dokumentit tietystä Web-palvelusta. Kyseessä voi olla esimerkiksi WSDL-dokumentteja.

4 WEB-PALVELUIDEN KRITIIKKIÄ

Heather Greger (2003) kirjoittaa artikkelissaan, että Web-palvelut ovat nuori teknologia ja sen sisältämät protokollat ovat vielä osaksi kehitysvaiheessa. Web-palvelujen alle on kehitetty lukematon määrä erilaisia protokollia ja hankkeita, jotka kukin tulee hoitamaan oman osuutensa Web-palvelujen infrastruktuurista. Riippumattoman ja itsenäinen organisaatio CBDI julkaisee www-sivustollaan (2005) päivitettyä listaa Web-palveluiden protokollista. 28.3.2005 protokollia oli 56 kappaletta, joista vain noin puolet on julkaistu tai hyväksytty standardiksi. Pieni osa protokollista on hylätty ja suuri osa on edelleen luonnosasteella tai odottaa hyväksyntää. Suurin osa tutkijoiden kritiikistä kohdistuukin näihin keskeneräisiin standardeihin. Lisäksi protokollia standardisoivia organisaatioita on useita: W3C, OASIS, WS-I jne., mikä osaltaan lisää tilanteen monimutkaisuutta.

Business Domain Specific extensions	Various	**Business Domain**
Distributed Management	WSDM, WS-Manageability	**Management**
Provisioning	WS-Provisioning	
Security	WS-Security	**Security**
Security Policy	WS-SecurityPolicy	
Secure Conversation	WS-SecureConversation	
Trusted Message	WS-Trust	
Federated Identity	WS-Federation	
Portal and Presentation	WSRP	**Portal and Presentation**
Asynchronous Services	ASAP	**Transactions and Business Process**
Transaction	WS-Transactions, WS-Coordination, WS-CAF	
Orchestration	BPEL4WS, WS-CDL	
Events and Notification	WS-Eventing, WS-Notification	**Messaging**
Multiple message Sessions	WS-Enumeration, WS-Transfer	
Routing/Addressing	WS-Addressing, WS-MessageDelivery	
Reliable Messaging	WS-ReliableMessaging, WS-Reliability	
Message Packaging	SOAP, MTOM	
Publication and Discovery	UDDI, WSIL	**Metadata**
Policy	WS-Policy, WS-PolicyAssertions	
Base Service and Message Description	WSDL	
Metadata Retrieval	WS-MetadataExchange	

Kuva 24. Nykyinen Web-palvelut-protokollapino (Wilkes, 2005)



Kuva 25. Keskeisimpien protokollien nykyinen valmiusaste (Wilkes, 2005)

4.1 Tietoturva

Kuten luvussa 3.3 todettiin, Web-palveluiden tiedonsiirrossa käytettävä SOAP-protokolla ei itsessään sisällä tietoturvaan (Security) vaikuttavia elementtejä. Protokollassa nojataan siihen lähtökohtaan, että tietoturva lisätään liikennöintiin joko SOAP-viestien otsikko-osan laajennuksissa tai verkkoprotokollan avulla (esimerkiksi HTTP+SSL). Heather Kregerin (2003) mukaan merkittävin Web-palvelujen puute on kuitenkin tietoturva.

David Geerin artikkelissa *Taking Steps to Secure Web Services* esitellään neljä Web-palveluiden turvallisuuteen vaikuttavaa perusvaatimusta:

- Autentikointi (Authentication) vahvistaa käyttäjän
- Valtuuttaminen (Authorization) määrittelee sen, mitä käyttäjä saa tehdä
- Luottamuksellisuus (Confidentialy) varmistaa sen, että ainoastaan tarkoitettu vastaanottaja voi lukea viestin sisällön

- Eheys (Integrity) varmistaa, että viestiin ei ole kajottu kuljetuksen aikana. Eheys toteutetaan mm. digitaalisilla allekirjoituksilla.

Tyypillisesti turvallisimmat Web-perustaiset kommunikointitavat ovat käyttäneet tiedon turvaamiseen SSL-teknologiaa (secure sockets layer), jota on sovellettu myös Web-palvelujen yhteydessä. Web-palvelut kykenevät kuitenkin suoriutumaan huomattavan suuresta transaktiomääristä lyhyessä ajassa samalla kun SSL–menetelmässä data kryptataan jokaisella web-palvelimella (SOAP-välisolmulla) uudestaan. SSL-tekniikan skaalatutuvuus ei siten ole yleisesti riittävä Web-palveluiden tarpeisiin. Suojatut kuljetusprotokollat kuten SSL ja IPSec voivat tarjota eheyden ja luottamuksellisuuden ainoastaan välisolmujen välillä. Data on siis luettavassa muodossa jokaisessa välisolmussa, johon hyökkäämällä hakkerit saavat suuremmalla todennäköisyydellä haltuunsa heille tarkoittamatonta dataa. (Geer, 2003)

Web-palveluiden tiedon turvaamiseen on perustettu muutamia toimittajariippumattomia hankkeita. Esimerkiksi Microsoftin, IBM:n ja VeriSign:in käynnistämä *Web Services Security protocol* (WS-Sec) on suunniteltu määrittelemään kuinka SOAP-viestiin voidaan liittää sähköisiä allekirjoituksia sekä salattua tietoa. Toinen esimerkki on *Security Assertion Markup Language* (SAML), joka määrittelee kuinka SOAP-viesteissä voidaan esittää autentikointiin ja valtuutukseen liittyvää tietoa. Geer on kirjoittanut artikkelinsa vuonna 2003, jolloin nämä hankkeet olivat vielä keskeneräisiä. Tällä hetkellä (maaliskuu 2005) sekä WS-Sec ja SAML ovat julkaistu OASIS:en toimesta, jälkimmäinen kuitenkin vasta 9. maalikuuta 2005 (OASIS, 2005), joten kovin vakiintunutta tämän tekniikan käyttö ei vielä ole.

4.2 Sanomanvälityksen luotettavuus

Toinen merkittävä Web-palvelujen ongelma on luotettava sanomanvälitys (Reliability), jota ei ole otettu huomioon perusarkkitehtuurissa laisinkaan. Luotettavuudella tarkoitetaan Kregerin (2001) mukaan sitä, että lähetetty sanoma tai

myöhemmin saapuu tarkoitettuun määränpäähänsä kerran ja vain kerran ja jos viestiä ei saada toimitettua, toimitetaan lähettäjälle välitön palaute. Web-palvelujen yhteydessä käytettävä HTTP-protokolla ei tue luotettavuutta ja Kreger pohtikin vuosituhannen alussa sitä, että voisi olla suositeltavaa kehittää luotettava HTTP-protokolla. IBM onkin kehittänyt HTTPR–tiedonsiirtoprotokollan, joka sisältää tämän kaivatun ominaisuuden (Banks ym. 2002). Lisäksi OASIS on standardisoinut WS-Reliability-protokollan, joka tarjoaa luotettavuutta SOAP-viestin otsikko-osan laajennuksessa.

4.3 Prosessit

Prosesseille on tunnusomaista se, että alkutilan ja lopputilan välillä suoritetaan useita toimenpiteitä, joiden välillä on riippuvuuksia. Prosessien yhteydessä käytetäänkin laajalti synkronista tiedonsiirtoa, mutta Web-palvelut itsessään mahdollistaa vain yksinkertaisen synkronisen kommunikoinnin kahden sovelluksen välillä. Lisäksi prosessien yhteydessä käytettäviä transaktioita ei tueta alkuperäisessä Web-palvelujen arkkitehtuurissa.

Monimutkaisten liiketoimintaprosessien hallitsemiseen on ollut käynnissä muutamia hankkeita. Esimerkiksi *Busines Process Execution Language for Web Services* (BPEL4WS) määrittelyn ensimmäinen vedos (draft) julkaistiin heinäkuussa 2002 ja toinen vedos julkaistiin toukokuussa 2003. BPEL4WS-määrittely on siis edelleen (28.3.2005) luonnostilassa (Arkin, Askary, Bloch ym. 2005), työn pitäisi olla kuitenkin piakkoin loppusuoralla.

BPEL4WS:llä on riippuvuussuhde WS-AtomicTransaction ja WS-Coordination –märittelyihin, jotka on jo julkaistu, mutta ei standardisoitu.

4.4 Datan määrä

Arsanjani ym. (2003) mukaan XML:n ja muun tekstimuotoisen datan aiheuttama datan suuri määrä aiheuttaa ongelmia Web-palveluille. Tietyissä tapauksis-

sa voidaan joutua tilanteeseen, jossa XML:n nimien alku- ja loppumerkinnät aiheuttavat suurimman osan datan määrästä varsinaisen hyötykuorman jäädessä hyvin pieneksi. Myös tietotyyppeihin liittyvä datanmäärä ja sen jäsentäminen lisää kuormitusta.

4.5 Suorituskyky

Luvussa 4.1 tietoturvapohdinnan yhteydessä mainittiin Web-palveluiden kykenevän huomattavin suuriin transaktiomääriin lyhyessä ajassa, varsinkin jos nopeutta verrataan SSL-salauksen suorituskykyyn. Teknologia ei kuitenkaan pärjää vertailussa perinteisille binäärisille tiedonsiirtoteknologioille, johon merkittävimpänä syynä on luvussa 4.4 mainittu suuri datan määrä. Elfwing Paulsson & Lundberg (2002) ovat verranneet Web-palvelut-teknologialla toteutettua yksinkertaista web-sovellusta binääriseen CORBA-tekniikalla toteutettuun vastaavaan sovellukseen. Alustavissa testeissä suorituskyvyn ero oli CORBA:n hyväksi käsittämättömät 1:400, mutta tämä johtui suurimmaksi osaksi web-palvelimena käytetyn Apache Axix:en toteutuksessa olleista puutteista. Tutkijat tekivät muutoksia kokoonpanoon ja artikkelissa todetaan, että parhaalla mahdollisella, vuonna 2002 saatavilla olevalla tekniikalla, ero saatiin kurottua pienimmillään suhteeseen 1:7. Tutkijat kehottavatkin Web-palvelujen sovelluskehittäjiä keskittymään agenttien toteutuksessa kahteen tärkeään asiaan:

- Kommunikointi on käsiteltävä niin tehokkaasti kuin mahdollista
- On valittava sellainen jäsennin (parser), joka sopii parhaiten kyseisessä Web-palvelu-sovelluksessa käytettäviin XML-dokumentteihin.

5 YHTEENVETO

Tässä tutkielmassa esiteltiin toteutusriippumaton palveluorientoitunut Web-palvelut-teknologia, jonka avulla eri ohjelmointikileillä toteutetut ja eri alustoilla ajettavat tietojärjestelmät kykenevät keskustelemaan keskenään tietoverkon kautta. Teknologia sisältää suuren määrän eri protokollia ja käytäntöjä, joista esiteltiin esimerkein keskeisimmät, eli kielioppiin (XML), sanomanvälitykseen (SOAP), palvelunkuvaukseen (WSDL) ja palveluiden löytämiseen (UDDI) liittyvät tekniikat.

Web-palvelut-teknologia on uusi ja kehittyvä teknologia, joten se on saanut osakseen myös paljon kritiikkiä. Tässä tutkielmassa käytiinkin läpi keskeisimmät Web-palveluita kohdanneet haasteet ja esiteltiin niille esitettyjä ratkaisuja. Web-palveluiden ensimmäinen tuleminen koettiin vuosituhannen alussa, mutta ilmeisesti juuri keskeisten protokollien keskeneräisyyden vuoksi teknologia ei kyennyt menestynyt odotusten mukaisesti. Nyt vuoden 2005 alkupuolella eletään aikoja jolloin yhä useampi merkittävä Web-palveluiden protokolla on julkaistu ja standardisoitu, mutta edelleen kaivataan viimeisteltyä käytäntöä esimerkiksi prosessien hallintaan. Web-palveluiden kypsyys on kuitenkin vuosituhannen alkuun verrattuna selvästi eri tasolla, joten teknologian mahdollisuudet läpimurtoon alkavat olla olemassa.

6 LIITTEET

6.1 SOAP 1.2 XML Schema Definition

```
<!-- Schema defined in the SOAP Version 1.2 Part 1 specification
     Proposed Recommendation:
     http://www.w3.org/TR/2003/PR-soap12-part1-20030507/
     $Id: soap-envelope.xsd,v 1.1 2003/04/17 14:23:23 ylafon Exp $

     Copyright (C)2003 W3C(R) (MIT, ERCIM, Keio), All Rights Reserved.
     W3C viability, trademark, document use and software licensing rules
     apply.
     http://www.w3.org/Consortium/Legal/

     This document is governed by the W3C Software License [1] as
     described in the FAQ [2].

     [1] http://www.w3.org/Consortium/Legal/copyright-software-19980720
     [2] http://www.w3.org/Consortium/Legal/IPR-FAQ-20000620.html#DTD
-->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="http://www.w3.org/2003/05/soap-envelope"
           targetNamespace="http://www.w3.org/2003/05/soap-envelope"
                     elementFormDefault="qualified" >

  <xs:import namespace="http://www.w3.org/XML/1998/namespace" />

  <!-- Envelope, header and body -->
  <xs:element name="Envelope" type="tns:Envelope" />
  <xs:complexType name="Envelope" >
    <xs:sequence>
      <xs:element ref="tns:Header" minOccurs="0" />
      <xs:element ref="tns:Body" minOccurs="1" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <xs:element name="Header" type="tns:Header" />
  <xs:complexType name="Header" >
    <xs:annotation>
            <xs:documentation>
            Elements replacing the wildcard MUST be namespace qualified, but can be in the targetNamespace
            </xs:documentation>
          </xs:annotation>
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"  />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <xs:element name="Body" type="tns:Body" />
  <xs:complexType name="Body" >
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <!-- Global Attributes.  The following attributes are intended to be
  usable via qualified attribute names on any complex type referencing
  them.  -->
  <xs:attribute name="mustUnderstand" type="xs:boolean" default="0" />
  <xs:attribute name="relay" type="xs:boolean" default="0" />
  <xs:attribute name="role" type="xs:anyURI" />

  <!-- 'encodingStyle' indicates any canonicalization conventions
  followed in the contents of the containing element.  For example, the
  value 'http://www.w3.org/2003/05/soap-encoding' indicates the pattern
  described in the last call working draft of SOAP Version 1.2 Part 2:
  Adjuncts -->

  <xs:attribute name="encodingStyle" type="xs:anyURI" />

  <xs:element name="Fault" type="tns:Fault" />
  <xs:complexType name="Fault" final="extension" >
    <xs:annotation>
            <xs:documentation>
              Fault reporting structure
            </xs:documentation>
          </xs:annotation>
    <xs:sequence>
      <xs:element name="Code" type="tns:faultcode" />
      <xs:element name="Reason" type="tns:faultreason" />
      <xs:element name="Node" type="xs:anyURI" minOccurs="0" />
            <xs:element name="Role" type="xs:anyURI" minOccurs="0" />
      <xs:element name="Detail" type="tns:detail" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="faultreason" >
    <xs:sequence>
            <xs:element name="Text" type="tns:reasontext"
                  minOccurs="1"  maxOccurs="unbounded" />
          </xs:sequence>
  </xs:complexType>

  <xs:complexType name="reasontext" >
    <xs:simpleContent>
            <xs:extension base="xs:string" >
              <xs:attribute ref="xml:lang" use="required" />
            </xs:extension>
          </xs:simpleContent>
```

```
  </xs:complexType>

  <xs:complexType name="faultcode">
    <xs:sequence>
      <xs:element name="Value"
                  type="tns:faultcodeEnum"/>
      <xs:element name="Subcode"
                  type="tns:subcode"
                  minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="faultcodeEnum">
    <xs:restriction base="xs:QName">
      <xs:enumeration value="tns:DataEncodingUnknown"/>
      <xs:enumeration value="tns:MustUnderstand"/>
      <xs:enumeration value="tns:Receiver"/>
      <xs:enumeration value="tns:Sender"/>
      <xs:enumeration value="tns:VersionMismatch"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="subcode">
    <xs:sequence>
      <xs:element name="Value"
                  type="xs:QName"/>
      <xs:element name="Subcode"
                  type="tns:subcode"
                  minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="detail">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"  />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <!-- Global element declaration and complex type definition for header entry returned due to a mustUnderstand fault
-->
  <xs:element name="NotUnderstood" type="tns:NotUnderstoodType" />
  <xs:complexType name="NotUnderstoodType" >
    <xs:attribute name="qname" type="xs:QName" use="required" />
  </xs:complexType>


  <!-- Global element and associated types for managing version transition as described in Appendix A of the SOAP
Version 1.2 Part 1 Last Call Working Draft -->  <xs:complexType name="SupportedEnvType" >
    <xs:attribute name="qname" type="xs:QName" use="required" />
  </xs:complexType>

  <xs:element name="Upgrade" type="tns:UpgradeType" />
  <xs:complexType name="UpgradeType" >
    <xs:sequence>
            <xs:element name="SupportedEnvelope" type="tns:SupportedEnvType" minOccurs="1" maxOccurs="unbounded" />
          </xs:sequence>
  </xs:complexType>


</xs:schema>
```

LÄHDELUETTELO

Arkin A., Askary S., Bloch B., Curbera F., Goland Y., Kartha N., Liu C.K, Thattle S., Yendluri P., Yiu.A. 2005. Web Services Business Process Execution Language (BPEL4WS) 2.0. Working draft. [online]. [viitattu 29.3.2005]. Saatavilla www-muodossa <http://www.oasis-open.org/committees/download.php/11601/wsbpel-specification-draft-022705.htm>

Arsanjani A, Hailpern B, Martin J, Tarr Peri, 2003. Web Services Promises and Compromises. QUEUE volume 1 issue 1. ACM Press. 45-58,

Banks A., Challenger J., Clarke P., Davis D., King R.P., Witting K., Donoho A., Holloway T., Ibbotson J., Todd S. HTTPR Specification (April 2002) [viitattu 28.3.2005]. Saatavilla www-muodossa <http://www-128.ibm.com/developerworks/library/ws-httprspec/>

Bellwood T. 2002. UDDI Version 2.04 API Specification. UDDI Committee Specification, 19 July 2002. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf>

Booth D., Haas H., McCabe F., Newcomer E., Champion M., Ferris C., Orchard D. Web Services Architecture. 2004. W3C Working Group Note 11 February 2004 [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www.w3.org/TR/ws-arch/>

Bray T., Hollander D., Layman A. 1999. Namespaces in XML. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www.w3.org/TR/REC-xml-names/>

CBDI, 2005, Web Services Protocols Summary. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://roadmap.cbdiforum.com/reports/protocols/summary.php>

Cohen F. 2003. Discover SOAP encoding's impact on Web service performance. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc/>

Cowan J., Tobin R. 2004. XML Information Set (Second Edition). W3C Recommendation 4 February 2004 [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www.w3.org/TR/xml-infoset/>

Curbera F., Matthew D., Rania ., William N., Nirmal M., Sanjiva W. 2002. Unraveling the web Services Web, An Infroduction to SOAP, WSDL and UDDI. IEEE Internet Computing, 3-4/2002, 86–93

Elfwing R., Paulsson U & Lundberg L. 2002. Performance of SOAP in Web Service Environment to CORBA. Proceedings of the Ninth Asia-Pacific Software Engineering Conference. Computer Society

Geer D. 2003. Taking Steps To Secure Web Services. ACM Computer 2003. Sivut 14-16

Gisolfi D. 2001. Is Web services the reincarnation of CORBA?. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www-106.ibm.com/developerworks/webservices/library/ws-arc3/>

Greger H. 2003. Fulfilling the web services promise. Communications of the ACM. 46,6(2003), sivut 29-34.

Januszewski K. 2002. The Importance of Metadata: Reification, Categorization, and UDDI. MSDN Microsoft. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnuddi/html/impmetadata.asp>

Kreger H. 2001. Web Services Conceptual Architecture (WCSA 1.0). [online]. [viitattu 30.3.2005] Saatavilla www-muodossa <www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>

Linthicum D.S. 2004. Next Generation Application Integration. Boston: Addison-Wesley

Mitra Nilo. 2003. SOAP Version 1.2 Part 0: Primer. W3C Recommendation 24 June 2003. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>

OASIS, 2005, OASIS Security Services (SAML) TC. [online]. [viitattu 29.3.2005]. Saatavilla www-muodossa <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security>

Skonnard A. 2002. The Birth of Web Services. [online]. MSDN Magazine 10/2002. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://msdn.microsoft.com/msdnmag/issues/02/10/XMLFiles/>

Skonnard A. 2003a. Understanding SOAP. Microsoft MSDN. [online]. [viitattu 30.3.2005] Saatavilla www-muodossa <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/understandsoap.asp>

Skonnard A. 2003b. Understanding WSDL. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/understandwsdl.asp>

Thompson H. S., David B., Maloney M., Mendelson N. 2004. XML Schema Part 1: Structures Second Edition. W3C Recommendation 28 October 2004. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>

Walsh N. 1999. Understanding XML Schemas. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://www.xml.com/pub/a/1999/07/schemas/index.html>

Wilkes L. 2005. The Web Services Protocol Stack. [online]. [viitattu 30.3.2005]. Saatavilla www-muodossa <http://roadmap.cbdiforum.com/reports/protocols/>