SOAP-protokollan hyödyntäminen PHP-ohjelmoinnissa

Pauli Rikala

1.8.2007

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Web-palvelut ovat saavuttaneet suuren suosion ja niitä hyväksikäyttäen yhä useammat tahot kehittävät sovelluksiaan palveluorientoituneen arkkitehtuurin mukaisesti. SOAP on protokolla, joka mahdollistaa XML-muotoisen tiedon välittämisen hajautetussa ympäristössä ja siksi se on tärkeä osa web-palveluita. Tutkielman tarkoituksena on perehdyttää lukija SOAP-protokollaan ja sen hyödyntämiseen PHP-ohjelmoinnissa. Lisäksi esitellään web-paveluiden kuvaamiseen käytetty WSDL-kieli sekä Edustaja-suunnittelumalliin perustuva edistyneempi lähestymistapa SOAP-protokollan hyödyntämiseen PHP-ohjelmoinnissa. Tutkielma käsittelee asioita ohjelmoijan näkökulmasta siten, että monia esimerkkejä demonstroidaan ohjelmakoodin tasolla.

ACM-luokat (ACM Computing Classification System, 1998 version): C.2, D.2.11, D.2.12, D.3.3, H.4.3

Avainsanat: PHP, SOA, SOAP, UDDI, WSDL, XML, web-palvelut

Sisältö

1 Johdanto

Vuonna 2000 W3C hyväksyi SOAP-protokollan määritykseksi laaditun ehdotuksen. Tämä XML-perustainen viestinvälitysformaatti määritti mallin sovellustenväliselle kommunikoinnille HTTP-protokollaa käyttäen. Ei-kaupallisena teknologiana SOAP tarjosi houkuttelevan vaihtoehdon perinteisille patentoiduille teknologioille kuten CORBA ja DCOM. Vuonna 2001 W3C julkaisi WSDL-spesifikaation, joka oli myös XML-perustainen. Tämä standardi mahdollisti web-palveluiden rajapinnan kuvaamisen (Erl, 2004).

W3C (2004) määrittelee web-palvelun seuraavasti: w*eb-palvelu* (Web Service) on järjestelmä, joka on suunniteltu eri koneiden välillä verkon yli tapahtuvan vuorovaikutuksen mahdollistamiseksi. Sillä on rajapinta, joka on kuvattu sellaisessa muodossa, jota tietokoneella voidaan prosessoida. Muut järjestelmät ovat vuorovaikutuksessa web-palvelun kanssa käyttäen SOAP-viestejä tavalla, joka on määrätty palvelun kuvauksessa, eli WSDL-kuvauksessa. Yleensä viestit välitetään käyttämällä HTTP-protokollaa sekä XML-julkaisua yhdessä muiden web-standardien kanssa.

Web-palvelut koostuvat joukosta avoimia standardeja, joiden käyttö johtaa palveluorientoituneen arkkitehtuurin laajamittaiseen omaksumiseen (Newcomer et al., 2005). *Palveluorientoitunut arkkitehtuuri* (Service Oriented Architecture, SOA) on suunnittelun malli, joka perustuu toiminnallisuuksien koteloimiseen palveluiksi, jotka ovat vuorovaikutuksessa yleisten yhteysprotokollien välityksellä. Kun web-palveluita käytetään tämän standardien mukaisen yhteysverkoston toteuttamiseen, edustavat ne palveluorientoituneen arkkitehtuurin web-pohjaista toteutusta. Tämän kehyksen tarjoamasta riippumattomuudesta johtuen palvelujen koteloiman ohjelmointilogiikan ei tarvitse noudattaa mitään tiettyä alustaa tai teknologiaa, mikä sallii sovellusten vapaamman toteutuksen (Erl, 2004).

Web-palvelut ja palveluorientoituneen arkkitehtuurin mukainen toiminta tarjoavat lähes rajattomasti mahdollisuuksia erilaisiin ongelmiin, joista muutamia esimerkkitapauksia on esitelty seuraavassa (Gilmore, 2006):

- Kuvitellaan, että ollaan toteuttamassa kansainvälistä verkkokauppasovellusta, joka vaatii hintojen esittämistä monille eri valuutoille jatkuvasti vaihtuvien kurssien perusteella. Sen sijaan, että ratkaistaisiin itse nämä ongelmat keräämällä tiedot joltakin verkkosivulta pala palalta, voidaan yhdistää suoraan web-palveluun, joka palauttaa nämä halutut arvot. Tuloksena on huomattavasti selkeämpi ohjelmakoodi, joka ei ole niin virhealtis esimerkiksi verkkosivun ulkoasun muutoksille.

- Kehittäjät joutuvat uhraamaan suunnattomasti aikaa sovittaessaan erilaisia ohjelmia toisiinsa. Jos kaksi toisiinsa sovitettavaa ohjelmaa tukevat web-palveluita, voidaan yhdistämisprosessi standardoida riippumatta sovelluksien toteutuskielestä.

- Koska web-palvelut tarjoavat alustariippumattoman rajapinnan sovelluksen metodeille, niitä voidaan käyttää samanaikaisesti myös erilaisilla käyttöjärjestelmillä. Esimerkiksi verkkokauppasovelluksen palvelimella olevaa web-palvelua voidaan käyttää ylläpitämään myyntitilastoja Windows-pohjaisessa sovelluksessa ja lisäksi samaa palvelua voi käyttää Linux-palvelimella ajettava Perl-skripti, joka lähettää päivittäisiä myyntitilastoja sisältäviä sähköposteja.

- Koska web-palvelut yleensä välittävät tietoa HTTP-protokollan välityksellä, palomuurit eivät ole ongelma, koska portit 80 (HTTP) ja 443 (HTTPS) ovat yleensä aina auki.

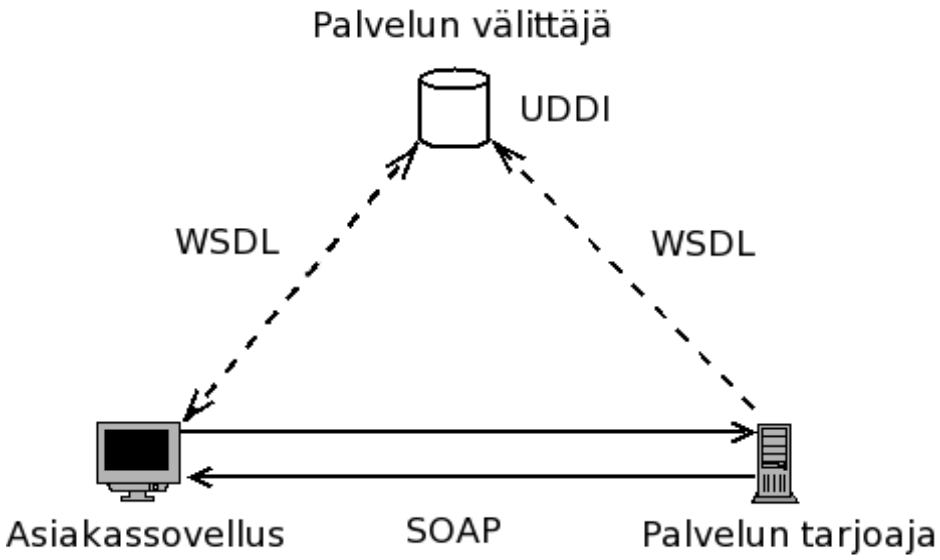
Kun on kyse web-palveluista, tulee SOAP:in ja WSDL:n lisäksi tuntea käsite UDDI. Nämä termit esiintyvät jatkuvasti web-palveluiden yhteydessä. Seuraavassa on esitelty lyhyet määritelmät kullekin termille (Newcomer, 2002):

- *WSDL* (Web Services Description Language): XML-perustainen teknologia, joka määrittelee web-palveluiden rajapinnan, välitettävän tiedon ja viestien tyypit, vuorovaikutusmallit sekä käytettävät protokollat.

- *SOAP* (Simple Object Access Protocol): Kokoelma XML-perustaisia teknologioita, joka määrittää kuoren web-palveluiden kommunikaatiolle ja tarjoaa sarjallistusformaatin XML-dokumenttien välittämiseksi verkon yli, sekä tavan etäproseduurikutsujen käyttöön.

- *UDDI* (Universal Description, Discovery, and Integration): Web-palveluiden rekisteröinti- ja havaitsemismekanismi, jota käytetään palveluiden tietojen kategorisointiin ja varastoimiseen, sekä web-palveluiden rajapintojen osoitteiden palauttamiseen.

Kuvassa 1 on kuvattu edellä määriteltyjen käsitteiden suhde web-palveluarkkitehtuurissa, kun käytössä on UDDI-rekisteri. Jos palvelun rajapinnan sijainti on ennestään tiedossa, ei UDDI:a ole pakko käyttää, ja siksi sen tarkempi käsittely on rajattu tämän tutkielman ulkopuolelle.

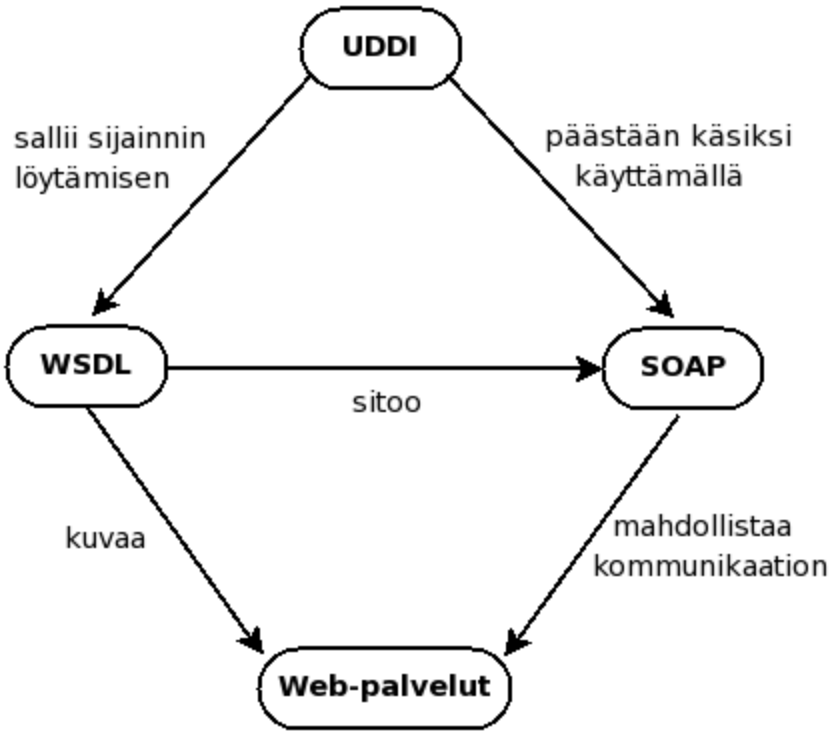


Kuva 1. Web-palvelun arkkitehtuuri.

Kuva 1 esittää web-palveluiden perusarkkitehtuurin, joka koostuu SOAP, WSDL ja UDDI-määrityksistä, jotka mahdollistavat vuorovaikutuksen asiakassovelluksen ja palvelun tarjoajan, sekä mahdollisesti palvelun välittäjän välillä. Yleensä palvelun tarjoaja julkaisee WSDL-kuvauksen tarjoamastaan web-palvelusta ja asiakassovellus hakee tämän kuvauksen käyttämällä UDDI:a tai jotakin muuta vastaavaa tapaa. Kun asiakassovelluksella on tiedossa palvelun kuvaus ja sijainti, se kutsuu palvelua lähettämällä sille SOAP-viestin (Newcomer et al., 2005). Vaikka tässä puhutaan asiakassovelluksesta ja palvelusta, ei osapuolten rooleja yleensä määritetä tarkasti, vaan ne voivat vaihtua tarpeen mukaan. Käytännössä palvelu voi siis toimia sekä asiakassovelluksena, että palvelun tarjoajana yhtä aikaa (Erl, 2004). Tässä tutkielmassa SOAP:in käyttöä tarkastellaan enimmäkseen asiakas-palvelin-arkkitehtuurin mukaisesti, jotta voidaan ilmentää kunkin roolin mukaista toimintaa mahdollisimman selkeästi.

Kuvassa 2 on esitetty käytettyjen standardien väliset suhteet käsitteellisellä tasolla. Kuten kuvasta ilmenee, UDDI-rekisteriä päästään hyödyntämään käyttämällä SOAP-viestejä. UDDI mahdollistaa palvelun WSDL-dokumentin sijainnin löytämisen ja WSDL määrittää SOAP-

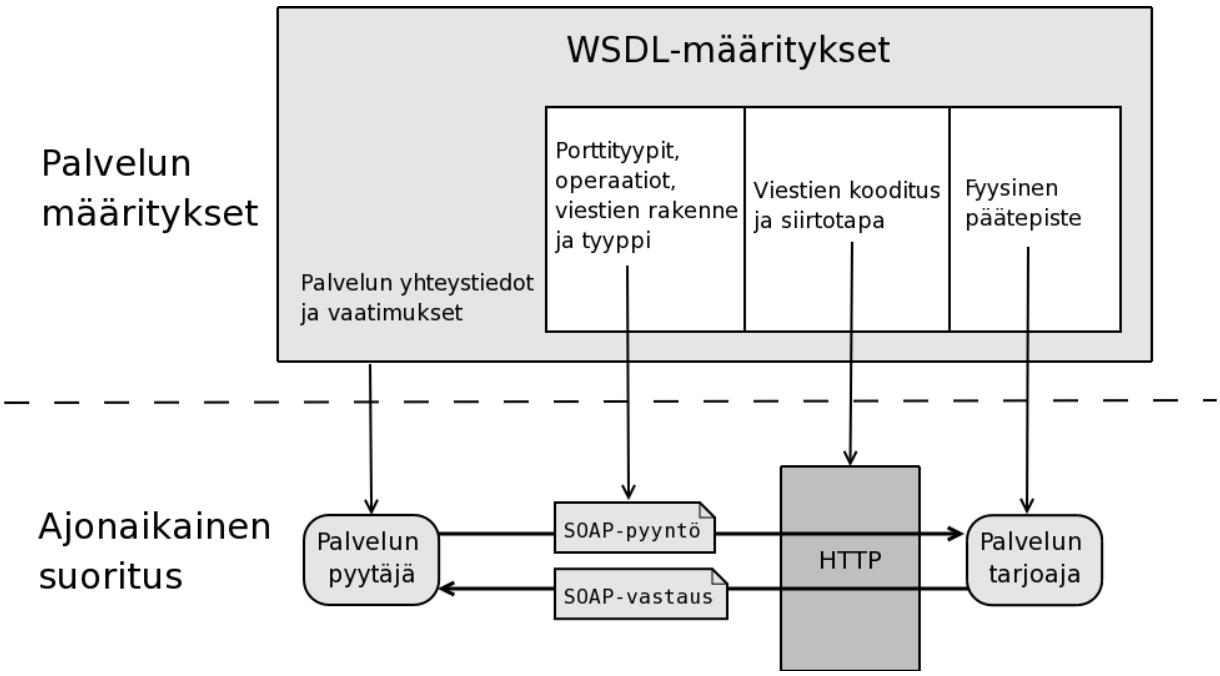
viesteissä käytettävän sidonnan, sekä kuvaa web-palvelun. SOAP mahdollistaa kommunikaation web-palveluiden välillä.



Kuva 2. Käytettyjen standardien väliset suhteet (Erl, 2004).

Web-palveluita ja asiakassovelluksia voidaan toteuttaa eri ohjelmointikielillä, kuten C#, Java ja PHP. Tässä tutkielmassa kieleksi on valittu *PHP,* joka on palvelimessa suoritettava html-koodiin upotettava skriptauskieli. Syntaksiltaan se on pääosin sekoitus C-, Java- ja Perl-ohjelmointikieliä. PHP:n avulla web-sivuille saa helposti ja nopeasti monimutkaistakin dynamiikkaa. Versio PHP 5 julkistettiin heinäkuussa 2004. Merkittävin muutos edelliseen versioon verrattuna on uusi Zend II-ydin, joka mahdollistaa entistä paremmin olioperustaisen ohjelmoinnin PHP:llä (The PHP-Group, 2007). PHP 5 toi mukanaan paljon uudistuksia myös XML-tiedon käsittelyyn ja mahdollisti siten monien XML-pohjaisten teknologioiden käytön. Natiivi tuki SOAP-protokollalle oli yksi näistä uudistuksista (Richards, 2006).

Kuva 3 ilmaisee tutkielmassa käsiteltävien asioiden suhdetta toisiinsa. Kuvan 3 katkoviivan yläpuolella sijaitsevat palvelun määritykset, joita käydään läpi luvussa 2 esittelemällä WSDL-dokumenttien perusrakenne ja sidontaan liittyvää informaatiota, sekä WSDL-dokumentin käyttöä SOAP:in tukena. Luvussa 3 esitellään SOAP-viestien (pyyntö ja vastaus kuvassa 3) rakenne käymällä läpi niiden olennaisimmat elementit ja niiden ominaisuudet. Lisäksi käydään läpi SOAP-protokollan mukaista virheiden käsittelyä. Luvussa 4 tutustutaan esimerkkien avulla SOAP:in käyttöön PHP:n SOAP-laajennuksen tarjoamin keinoin asiakas- ja palvelinsovelluksien näkökulmasta. Luvussa 4 käsiteltävät asiat sijoittuvat kuvassa 3 palvelun pyytäjän ja palvelun tarjoajan toiminnan määrittämiseen, sekä SOAP-pyynnön ja SOAP-vastauksen käsittelyyn ohjelmallisesti. Luvussa 5 esitellään Edustaja-suunnittelumallin muodossa edistyneempi näkökulma PHP:n olioperustaisten ominaisuuksien hyödyntämiseen SOAP:in kanssa toimiessa. Tutkielman päättää luvun 6 yhteenveto.



Kuva 3. Tutkielmassa käsiteltävien asioiden rakenteellinen kuvaus.

2 WSDL-määrittelykieli

WSDL (Web Services Description Language) on XML-pohjainen kieli, jota käytetään web-palveluiden kuvaamiseen (W3C, 2001). Se määrittelee kuinka web-palveluun päästään käsiksi, kuinka viestit välitetään, sekä viestien rakenteen. Vaikka WSDL:ää ei vaadita SOAP:in kanssa toimimiseen, se on olennainen osa WS-I:n (2007) perusprofiilia, joka tarjoaa yhteentoimivuusopastusta web-palvelujen käyttöön. Se tekee myös SOAP:in kanssa toimimisen paljon helpommaksi. Hyvä asia WSDL:n kanssa toimimisessa on se, että palvelun käyttäjän näkökulmasta katsottuna ei tarvitse tietää yhtään rajapinnan yksityiskohtaa. Itseasiassa WSDL:n ymmärrystä ei edes tarvita palvelun käyttämiseksi, mutta jos joutuu itse vastuuseen web-palvelun kehittämisestä, täytyy osata kirjoittaa WSDL-dokumentteja (Richards, 2006).

WSDL 1.1 on de facto -standardi web-palveluiden kuvaamiseen. Monet palveluntarjoajat tukevat sitä niin kehitystyökaluissa kuin sovellusympäristöissä. SOAP-protokollan yleistymisen jälkeen WSDL:stä on tullut suosituin web-palvelujen kuvaamiseen käytetty standardi (Weerawarana et al., 2005). Kuvassa 4 on esitetty pelkistetty WSDL-dokumentin rakenne.

Kuva 4. WSDL-dokumentin pelkistetty rakenne

Kuten kuvasta 4 ilmenee, WSDL-dokumentti koostuu abstraktista ja konkreettisesta osasta. Abstraktissa osassa määritetään tietoja palvelun rajapinnasta. Näitä tietoja ovat käytettävät tyypit, viestin rakenne, operaatiot ja porttityypit (vrt. kuva 3). Konkreettinen osa määrittää viestien koodituksen ja siirtotavan, sekä palvelun sijainnin (vrt. kuva 3). Seuraavaksi esitellään, kuinka edellä mainittuja tietoja määritellään WSDL-dokumentissa.

2.1 Määritelmät

WSDL-dokumentin juurielementtinä on `definitions`-elementti, joka koostuu joukosta määritelmiä (Richards, 2006). Määritelmät sisältävät tyypit (`types`), viestin (`message`), porttityypin (`portType`), sekä sidonnan (`binding`) ja palvelun (`service`). Kuvassa 5 on esimerkki WSDL-dokumentin rakenteesta. Elementit `types`, `message` ja `portType` ovat abstrakteja osia, jotka määrittävät käytettävän rajapinnan ja elementit `binding` ja `service` ovat osia, jotka määrittävät palvelun konkreettiseen toteutukseen liittyviä asioita. Liitteessä 1 on esitelty WSDL-dokumentti, jota tarkastellaan tämän luvun esimerkeissä. Liitteen 1 WSDL-dokumentissa ei ole esitelty kaikkia WSDL-dokumentin valinnaisia piirteitä.

Elementti `definitions` sisältää joukon määritelmiä yhdelle nimiavaruudelle, jota kutsutaan kohdenimiavaruudeksi (target namespace). Nimiavaruus on eräänlainen omistusoikeus dokumentin määritelmille (Weerawarana et al., 2005). Yksinkertaisuuden vuoksi kuvan 5 esimerkissä ei ole määritelty kuin se nimiavaruus, johon tärkeimmät elementit kuuluvat. Liitteessä 1 on nimiavaruuksien käyttö esitelty tarkemmin. Seuraavaksi tarkastellaan tarkempia kuvauksia `definitions`-elementin lapsielementeistä.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <!-- Määritelmä tyypeistä, joita WSDL-dokumentissa käytetään -->
  </types>

  <message>
    <!-- Abstrakti määritelmä tiedosta, jota välitetään  -->
  </message>

  <portType>
    <!-- Joukko abstrakteja operaatioita, jotka
         viittaavat input ja output viesteihin     -->
  </portType>

  <binding>
    <!-- Konkreetiset protokollan ja tiedon formaatin määritteet -->
  </binding>

  <service>
    <!-- Määrittelee sijainnit ja sidonnat palvelua varten -->
  </service>
</definitions>
```

Kuva 5. WSDL-dokumentin rakenne (Richards, 2006).

2.2 Tyypit

Elementti `types` pitää sisällään niiden tietotyyppien määrittelyt, joita käytetään viestejä välitettäessä. Maksimaalisen yhteentoimivuuden ja alustariippumattomuuden takia suositellaan käytettäväksi XML-skeemaa (W3C, 2007b), joka varmistaa sen, että yhteensopivuusongelmia ei pitäisi tulla (W3C, 2001). Elementin määrittely on muotoa:

```
<types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- complexType määritelmät -->
  </xsd:schema>
</types>
```

Elementin käyttö riippuu välitettävän datan tyypistä. Esimerkiksi jos välitetään yksinkertaisia merkkijonoja edestakaisin, voidaan niitä hallita käyttämällä yksinkertaista `xsd:string`-tyyppiä, jota ei tässä tarvitse määritellä. Oliot ja taulukot sen sijaan tarvitsevat typpimäärittelyt

(Richards, 2006). Tarkempiin tyyppimäärittelyihin palataan myöhemmin esimerkkien muodossa.

2.3 Viestit

Viestielementti (`message`) on abstrakti määritelmä välitettävästä datasta. Yksinkertaisesti ilmaistuna viestit määrittävät syöte- ja tulosteparametrit. Tämä tulkinta on helpoin tapa ymmärtää viesti ilman, että tarvitsee tietää jokaisesta WSDL:n ja SOAP:n yksityiskohdasta, jotka löytyvät W3C:n määrityksistä. Kun käytetään tällaista yksinkertaista lähestymistapaa, asiakassovelluksen täytyy tietää käytetyt tyypit viestiä lähettäessä, mikä voidaan rinnastaa funktion kutsumiseen. Täytyy myös tietää tyypit, joiden mukaista tietoa "funktio" palauttaa. Palvelimella tulee olla myös samat tiedot, mutta tässä yhteydessä tarvitsee tietää vain ne tyypit, joita oletetaan funktion kutsussa ja palautuksessa (Richards, 2006).

Elementin `message` käsittelyyn on olemassa kaksi sidontatyyliä, RPC (Remote Procedure Call) ja dokumentti (document). Kumpikin voi käyttää toista kahdesta sidonnasta, jotka ovat tyyppiä `encoded` tai `literal`. Tyyli ja valitun sidonnan käyttö määrää sen, kuinka `message`-elementti kirjoitetaan (Richards, 2006). Sidontaan palataan tarkemmin kohdassa 2.5 ja sidontatyyppien eroja on käsitelty kohdassa 2.6. Elementin määrittely on muotoa:

```
<message name="nmtoken">
   <part name="nmtoken" element="qname"? type="qname"?/> *
</message>
```

Jokaisella viestielementillä tulee olla uniikki nimi, joka erottaa sen muista viestielementeistä. Viestielementti voi sisältää `part`-elementtejä, jotka esittävät parametrit tai funktion vastauksen. Jokainen `part`-elementti sisältää myös `name`-attribuutin, jonka tulee identifioida jokaisen viestielementin alaisuudessa oleva `part`-elementti. Attribuutit `type` ja `element` ovat

toisensa poissulkevia. RPC-tyyliä käytettäessä, täytyy käyttää `type`-attribuuttia ja dokumenttityyliä käytettäessä tulee käyttää `element`-attribuuttia (Richards, 2006).

2.4 Porttityypit

Porttityyppi (`portType`) on nimetty joukko operaatioita ja mukana olevia viestejä. Attribuutti `name` antaa uniikin nimen, joka identifioi porttityypin kyseessä olevan WSDL-dokumentin sisällä (W3C, 2001). Porttityyppi määrittää palvelun tukemat operaatiot, mikä käytännössä tarkoittaa sitä, että määritellään toisiinsa liittyvät välitettävät viestit ryhmiksi (Weerawarana et al., 2005). Porttityypin määrittely on muotoa:

```
<portType name="nmtoken">
  <operation name="nmtoken" .... /> *
</portType>
```

Operaatiot nimetään ja määritetään viesteillä, jotka palvelu lähettää tai vastaanottaa. Jokainen operaatio voi lähettää tai vastaanottaa enintään yhden viestin kuhunkin suuntaan (Weerawarana et al., 2005). WSDL 1.1 -versiossa on neljä erilaista operaatiotyyppiä (W3C, 2001): one-way, request-response, solicit-response ja notification.

One-way operaatiotyypin tapauksessa viesti tulee palvelulle, mutta palvelu ei tuota mitään vastaukseksi. Kuvassa 6 `input`-elementti määrittää viestin formaatin yksisuuntaista operaatiota varten (W3C, 2001). Tätä operaatiotyyppiä voidaan käyttää esimerkiksi asynkronisen palvelun kutsussa, jossa palvelulle lähetetään kutsu, jonka palvelu käsittelee. Tässä tapauksessa palvelu ei odota mitään vastaukseksi (Goel et al., 2006a).

```
<wsdl:definitions .... >
  <wsdl:portType .... > *
    <wsdl:operation name="nmtoken">
      <wsdl:input name="nmtoken"? message="qname"/>
    </wsdl:operation>
  </wsdl:portType >
</wsdl:definitions>
```

Kuva 6. One-way-operaatiotyyppi.

Request-response-operaatiotyypin tapauksessa viesti tulee palvelulle ja palvelu tuottaa viestin vastaukseksi. Kuvassa 7 `input`- ja `output`-elementit määrittävät viestin formaatin pyyntöä ja vastausta varten. Valinnainen `fault`-elementti määrittää viestin formaatin mahdollista operaation suorituksesta seuraavaa virhetilannetta varten (W3C, 2001). Tätä operaatiotyyppiä voidaan käyttää synkronisen palvelun kutsussa, jossa asiakassovellus lähettää viestin palvelulle ja jää odottamaan vastausta. Tässä tapauksessa palvelu vastaanottaa viestin, prosessoi sen ja lähettää vastauksen asiakassovellukselle (Goel et al., 2006a).

```
<wsdl:definitions .... >
  <wsdl:portType .... > *
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
      <wsdl:input name="nmtoken"? message="qname"/>
      <wsdl:output name="nmtoken"? message="qname"/>
      <wsdl:fault name="nmtoken" message="qname"/>*
    </wsdl:operation>
  </wsdl:portType >
</wsdl:definitions>
```

Kuva 7. Request-response-operaatiotyyppi.

Solicit-response-operaatiotyypin palvelu lähettää viestin ja saa takaisin vastauksen. Kuvassa 8 `output`- ja `input`-elementit määrittävät viestin formaatin pyyntöä sekä vastausta varten. Valinnainen `fault`-elementti määrittää viestin formaatin mahdollista operaation suorituksesta seuraavaa virhetilannetta varten (W3C, 2001). Tätä operaatiotyyppiä voidaan käyttää esimerkiksi synkronisessa palvelussa, joka tarkkailee usean sovelluksen tilaa (Goel et al.,

2006b). Esimerkiksi jos kyseessä oleva palvelu toimittaa säätietoja usealta eri sääasemalta, voi palvelu halutessaan pyytää tietoja sääasemalta ja saada tiedot vastauksena. Tämä operaatiotyyppi on lähes samanlainen kuin request-response-tyypin operaatio, asiakassovellus ja palvelu vain vaihtavat rooleja.

```
<wsdl:definitions .... >
  <wsdl:portType .... > *
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
      <wsdl:output name="nmtoken"? message="qname"/>
      <wsdl:input name="nmtoken"? message="qname"/>
      <wsdl:fault name="nmtoken" message="qname"/>*
    </wsdl:operation>
  </wsdl:portType >
</wsdl:definitions>
```

Kuva 8. Solicit-response-operaatiotyyppi.

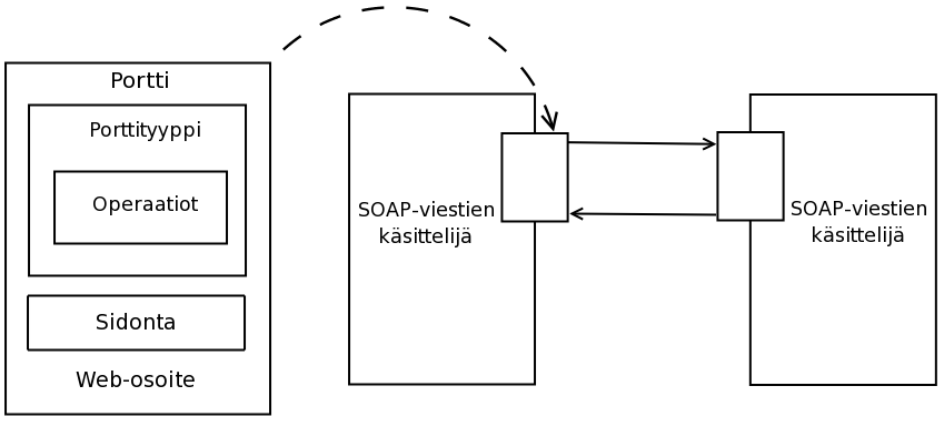
Notification-operaatiotyypin palvelu lähettää viestin eikä saa mitään vastaukseksi. Kuvassa 9 `output`-elementti määrittää viestin formaatin operaatiota varten. Tätä operaatiotyyppiä voidaan käyttää esimerkiksi asynkronisessa palvelussa takaisinkutsua (callback) suoritettaessa. Yleensä one-way- ja notification-tyyppisiä operaatioita käytetään yhdessä asynkronisia palveluja määriteltäessä (Newcomer et al., 2005).

```
<wsdl:definitions .... >
  <wsdl:portType .... > *
    <wsdl:operation name="nmtoken">
      <wsdl:output name="nmtoken"? message="qname"/>
    </wsdl:operation>
  </wsdl:portType >
</wsdl:definitions>
```

Kuva 9. Notification-operaatiotyyppi

2.5 Sidonta

Sidonta (`binding`) määrittelee konkreettisia yksityiskohtia porttityypistä (`portType`) ja protokollasta. Yksinkertaisesti ilmaistuna sidonnat tarjoavat tietoa käytetystä tiedonsiirtoprotokollasta ja viestien formaatista. WSDL 1.1 -versiossa voidaan käyttää kolmea erilaista sidontaa: SOAP, HTTP ja MIME (Richards, 2006). Koska tämä tutkielma käsittelee SOAP-protokollaa, keskitytään pelkästään SOAP-sidontaan. Kuvassa 10 on esitetty sidonnan ja muiden WSDL:n elementtien roolia sekä sijoittumista kokonaisuuteen. Elementit `types`, `message` ja `portType` muodostavat kuvauksen siitä, mitä ja millaisia viestejä palvelu lähettää ja vastaanottaa. Elementti `binding` kuvaa sen, missä formaatissa edellä mainittuja viestejä käsitellään, kun ollaan yhteydessä toiseen palveluun (Weerawarana et al., 2005). Eli käytännössä sidonta tarkoittaa sitä, että abstrakti porttityypin määritys saa konkreettiset ohjeet viestien käsittelyyn. Kun palvelun määritys täydennetään (kohta 2.5.6) ja se liitetään tiettyyn osoitteeseen, toimii kuvan 10 mukainen portti viestejä välittävänä elementtinä SOAP-viestien käsittelijöiden välillä.



Kuva 10. WSDL:n elementtien sijoittuminen kokonaisuuteen.

2.5.1 *SOAP-sidonta*

WSDL tarjoaa sidontaelementin `soap:binding`. Tämän elementin käyttäminen osoittaa sen, että käytetään SOAP-sidontaa:

```
<soap:binding transport="uri" style="rpc|document"?>
```

SOAP-sidonta määrittää sen, kuinka `message`-elementin `input`- ja `output`-viestejä käsitellään ja kuinka niistä muodostetaan SOAP-viestiin sisällytettävä `Envelope`-elementti (ks. 3.1.2). Attribuutti `transport` ilmaisee laadun SOAP-siirrolle, jota sidonta käyttää. Yleensä tässä käytetään HTTP-protokollaa, jolloin nimiavaruudeksi määritetään `http://schemas.xmlsoap.org/soap/http`. Attribuutti `style` asettaa oletustyylin jokaiselle operaatiolle, joka sisältyy WSDL-sidontaan ja se voi saada joko arvon `rpc` tai `document`. Tämä attribuutti ei ole pakollinen ja oletuksena sen arvo on `document`, jos sitä ei erikseen määritellä (Richards, 2006).

Jos käytetään `rpc`-tyyliä, kaikki `message`-elementin osat kääritään johonkin ulkoiseen ennalta määrättyyn elementtiin, joka sisältää kaikki tarvittavat tiedot. Tämä kääre-elementti sisällytetään yksittäisenä lapsielementtinä muodostettavan SOAP-viestin `Body`-elementtiin (ks. 3.1.6). Jos käytetään tyyliä `document`, asetetaan kaikki `message`-elementin osat suoraan SOAP:in `Envelope`-elementin lapsielementeiksi (Weerawarana et al., 2005). Tarkempia kuvauksia sidontatyyleistä käsitellään kohdassa 2.6.

Käytettävä sidontatyyppi (`encoded` tai `literal`) määritellään kohdassa 2.5.3 WSDL-sidonnan rakenteessa. Jos halutaan määrittää SOAP-sidonta käyttäen `document`-tyyliä ja HTTP-protokollaa, voidaan määritys tehdä seuraavilla tavoilla (Richards, 2006):

```
<soap:binding transport = "http://schemas.xmlsoap.org/soap/http"
              style = "document" />

<soap:binding transport = "http://schemas.xmlsoap.org/soap/http" />
```

2.5.2 *WSDL-operaatio*

WSDL-operaatioelementti (`operation`) eli WSDL-operaatio esiintyy `binding`-rakenteessa sekä määrittelee sidontainformaatiota saman nimiselle `operation`-elementille, joka on määritelty `portType`-rakenteessa (kuvat 6-9). Käyttämällä erillisiä nimiä voidaan varmistua siitä, että kaikki liitokset suoritetaan oikein. Kun käytetään SOAP-sidontaa, toimii SOAP-operaatioelementti (`soap:operation`) lapsielementtinä WSDL-operaatioelementille ja sen määrittely on muotoa:

```
<soap:operation soapAction="uri"? style="rpc|document"?>?
```

Kun käytetään SOAP-sidontaa, voidaan WSDL-operaatioelementti luoda seuraavalla tavalla:

```
<operation name="getPeopleByFirstLastName">
   <soap:operation soapAction="getPeopleByFirstLastName"/>
   <!-- input, output ja fault-elementtien määritykset -->
</operation>
```

Tässä esimerkissä WSDL-operaation nimeä käytetään `soapAction`-attribuutin nimenä. Koska `style`-attribuuttia ei ole määritelty, käytetään `soap:binding`-elementin vastaavaa arvoa (Richards, 2006).

2.5.3 *WSDL-operaation input-, output- ja fault-elementit*

WSDL-operaatiossa määritellään myös `input-`, `output-` ja `fault`-elementit. Elementit, joita tarvitsee käyttää, riippuvat määritellyn WSDL-operaation tyypistä. Sidonnassa `input-` ja `output`-elementit saavat kukin valinnaisen nimiattribuutin ja `fault`-elementille se on pakollinen. Nimiattribuutin käyttö määritetään lisäämällä attribuutti siihen elementtiin, johon se halutaan sisällyttää. Kun tehdään näin porttityypin alaisuudessa, samaa nimeä pitäisi soveltaa elementtiin koko sidonnan laajuudessa.

Jos katsotaan operaatiota `getPeopleByFirstLastName`, joka kuuluu `ExamplePortType`-porttityyppiin ja joka käyttää request-response-formaattia, voidaan käyttää seuraavanlaista `input`-elementtiä:

```
<input message="tns:getPeopleByFirstLastName"/>
```

Tämä elementti ei sisällä nimiattribuuttia, joten sen sidonnan määrittäminen onnistuu helposti:

```
<input>
  <!-- Sisältö tähän, esimerkiksi <soap:body use="literal"> -->
</input>
```

Samat asiat tehdään `output-` ja `fault`-elementeille. Elementit `input` ja `output` sisältävät sidonnassa yleensä `soap:body`-elementin, joka on muotoa:

```
<soap:body parts="nmtokens"? use="literal|encoded"?
           encodingStyle="uri-list"? namespace="uri"?>
```

Attribuutti `parts` määrittää, mitkä viestin osat ilmestyvät välitettävän SOAP-viestin `body`-osassa. Yleensä kaikki viestin osat sisällytetään ja tätä attribuuttia ei käytetä. Attribuutti `use` määrittää sen, kooditetaanko viestin osia (`encoded`) vai määrittävätkö osat viestin rakenteen

(literal). Kun käytetään arvoa literal, muita attribuutteja ei vaadita. Kun käytetään arvoa encoded, encodingStyle- ja namespace-attribuutit ovat käytössä. Kun käytetään SOAP:ia, arvo encodingStyle-attribuutille on http://schemas.xmlsoap.org/soap/encoding/. Arvo nimiavaruuden attribuutille on yleensä sama kuin attribuutilla targetNamespace. Jos oletetaan, että kohdenimiavaruus on urn:ExampleAPI, arvo tälle attribuutille tulee olemaan myös urn:ExampleAPI. Elementti soap:fault toimii sidonnassa kuten soap:body-elementti, mutta sillä on name-attribuutti, eikä sillä ole parts-attribuuttia. Attribuutin name arvon tulee olla sama kuin porttityypissä määritellyn fault-elementin name-attribuutilla, jotta ne voisivat olla liitoksissa toisiinsa (Richards, 2006). Määrittely fault-elementille sidonnassa on muotoa:

```
<soap:fault name="nmtoken" use="literal|encoded"
            encodingStyle="uri-list"? namespace="uri"?>
```

2.5.4 *SOAP-operaation header- ja headerfault-elementit*

Valinnaiset elementit soap:header ja soap:headerfault määrittelevät otsikot, jotka välitetään SOAP-viestin valinnaisessa soapenv:Header-elementissä. Tähän palataan myöhemmin luvussa 3 SOAP-viestien käsittelyn yhteydessä. Määrittely soap:header- ja soap:headerfault-elementeille on muotoa:

```
<soap:header message="qname" part="nmtoken" use="literal|encoded"
             encodingStyle="uri-list"? namespace="uri"?>*

<soap:headerfault message="qname" part="nmtoken" use="literal|encoded"
                  encodingStyle="uri-list"? namespace="uri"?/>*
```

Attribuutin message arvo on sama kuin WSDL-dokumentissa määritellyn message-elementin name-attribuutin arvo. Attribuutit part ja message muodostavat yhdessä viittauksen viestin

osaan, joka määrittää SOAP-viestin `Header`-elementin sisällön. Jäljellä olevat attribuutit määritellään samalla tavalla kuin `soap:body`- ja `soap:fault`-elementit (Richards, 2006).

2.5.5 *Esimerkki sidonnasta*

Tässä vaiheessa on vielä vaikea esitellä WSDL:n käyttöä SOAP-protokollan kanssa, koska joudutaan viittaamaan paljon SOAP-protokollaan liittyviin yksityiskohtiin, joita ei ole vielä käsitelty. Seuraava esimerkki kuvassa 11 kuvaa `portType`-määritystä, joka on request-response-tyyppinen operaatio.

```
<portType name="ExamplePortType">
  <operation name="getPeopleByFirstLastName">
    <input message="tns:getPeopleByFirstLastName"/>
    <output message="tns:getPeopleByFirstLastNameResponse"/>
    <fault name="nodb" message="tns:DBUnavailableFault"/>
    <fault name="sysmaint" message="tns:SystemMaintenance"/>
  </operation>
  <!-- Muut operaatiot -->
</portType>
```

Kuva 11. Porttityypin määritys.

Käyttämällä kaikkea edellä mainittua informaatiota sidonnasta, voidaan kirjoittaa sidonta-rakenne kuvan 11 porttityypille kuvan 12 osoittamalla tavalla käyttäen `document/literal`-tyyliä (Richards, 2006).

```
<binding name="ExampleBinding" type="tns:ExamplePortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getPeopleByFirstLastName">
    <soap:operation soapAction="getPeopleByFirstLastName" />
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
    <fault name="nodb">
      <soap:fault name="nodb" use="literal"/>
    </fault>
    <fault name="sysmaint">
      <soap:fault name="sysmaint" use="literal"/>
    </fault>
  </operation>
  <!-- Muut operaatiot -->
</binding>
```

Kuva 12. Sidonta kuvan 11 porttityypille.

2.5.6 *WSDL-palvelu*

Elementti `service` kuvaa tietyn web-palvelun tarjoamalla nimen ja sijainnin, sekä liittämällä sidonnan tiettyyn porttiin. Se on kokoelma portteja, viitattuna ikäänkuin päätepisteinä, jotka hyödyntävät sidontaa. Määrittely `service`-elementille on muotoa:

```
<service name="nmtoken">*
  <port name="nmtoken" binding="qname">*
    <!-- Palvelun osoite -->
  </port>
</service>
```

WSDL-dokumentti voi sisältää useita palveluja, mutta yleisesti käytetään vain yhtä palvelua eli `service`-elementtiä. Palvelulle annetaan aina nimi käyttäen `name`-attribuuttia. Jos määritellään useita palveluja, jokaiselle palvelulle annetaan yksikäsitteinen nimi. Elementti `service` sisältää kokoelman yksikäsitteisesti nimettyjä portteja palvelua varten. On mahdollista, että portit voivat tarjota palveluja muillekin sidonnoille kuin SOAP:ille, kuten esimerkiksi HTTP GET- ja POST-operaatioille. Ainoa sääntö porttielementeille `name`- ja `binding`-

attribuuttien oikeanlaisen nimeämisen lisäksi on se, että portin tulee määritellä osoite. Kun käytetään SOAP-sidontaa, määritellään osoite `location`-attribuutilla `soap:address`-elementissä:

```
<soap:address location="uri"/>
```

Jos oletetaan palvelun sijainniksi `http://www.example.com/ExampleService`, voidaan portille linkittää kuvan 12 `ExampleBinding`-sidonnassa määritellyt säännöt (Richards, 2006):

```
<service name="ExampleService">
  <port name="ExamplePort" binding="tns:ExampleBinding">
    <soap:address location="http://www.example.com/ExampleService"/>
  </port>
</service>
```

Nyt on läpikäyty kokonainen WSDL-dokumentti, joka löytyy kokonaisuudessaan liitteestä 1.

2.6 Sidontatyylit

WSDL:ssä käytettävä SOAP-sidonta voi olla tyyliltään joko RPC tai document. Sidonta voi olla joko tyyppiä encoded tai literal. Tästä seuraa, että voidaan määritellä 4 erilaista tyyliä, joita ovat RPC/encoded, RPC/literal, document/encoded ja document/literal. Tosin tyyli document/encoded ei noudata WS-I:n perusprofiilia ja se on muutenkin epäkäytännöllinen, joten sitä ei yleensä käytetä. Tulee ottaa huomioon, että sidontatyyleillä ei ole mitään tekemistä ohjelmointimallin kanssa, vaan niiden ainoa tehtävä on määrittää se, kuinka WSDL-sidonta vaikuttaa SOAP-viestien luomiseen. Eli esimerkiksi RPC-tyylin käyttö ei ole sidoksissa RPC-ohjelmointimallin käyttöön, vaikka nimi siihen erehdyttävästi viittaakin (Butek, 2005).

Seuraavaksi esitellään kolme yleisintä sidontatyyppiä, sekä paranneltu versio tyypistä document/literal kuvaamalla kunkin tyypin osalta WSDL-määritys ja sidonnan käytöstä seuraava SOAP-viestin rakenne. Seuraavaksi käsiteltävissä esimerkeissä määritykset on tehty metodille `void myMethod(int x, float y)`, joka ei palauta mitään. Esimerkeissä metodia kutsutaan parametrin `x` arvolla 5 ja parametrin `y` arvolla 5.0.

2.6.1 *RPC/encoded*

Tyyliä RPC/encoded pidetään helpoiten käytettävänä sekä WSDL-dokumentin kirjoittamisen että WSDL-dokumenttia käyttävän asiakassovelluksen näkökulmasta. Luotava SOAP-viesti sisältää parametrien nimet elementtien niminä, jotka sisältävät välitettävät arvot. Koska käytetään tyyppiä RPC, elementit kääritään yhteen elementtiin, jonka nimi määritetään sidonnassa (Richards, 2006). Kuvassa 13 on esitetty tyypin RPC/encoded WSDL-määritys ja sen mukainen SOAP-viesti.

Vahvuutena tässä tyypissä on se, että WSDL-määritys on erittäin suoraviivainen ja operaation nimi sisältyy viestiin, joten vastaanottajan on helppo poimia viestistä operaation suorittamiseen vaadittavat tiedot (Butek, 2005). Toisaalta kooditusinformaatio (kuva 13), `"xsi:type="xsd:int"`, on yleensä vaikeaselkoista, mikä vaikeuttaa dokumentin luettavuutta. Viestin validointi saattaa olla hankalaa, koska vain rivit `<x ...>5</x>` ja `<y ...>5.0</y>` sisältävät skeemassa (`http://www.w3.org/2001/XMLSchema`) määriteltyjä asioita. Loput `soap:Body`-elementin sisällöstä tulee WSDL-määrityksistä. Vaikka tyyli on laillista WSDL-kieltä, se ei ole WS-I:n määritystä noudattava.

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>

<!-- Loput määritykset ohitetaan tässä tapauksessa.
     Oletetaan niiden olevan RPC/encoded-tyyppisiä   -->
```

```
SOAP-viesti:

<soap:Envelope>
    <soap:Body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:Body>
</soap:Envelope>
```

Kuva 13. Tyypin RPC/encoded WSDL-määritys ja SOAP-viesti (Butek, 2005).

2.6.2 *RPC/literal*

Tyylin RPC/literal viesti on saman muotoinen kuin RPC/encoded ja on yhtä yksinkertainen. Erona on se, että luotava SOAP-viesti ei sisällä tyypin kooditusinformaatiota (Richards, 2006). Kuvassa 14 on esitetty tyypin RPC/literal WSDL-määritys ja sen mukainen SOAP-viesti.

Vahvuutena tässäkin tyypissä on se, että WSDL-määritys on erittäin suoraviivainen ja operaation nimi sisältyy viestiin, joten vastaanottajan on helppo poimia viestistä operaation suorittamiseen vaadittavat tiedot (Butek, 2005). Tyypin kooditusinformaatio jätetään pois SOAP-viestistä, joten se tekee viestistä helpommin tulkittavan. Vahvuutena voidaan pitää myös sitä, että tämä tyyppi noudattaa WS-I:n määrityksiä. Tässäkin tyypissä heikkoutena

voidaan pitää sitä, että viestin validointi saataa olla hankalaa, koska vain rivit `<x ...>5</x>` ja `<y ...>5.0</y>` sisältävät skeemassa (`http://www.w3.org/2001/XMLSchema`) määriteltyjä asioita.

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/%gt;
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- Loput määritykset ohitetaan tässä tapauksessa.
     Oletetaan niiden olevan RPC/literal-tyyppisiä   -->
```

```
SOAP-viesti:

<soap:Envelope>
    <soap:Body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:Body>
</soap:Envelope>
```

Kuva 14. Tyypin RPC/literal WSDL-määritys ja SOAP-viesti (Butek, 2005).

2.6.3 *Document/literal*

Document-tyyli on osittain samanlainen kuin RPC-tyyli `message`-elementin määrittelyn osalta sillä poikkeuksella, että `part`-elementti käyttää `element`-attribuuttia. Useimmissa tapauksissa document/literal-tyyppiä käyttävä WSDL-dokumentti sisältää vain yhden `part`-elementin yhtä `message`-elementissä määritettyä viestiä kohden riippumatta siitä, kuinka monta parametria funktio vaatii. Syy tähän on se, että document/literal-tyyli ei muodosta parametreja sisältävää

operaation nimen mukaan nimettyä XML-elementtiä, vaan nimi otetaan operaation nimen sijaan `part`-elementin `element`-attribuutista. Sen sijaan, että SOAP-viesti sisältäisi yhden elementin, joka sisältää parametrit ja operaation nimen, käytetään elementtiä jokaiselle `message`-määrityksessä määritellylle `part`-elementille (Richards, 2006). Kuvassa 15 on esitetty tyypin document/literal WSDL-määritys ja sen mukainen SOAP-viesti.

```
<types>
    <schema>
        <element name="xElement" type="xsd:int"/>
        <element name="yElement" type="xsd:float"/>
    </schema>
</types>

<message name="myMethodRequest">
    <part name="x" element="xElement"/>
    <part name="y" element="yElement"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- Loput määritykset ohitetaan tässä tapauksessa.
     Oletetaan niiden olevan Document/literal-tyyppisiä   -->
```

```
SOAP-viesti:

<soap:Envelope>
    <soap:Body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </soap:Body>
</soap:Envelope>
```

Kuva 15. Tyypin document/literal WSDL-määritys ja SOAP-viesti (Butek, 2005).

Vahvuutena tässä tyypissä on se, että SOAP-viestiin ei viedä kooditusinformaatiota (Butek, 2005). Viesti voidaan validoida XML-validaattorilla, koska kaikki `soap:Body`-elementin

sisältämä tieto on määritelty skeemassa. Haittana tämän tyypin käytössä on se, että WSDL-dokumentista tulee hieman vaikeaselkoinen. Tämä tyyppi noudattaa WS-I:n määrityksiä muutamin poikkeuksin. WS-I:n määritys sallii vain yhden lapsielementin `soap:Body`-elementille, mikä rajoittaa tämän tyypin hyödyntämistä. Operaation nimeä ei sisällytetä SOAP-viestiin, mikä voi haitata tiedon käsittelyä. Tämä ongelma voidaan ratkaista käyttämällä käärittyä document/literal-tyyppiä.

2.6.4 *Kääritty document/literal*

Kärityn document/literal-tyypin SOAP-viesti näyttää lähes samanlaiselta kuin RPC/literal-tyypin SOAP-viesti. Näissä on kuitenki pieni ero. RPC/literal-tyyppisessä viestissä `soap:Body`-elementin lapsielementti `myMethod` oli saman niminen kuin operaatio. Kärityn document/literal-tyypin SOAP-viestissä `myMethod` on kääre-elementin nimenä, eli syöte-elementti on saman niminen kuin kutsuttava operaatio. Tällä tavalla saadaan sisällytettyä operaation nimi SOAP-viestiin. Tälle tyypille ei ole virallista määritystä, mutta se noudattaa WS-I:n määrityksiä ja sitä käytetään yleisesti (Butek, 2005). Usein tämän tyypin käytön yhteydessä puhutaan pelkästään document/literal-tyypistä. Eli käytännössä kääritty document/literal-tyyppi on tapa hyödyntää document/literal-tyyppistä määritystapaa. Kuvassa 16 on esitetty kärityn document/literal-tyypin WSDL-määritys ja sen mukainen SOAP-viesti.

Vahvuutena tässäkin tyypissä on se, että SOAP-viestiin ei viedä kooditusinformaatiota. Viesti voidaan validoida XML-validaattorilla, koska kaikki `soap:Body`-elementin sisältämä tieto on määritelty skeemassa. Tämä tyyppi noudattaa WS-I:n määrityksiä poikkeuksetta. Haittana on se, että WSDL-dokumentista tulee vielä monimutkaisempi kuin tavallisen document/literal-tyypin tapauksessa.

```
<types>
    <schema>
        <element name="myMethod">
            <complexType>
                <sequence>
                    <element name="x" type="xsd:int"/>
                    <element name="y" type="xsd:float"/>
                </sequence>
            </complexType>
        </element>
        <element name="myMethodResponse">
            <complexType/>
        </element>
    </schema>
</types>
<message name="myMethodRequest">
    <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
    <part name="parameters" element="myMethodResponse"/>
</message>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- Loput määritykset ohitetaan tässä tapauksessa.
     Oletetaan niiden olevan Document/literal-tyyppisiä   -->
```

```
SOAP-viesti:

<soap:Envelope>
    <soap:Body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:Body>
</soap:Envelope>
```

Kuva 16. Kädityn document/literal-tyypin WSDL-määritys ja SOAP-viesti (Butek, 2005).

3 SOAP-protokolla

SOAP on protokolla, joka mahdollistaa informaation vaihtamisen hajautetussa ympäristössä. Se on XML-perustainen protokolla, joka koostuu kolmesta osasta, jotka ovat kuorielementti, kooditussäännöt, sekä ohjeet etäproseduurikutsujen ja vastauksien kanssa toimimiseen. Kuorielementti määrittää sen, mitä viesti sisältää ja kuinka sitä tulisi käsitellä. Kooditussäännöt ilmaisevat ohjelmallisesti määriteltyjen tietotyyppien ilmentymiä (W3C, 2000). SOAP:ia voidaan käyttää yhdessä monen muun protokollan kanssa, mutta yleisesti sitä käytetään yhdessä HTTP-protokollan kanssa (Richards, 2006).

SOAP ei määrittele minkäänlaista semantiikkaa, kuten ohjelmointimallia tai toteutuksen tarkkuudelle menevää mallia, vaan se määrittää yksinkertaisen mekanismin ohjelman semantiikan ilmaisemiseen tarjoamalla modulaarisen pakkausmallin ja koodausmekanismin, jolla data voidaan koodata moduuleihin. Tämä mahdollistaa SOAP:in käytön monenlaisissa systeemeissä yksinkertaisen viestin välityksestä etäproseduurikutsuihin (W3C, 2000). Uusin SOAP-versio on 1.2, mutta tässä tutkielmassa käsitellään versiota 1.1, koska se on yhä eniten käytetty versio ja se noudattaa WS-I:n perusprofiilia, joka tarjoaa yhteentoimivuusopastusta web-palvelujen käyttöön (Richards, 2006; WS-I, 2006).

Edellisesssä luvussa kerrottiin WSDL-dokumentin muodostuksesta. Kuvissa 17 ja 18 olevat esimerkit ovat edellä läpikäydyn, liitteestä 1 löytyvän WSDL-dokumentin määrityksiä vastaavia SOAP-viestejä, jotka perustuvat pyyntöön ja vastaukseen kutsuttaessa `getPeopleByFirstLastName`-metodia. Esimerkeissä esiintyvät merkinnät j* ja * ovat hakukriteerejä henkilötietojen hakemiseksi.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1="urn:ExampleAPI">
    <SOAP-ENV:Body>
      <ns1:getPeopleByFirstLastName>
        <first>j*</first>
        <last>*</last>
      </ns1:getPeopleByFirstLastName>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Kuva 17. SOAP-pyyntö (Richards, 2006).

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
      <result>
        <SOAP-ENC:Struct>
          <id>1</id>
          <firstName>John</firstName>
          <lastName>Smith</lastName>
        </SOAP-ENC:Struct>
        <SOAP-ENC:Struct>
          <id>2</id>
          <firstName>Jane</firstName>
          <lastName>Doe</lastName>
        </SOAP-ENC:Struct>
      </result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Kuva 18. SOAP-vastaus (Richards, 2006).

3.1 SOAP-viestien rakenne

SOAP-viesti sisältää kuorielementin, joka koostuu otsikosta (valinnainen) ja rungosta:

```
<Envelope>
  <Header>...</Header>?
  <Body>...</Body>
</Envelope>
```

Etuliitteet ja nimiavaruudet taulukossa 1 ovat sellaisia, joita saatetaan kohdata SOAP-viesteissä. Etuliite `ns*` tarkoittaa mitä tahansa liitettä.

Taulukko 1. Tyypillisiä SOAP-viesteissä käytettäviä etuliitteitä (Richards, 2006).

Etuliite	**Kuvaus**
`SOAP-ENV`	`http://schemas.xmlsoap.org/soap/envelope/`
`SOAP-ENC`	`http://schemas.xmlsoap.org/soap/encoding/`
`xsd`	`http://www.w3.org/2001/XMLSchema`
`ns*`	Etuliite ja nimiavaruus-URI:t ovat sovelluksesta riippumattomia

3.1.1 *Kooditustyyli*

Globaalia attribuuttia `encodingStyle` voidaan käyttää niiden sarjallistussääntöjen ilmaisemiseksi, joita SOAP-viestissä käytetään. Tämä attribuutti saattaa ilmetä missä tahansa elementissä. SOAP-viesteille ei ole määritelty oletukseksi mitään kooditustyyliä (W3C, 2000).

Kun tätä attribuuttia käytetään, se sijaitsee `SOAP-ENV`-nimiavaruudessa ja normaalisti se ottaa arvon nimiavaruudesta `http://schemas.xmlsoap.org/soap/encoding/` (Richards, 2006).

3.1.2 *Kuori*

Jokaisen SOAP-viestin tulee sisältää `Envelope`-elementti, joka sijaitsee `SOAP-ENV`-nimiavaruudessa. Tämä nimiavaruus identifioi viestin SOAP 1.1 -viestiksi. Yksinkertaisimmillaan kuorielementti voi olla seuraavanlaisessa muodossa (Richards, 2006):

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <!-- Lapsielementit -->
</SOAP-ENV:Envelope>
```

3.1.3 *Otsikko*

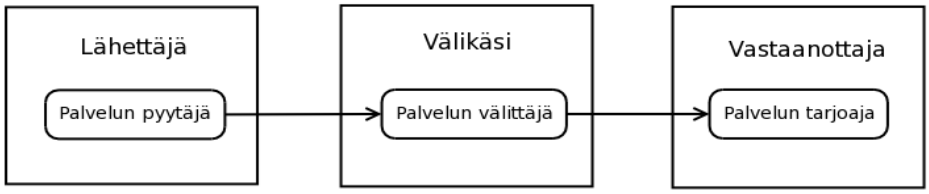
Otsikkoelementti `Header` ei ole pakollinen. Se tarjoaa joustavan mekanismin laajentaa viestejä hajautetusti ja modulaarisesti ilman, että kommunikoivista osapuolista tarvitsisi tietää etukäteen yhtään mitään. Kun tätä elementtiä käytetään, sen tulee olla ensimmäisenä lapsielementtinä kuorielementissä. Kaikkiin `Header`-elementin välittömiin lapsielementteihin viitataan nimellä *sisääntuloelementti* (header entry) (W3C, 2000). Esimerkki `Header`-elementistä:

```
<SOAP-ENV:Header>
  <t:Transaction xmlns:t="http://www.example.com">
    <!-- Käsiteltävä tieto -->
  </t:Transaction>
</SOAP-ENV:Header>
```

3.1.4 *Toimija*

Aina ei ole itsestään selvää, että kaikki SOAP-viestin osat on tarkoitettu välitettäväksi ensimmäiselle vastaanottajalle, sillä matkan varrella voi olla yksi tai useampia välikäsiä. Termi välikäsi tarkoittaa tässä yhteydessä sovellusta, joka vastaanottaa ja lähettää eteenpäin SOAP-viestejä. SOAP-viesti voi sisältää osia, jotka ovat määritelty vain tietyille välittäjille, eikä vain lopullliselle kohteelle. Sekä välittäjät, että lopulliset vastaanottajat identifioidaan URI:n avulla (W3C, 2000).

Kuvassa 19 on kuvattu kolme eri tyyppistä SOAP-palvelua: lähettäjä, välikäsi ja vastaanottaja. SOAP-palvelu voi voi vaihtaa rooliaan riippuen vallitsevasta prosessin tilasta ja sen sijoittumisesta viestin kulkemalle reitille. Esimerkiksi viestin lähettävä palvelu voi myöhemmin toimia myös viestin lopullisena vastaanottajana (Erl, 2004).



Kuva 19. SOAP-palveluiden rooleja.

Attribuuttia `actor` voidaan käyttää `Header`-elementin vastaanottajan osoittamiseen. Se sijaitsee `SOAP-ENV`-nimiavaruudessa ja saa arvokseen nimiavaruuden, joka identifioi viestin vastaanottajan. Kun arvo on tyhjä tai attribuuttia ei käytetä, osoite suunnataan lopulliselle vastaanottajalle. Kun `Header`-elementin sisääntuloelementti on prosessoitu, se poistetaan ennenkuin viesti lähetetään eteenpäin. Syynä tähän on se, että tietylle vastaanottajalle tarkoitettu sisältö käsitellään sopimuksena kutsun luojan ja vastaanottajan välillä, eikä ole vastaanoton jälkeen enää validi (Richards, 2006).

Esimerkkinä viestin kulkemisesta välikäsien kautta voidaan pitää seuraavanlaista polkua, jonka SOAP-viesti kulkee (Richards, 2006): Palvelin A => Palvelin B => Palvelin C. Palvelin A on viestin aloituspiste ja palvelin C on viestin lopullinen vastaanottaja. Palvelimen B identifioi `urn:PalvelinB`-nimiavaruus ja palvelimen C identifioi `urn:PalvelinC`-nimiavaruus. Kuvan 20 mukaisella otsikolla palvelin B vastaanottaa SOAP-viestin. Se tietää, että viestille pitää tehdä jotakin, koska se on merkitty `actor`-attribuutilla vastaanottajaksi. Riippumatta siitä mitä operaatio tekee, poistetaan tämän `actor`-parametrin sisältö, kun viesti vastaanotetaan. Tämän esimerkin tapauksessa palvelin B ei lisää uutta arvoa, joten tyhjän sisääntuloelementin sisältämä viesti lähetetään eteenpäin palvelimelle C, joka on viestin lopullinen vastaanottaja (Richards, 2006).

```
<SOAP-ENV:Header>
  <t:Transaction xmlns:t = "http://www.example.com"
    SOAP-ENV:actor = "urn:PalvelinB">
      <!-- Käsiteltävä tieto -->
  </t:Transaction>
</SOAP-ENV:Header>
```

Kuva 20. Esimerkki `actor`-attribuutin käytöstä vastaanottajan määrittämiseksi.

Attribuutin `actor` arvoksi voidaan määritellä myös `http://schemas.xmlsoap.org/soap/actor/next`. Tämä osoittaa sen, että `Header`-elementti on osoitettu ensimmäiselle SOAP-sovellukselle, joka viestiä käsittelee (W3C, 2000).

3.1.5 *Ymmärrettävyys*

Attribuuttia `mustUnderstand` voidaan käyttää antamaan ohjeita siitä, mitä pitäisi tehdä, jos sisääntuloelementtiä ei voida prosessoida oikein. Tämä attribuutti ottaa arvon 1 tai 0 (oletus). Kun arvoa 1 käytetään ja vastaanottaja ei pysty prosessoimaan sisältöä täysin, sitä ohjeistetaan lopettamaan prosessointi ja tekemään SOAP-virheilmoitus. Kun attribuutti jätetään pois tai sen arvoksi asetetaan 0, sovellus voi ohittaa virheen ja jatkaa viestin prosessointia (Richards, 2006). Kuva 21 havainnollistaa attribuutin käyttöä.

```
<SOAP-ENV:Header>
  <t:Transaction xmlns:t="urn:PalvelinB"
    SOAP ENV:actor="urn:PalvelinB"
    SOAP-ENC:mustUnderstand="1">
      <!-- Käsiteltävä tieto -->
  </t:Transaction>
</SOAP-ENV:Header>
```

Kuva 21. Esimerkki `mustUnderstand`-attribuutin käytöstä.

3.1.6 *Runko*

Runkoelementti `Body` sisältää informaatiota, joka on tarkoitettu viestin lopulliselle vastaanottajalle. Informaation rakenne riippuu WSDL-dokumenttia luotaessa valitusta viestin rakenteesta. Voidaan valita `rpc/encoded`, `rpc/literal` tai `document/literal`. Käytetty metodi määrittää `Body`-elementin sisällön rakenteen. Jos käytetään otsikkoa (`Header`), runkoelementti tulee heti tämän jälkeen. Muuten runkoelementti on ensimmäinen `Envelope`-elementin lapsielementti (Richards, 2006). Kaksi esimerkkiä runkoelementistä on esitelty kuvassa 22.

```
<!-- Body-elementti, jota ennen Header-elementti -->
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
      <!-- Data -->
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <!-- Data -->
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

<!-- Body-elementti ilman Header-elementtiä -->
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
      <!-- Data -->
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Kuva 22. Esimerkkejä `Body`-elementin käytöstä (Richards, 2006).

Vaikka `Header`- ja `Body`-elementit määritellään itsenäisinä elementteinä, ne liittyvät toisiinsa. Suhde näiden sisääntuloelementtien välillä on seuraavanlainen: `Body`-elementin sisääntuloelementti on semanttisesti samanlainen kuin `actor`-parametrin oletusarvon (arvo on tyhjä tai attribuuttia ei käytetä) ja `mustUnderstand`-parametrin arvon 1 sisältävän `Header`-elementin sisääntuloelementti (W3C, 2000).

3.2 Virhetilanteet

Virhetilanteiden hoitamiseksi pitää käyttää `Fault`-elementtiä, jolla voidaan välittää virhetietoja SOAP-viestin mukana. Jos elementti on mukana, tulee sen olla `Body`-elementin sisääntuloelementtinä ja se voi esiintyä vain kerran (W3C, 2000). SOAP:in `Fault`-elementti määrittää neljä lapsielementtiä, joita käsitellään seuraavaksi.

3.2.1 *Virhekoodi*

Elementti `faultcode` on tarkoitettu käytettäväksi sovellukselle, joka tunnistaa havaitun virheen ohjelmallisesti. Elementin `faultcode` tulee sisältyä SOAP:in `Fault`-elementtiin ja sen arvon tulee olla määrätty nimi. SOAP-määrityksiin kuuluu muutamia virhekoodeja (taulukko 2), jotka kattavat yleisimmät virheet (W3C, 2000).

Taulukko 2. SOAP-määrityksiin kuuluvia virhekoodeja.

Nimi	**Tarkoitus**
`VersionMismatch`	Prosessoiva osapuoli on löytänyt epäkelvon nimiavaruuden SOAP:in `Envelope`-elementille.
`MustUnderstand`	Sisääntuloelementin, joka sisältää `mustUnderstand`-attribuutin arvolla 1, käsittely tai ymmärtäminen epäonnistui.
`Client`	SOAP-viesti on väärin muodostettu tai ei sisällä sopivaa tietoa.
`Server`	SOAP-viestiä ei voida prosessoida syistä, jotka eivät ole suoraan määritettävissä attribuuteilla viestissä itsessään, vaan ennemminkin viestin prosessoinnissa.

Elementti `Fault` seuraavassa esimerkissä osoittaa, että asiakkaan lähettämässä SOAP-viestissä oli ongelma:

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Client</faultcode>
  <!-- Lisäksi muita Fault-lapsielementtejä -->
</SOAP-ENV:Fault>
```

Virhekoodi voi olla myös rakenteinen. Käyttämällä pistettä (.) eroittimena, voidaan välittää useampia riippuvuuksia omaavia koodeja. Oletetaan esimerkiksi, että täytyy suorittaa autentikointi. Tämä tieto voidaan välittää SOAP `Header`-entiteetissä. Sovellus, joka vastaanottaa viestin ilman tätä entiteettiä, saattaa haluta palauttaa autentikointivirheen. Tämä voitaisiin rinnastaa SOAP:in `Client`-virheeseen, koska asiakas ei välittänyt sopivaa rakennetta (Richards, 2006):

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Client.Autentikointi</faultcode>
  <!-- Lisäksi muita Fault-lapsielementtejä -->
</SOAP-ENV:Fault>
```

3.2.2 *Virheilmoitus*

Elementti `faultstring` tarjoaa selkokielisen kuvauksen ilmenneestä virheestä, eikä ole tarkoitettu ohjelmallisesti prosessoitavaksi. Tämän elementin täytyy sisältyä SOAP:in `Fault`-elementtiin ja sen tulisi tarjota jotain informaatiota, joka kuvaa virheen laatua (W3C, 2000). Esimerkki `faultstring`-elementistä:

```
<SOAP-ENV:Fault>
  <faultstring>Autentikointi-informaatio puuttuu</faultstring>
  <!-- Lisäksi muita Fault-lapsielementtejä -->
</SOAP-ENV:Fault>
```

3.2.3 *Virheen aiheuttaja*

Elementti `faultactor` osoittaa virheen aiheuttajan. Tämä elementti on melko samanlainen kuin `actor`-attribuuttti `Header`-elementin sisääntuloattribuutissa, mutta sen sijaan että se osoittaisi sisääntuloelementin kohteen, se osoittaa virheen alkuperän. Arvo on URI, joka identifioi virheen alkuperän (W3C, 2000). Esimerkki `faultactor`-elementistä:

```
<SOAP-ENV:Fault>
  <faultactor>urn:PalvelinB</faultstring>
  <!-- Lisäksi muita Fault-lapsielementtejä -->
</SOAP-ENV:Fault>
```

3.2.4 *Yksityiskohdat*

Elementti `detail` sisältää sovelluskohtaista informaatiota, joka liittyy SOAP-viestin `Body`-elementtiin, joten sen rakenne on sovellusriippuvainen. Tämän elementin täytyy olla mukana, jos `Body`-elementin sisältöä ei voida prosessoida kunnolla. Jos virhe ei liity `Body`-elementin prosessointiin, niin `detail`-elementtiä ei tule sisällyttää `Fault`-elementin rakenteeseen (W3C, 2000).

Elementti `detail` sisältää itsenäisiä lapsielementtejä, joita kutsutaan nimellä sisääntuloelementti (detail entry). Kuvassa 23 on esimerkki `detail`-elementin määrityksestä, joka sisältää yhden lapsielementin.

```
<SOAP-ENV:Fault>
  <faultcode>ns1:DBError</faultcode>
  <faultstring>Tietokantavirhe havaittu</faultstring>
  <detail>
    <ns1:DBUnavailableFault>
      <DBMessage>Tietokantaan yhdistäminen epäonnistui</DBMessage>
      <RetryInMinutes>60</RetryInMinutes>
    </ns1:DBUnavailableFault>
  </detail>
</SOAP-ENV:Fault>
```

Kuva 23. Esimerkki `detail`-elementin määrityksestä (Richards, 2006).

4 SOAP-protokollan käyttö PHP-ohjelmoinnissa

PHP 5 -version julkaisu toi mukanaan paljon uudistuksia XML-tiedon käsittelyyn ja mahdollisti siten monien XML-pohjaisten teknologioiden käytön. Natiivi tuki SOAP-protokollalle oli yksi näistä uudistuksista. PHP:n SOAP-laajennus tarjoaa tuen SOAP 1.1 (W3C, 2000) ja SOAP 1.2 (W3C, 2007a) -versioille, sekä WSDL 1.1 (W3C, 2001) -määrityksille (PHP Group, 2007b). Tässä tutkielmassa keskitytään SOAP 1.1 -versioon.

On myös toteutettu ulkoisia kirjastoja, jotka mahdollistavat SOAP-protokollan käytön PHP-ohjelmoinnissa. Ulkoisille kirjastoille on käyttöä esimerkiksi silloin, kun web-palvelimella oleva PHP:n versio ei sisällä natiivia tukea SOAP-protokollalle. Tässä luvussa kerrotaan ja demonstroidaan SOAP-protokollan käyttöä PHP-ohjelmoinnissa Richardsin (2006) ja PHP-manuaalin (PHP Group, 2007b) mukaisesti. Liitteessä 1 on esitelty WSDL-tiedosto, jota tarvitaan liitteen 2 esimerkkisovelluksessa. Liitteen 2 esimerkkisovelluksena on yksinkertainen asiakas-palvelin-sovellus, jonka asiakasosaa tarkastellaan kohdassa 4.2 ja palvelinosaa kohdassa 4.3.

4.1 Yleiset SOAP-luokat

SOAP-laajennuksen yleisiä luokkia käytetään silloin kun kirjoitetaan asiakas- ja palvelinsovelluksia. Ennen SOAP:in kanssa työskentelyä kannattaa opetella tuntemaan nämä luokat, sekä tietää mitä ne tekevät ja miten ne luodaan. Vaikka kaikkia luokkia ei tarvittaisikaan, muutamat niistä voivat osoittautua hyödylliseksi, kun lähetetään tai vastaanotetaan dataa käyttäen SOAP-protokollaa.

4.1.1 *SoapVar*

Jos työskennellään ilman WSDL:ää, on SOAP:in käyttö astetta vaikeampaa. Asiakassovellukset ja palvelimet eivät osaa automaattisesti määrittää oikeita tietotyyppejä, kun taas ne sovellukset, jotka käyttävät WSDL:ää, osaavat tehdä tämän. Kun käytetään `SoapVar`-oliota, voidaan määrittää tietotyyppi ja haluttaessa voidaan määrittää viestiä luotaessa käytettävän elementin nimi. Luokka `SoapVar` on käytännöllinen myös silloin, kun työskennellään SOAP-viestien erilaisten aspektien, kuten esimerkiksi `Header`-elementin kanssa. Luokka `SoapVar` koostuu pelkästään konstruktorista:

```
__construct(mixed data, int encoding [, string type_name [,
  string type_namespace [,string node_name [, string node_namespace]]]])
```

Konstruktori ottaa vastaan useita parametreja, joista jokainen vaikuttaa siihen, miten tieto sarjallistetaan SOAP-viestiä luotaessa. Taulukossa 3 on kuvattu konstruktorille välitettäviä parametreja.

Taulukko 3. Luokan `SoapVar` konstruktorille välitettäviä parametreja (Richards, 2006).

Parametri	**Kuvaus**
`data`	Tieto, joka välitetään SOAP-viestissä, välitetään tällä parametrilla.
`encoding`	Osoittaa välitettävän tiedon tyypin. Tämän parametrin arvo on yksi SOAP-koodituksen vakioista (esim. `rpc/encoded`).
`type_name`	Tiedon tyypin nimi. Kun käytetään tyyppiä `rpc/encoded`, `type_name` on attribuutin `xsi:type` arvo välitettävässä SOAP-viestissä.
`type_namespace`	Nimiavaruus, jossa `type_name`-parametrilla määritelty parametri sijaitsee.
`node_name`	SOAP-viestissä käytettävän elementin nimi.
`node_namespace`	Nimiavaruus, jossa `node_name` parametri sijaitsee.

Kuvan 24 esimerkki SoapVar-luokan käytöstä luo SOAP:in Body-elementin, kun ilmentymän luomiseen ei käytetä WSDL-dokumenttia, vaan se luodaan täysin ohjelmallisesti. Kuvan 24 koodi luo SoapVar-luokan ilmentymän, joka määrittelee saman rakenteen osan, mikä ilmenee myös kuvan 17 SOAP-viestistä.

```
/* Määritetään arvot ja rakenne getPeopleByFirstLastName tyypille */
class getPeopleByFirstLastName {
    public $first = 'j*';
    public $last = '*';
}

$PeopleStruct = new getPeopleByFirstLastName();

/*
    Luodaan uusi SoapVar-ilmentymä käyttämällä liitteen 1
    WSDL-tiedostossa määriteltyjen tyyppien mukaisia tyyppejä.
*/
$PeopleVar = new SoapVar($PeopleStruct, SOAP_ENC_OBJECT,
                         "getPeopleByFirstLastName","urn:ExampleAPI",
                         "getPeopleByFirstLastName","urn:ExampleAPI");
```

Kuva 24. Esimerkki SoapVar-luokan käytöstä (Richards, 2006).

Käytettävä data on getPeopleByFirstLastName-olio ja se on vastaava liitteen 1 WSDL-dokumentin kanssa, koska siinä on first- ja last-osat. Olio on koodattu SOAP_ENC_OBJECT-muotoon. Ennalta määrätty vakio SOAP_ENC_OBJECT ilmaisee sen, että SoapVar-ilmentymään sisällytettävä data on tyypiltään olio. Olion tyyppi on getPeopleByFirstLastName, joka sijaitsee urn:ExampleAPI-nimiavaruudessa ja se noudattaa liitteen 1 WSDL-dokumentissa määriteltyä monimutkaisen tyypin määritystä. Viimeisenä jäsenet kääritään getPeopleBy-FirstLastName-elementtiin, joka sijaitsee SOAP-viestin urn:ExampleAPI-nimiavaruudessa, joten nämä välitetään node_name- ja node_namespace-parametrien välityksellä.

Luokan SoapVar käytölle ei ole tarvetta, jos käytetään WSDL-tiedostoa, mutta tämän esimerkin tarkoituksena on osoittaa SoapVar-parametrien ja WSDL-määrityksien välinen suhde. Jos WSDL-tiedostoa olisi käytetty, oltaisiin voitu käytää suoraan $PeopleStruct-oliota.

4.1.2 *SoapHeader*

Luokka `SoapHeader` mahdollistaa SOAP:in `Header`-entiteettien luomisen, jotka mahdollistavat SOAP-viestien modulaarisen laajennettavuuden. Luokkaa voidaan käyttää WSDL-dokumentin kanssa tai ilman sitä. Luokka koostuu pelkästään konstruktorista:

```
__construct(string namespace, string name [, mixed data [,
            bool mustUnderstand [,mixed actor]]])
```

Konstruktori ottaa vastaan useita parametreja, jotka ovat rinnastettavissa lähes suoraan `Header`-entiteetin rakenteeseen. Parametrit `namespace` ja `name` määrittävät `Header`-entiteettiä varten luodun elementin. Parametria `data`, joka on valinnainen, käytetään `Header`-entiteetin sisällölle. Se voi olla PHP-tyyppiä tai kompleksinen tyyppi, joka on luotu `SoapVar`-oliota käyttämällä. Valinnainen `mustUnderstand`-parametri asettaa `mustUnderstand`-attribuutin `Header`-entiteetille. Jos tämä jätetään pois tai välitetään arvo FALSE, attribuuttia ei lisätä. Valinnainen `actor`-parametri asettaa arvon `Header`-entiteetin `actor`-attribuutille. Kun tätä käytetään, tulee sen arvon olla joko URI:n sisältävä merkkijono tai yksi `SOAP_ACTOR`-vakioista, joita ovat `SOAP_ACTOR_NEXT`, `SOAP_ACTOR_NONE` ja `SOAP_ACTOR_UNLIMITERECEIVER`.

Luodaan esimerkkinä `SoapHeader`-luokka käyttäen samanlaista `Header`-elementtiä kuin SOAP-otsikkoelementtiä käsittelevässä kohdassa 3.1.3. Luodaan `Transaction`-elementti, joka sijaitsee `http://www.example.com`-nimiavaruudessa. Se määrittää `mustUnderstand`-attribuutin arvoksi `TRUE` sekä asettaa `actor`-attribuutiksi SOAP-viestin seuraavan vastaanottajan:

```
$soapHeader = new SoapHeader("http://www.example.com", "Transaction",
                             "Sisältö", TRUE, SOAP_ACTOR_NEXT);
```

Kun viesti on sarjallistettu, `Header`-elementti näyttää seuraavanlaiselta, kun oletetaan että nimiavaruus `http://www.example.com` on liitetty etuliitteeseen `ns2`:

```
<SOAP-ENV:Header>
  <ns2:Transaction SOAP-ENV:mustUnderstand="1"
    SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
      Sisältö
  </ns2:Transaction>
</SOAP-ENV:Header>
```

4.1.3 *SoapParam*

Luokka `SoapParam` mahdollistaa nimettyjen parametrien luomisen. Kuten muutkin yleiset luokat, `SoapParam` koostuu pelkästään konstruktorista, joka yksinkertaisesti ottaa vastaan datan, joka tulee assosioida johonkin nimeen:

```
__construct (mixed data, string name)
```

Parametri `data` voi saada arvon, joka sisältää joko natiivin PHP-tyyppisen tiedon tai `SoapVar`-luokan ilmentymän. Parametri `name` on merkkijono, joka sisältää parametrille käytettävän nimen. Käyttämällä `$PeopleStruct`-oliota, joka luotiin kuvassa 24, voidaan luoda `getPeopleByFirstLastName`-niminen parametri, joka sisältää olion tiedot:

```
$PeopleVar = new SoapVar($PeopleStruct, SOAP_ENC_OBJECT,
                         "getPeopleByFirstLastName","urn:ExampleAPI",
                         "getPeopleByFirstLastName","urn:ExampleAPI");

$PeopleParam = new SoapParam($PeopleVar, 'getPeopleByFirstLastName');
```

4.1.4 *SoapFault*

Asiakassovellus käyttää `SoapFault`-luokkaa, kun SOAP-virhe vastaanotetaan. Palvelin käyttää tätä luokkaa luomaan virheilmoitusviestin, joka palautetaan asiakassovellukselle. Asiakassovellukselle `SoapFault`-olion rakenne on tärkeä. Luokan `SoapFault` ilmentymää käsitellään joko poikkeuksena (oletus), tai sitten se välitetään palautusarvona funktiota kutsuttaessa. Molemmissa tapauksissa olion rakenne on sama. Se sisältää useita parametrejä, jotka informoivat virheestä. Käytetyt parametrit ovat `faultcode`, `faultstring`, `faultactor` ja `detail`, joiden sisältö on määritelty kohdassa 3.2.

Palvelimen perspektiivistä katsottuna `SoapFault`-luokka luo SOAP-virheitä, eikä niinkään käsittele niitä. Virhe luodaan kutsumalla konstruktoria:

```
__construct(string faultcode, string faultstring [, string faultactor [,
            mixed detail [, string faultname [, SoapHeader headerfault]]]])
```

Ensimmäiset kolme parametriä ovat ilmeisiä viitaten kohdassa 3.2 käsiteltyihin tietoihin SOAP-virheistä. Parametri `detail` voi olla tyypiltään mikä tahansa. Tämä osa `Fault`-elementistä on sovelluksesta riippuva, joten voidaan välittää kompleksisia rakenteita tarjoamaan asiakassovelluksen tarvitsemia tietoja. WSDL-dokumentissa määritellään virheen nimi. Parametri `faultname` yhdistää `SoapFault`-ilmentymän nimettyyn virheeseen. Välittämällä tämän parametrin, asiakassovellus tietää mitä odottaa, sekä mahdollistaa välitettävän tiedon kunnollisen koodituksen (encoding). Parametria `headerfault` käytetään, kun virhe havaitaan `Header`-entiteetin prosessoinnin aikana. Se mahdollistaa `soap:headerfault`-attribuutin palauttamisen, joka on virheviesti vastausviestin `Header`-elementissä. Esimerkki `SoapFault`-ilmentymän luomisesta:

```
new SoapFault("DBError", "Virhe Header-elementissä", "urn:ExampleAPI",
              "Virhe: Yksinkertaisen merkkojonon yksityiskohdat", "nodb");
```

4.2 Asiakas

Luokka `SoapClient` on tärkeä työväline SOAP-palveluiden käytössä. Sitä käyttäen voi tehdä pyyntöjä SOAP-palvelimelle. Käytettäessä WSDL-pohjaista SOAP:ia, `SoapClient`-luokka osaa tehdä suurimman osan tietotyyppien muodostuksista ja koodituksista, joita tarvitaan muodostettaessa SOAP-viestejä PHP-muuttujista ja tyypeistä. Jos ei käytetä WSDL-pohjaista SOAP:ia, on tilanne hiukan hankalampi ja siksi tarvitaan edellä tarkasteltuja yleisiä luokkia. Seuraavaksi käsitellään erilaisia `SoapClient`-luokan toiminnallisuuksia.

4.2.1 *SoapClient-ilmentymän luominen*

Luokan `SoapClient` ilmentymä luodaan kutsumalla konstruktoria. Tässä vaiheessa voidaan määrittää monia asetuksia sekä se, käytetäänkö WSDL-tiedostoa vai ei. Konstruktori on seuraavanlainen:

```
__construct (mixed wsdl [, array options])
```

Parametri `wsdl` on tärkeä, koska se määrittää WSDL-tiedoston sijainnin. Jos tiedostoa ei käytetä, tulee arvoksi asettaa NULL. Taulukossa 4 on listattu asetukset, jotka voidaan välittää konstuktorille assosiatiivisena taulukkona. Kun ei käytetä WSDL-tiedostoa, tämä taulukko sekä siihen sisältyvät `location`- ja `uri`-tiedot vaaditaan. Toisin kuin WSDL-tiedostoa käytettäessä, näitä kahta asetusta ei voida automaattisesti määrittää, vaan ne täytyy itse asettaa.

Taulukossa 4 on kuvattu parametreja, joita konstuktorille voidaan syöttää ilmentymää luotaessa ja jotka tarjoavat joustavuutta monien asiakassovelluksen aspektien hallintaan.

Taulukko 4. `SoapClient`-ilmentymää luotaessa käytettäviä parametreja (Richards, 2006).

Asetus	**Kuvaus**
`location`	URL, johon kutsut web-palvelulle kohdistetaan. WSDL-moodissa, asetus kumoaa WSDL-tiedostossa määritellyn oletusosoitteen.
`uri`	SOAP-palvelimen kohdenimiavaruus (target namespace).
`style`	Viestien rakenteiden tyyli. Joko `SOAP_DOCUMENT` tai `SOAP_RPC`.
`use`	Käytettävä kooditus (`SOAP_ENCODED` tai `SOAP_LITERAL`).
`soap_version`	Määrittää sen, käytetäänkö SOAP 1.1 vai SOAP 1.2 versiota. Voi saada joko arvon `SOAP_1_1` tai `SOAP_1_2`.
`connection_timeout`	Enimmäisaika (s), joka odotetaan palvelimeen yhdistettäessä.
`stream_context`	Pyynnöissä käytettävän tietovirran tyyppi.
`login`	Käyttäjätunnus, jota käytetään HTTP-autentikoinnissa.
`password`	Salasana HTTP-autentikointia varten.
`authentication`	Asettaa autentikointityypin. Arvo `SOAP_AUTHENTICATION_DIGEST` asettaa asiakassovelluksen käyttämään `Digest`-autentikointia.
`proxy_host`	Proxy-palvelin, jonka kautta yhteys mahdollisesti muodostetaan.
`proxy_login`	Käyttäjätunnus proxy-palvelimelle
`proxy_password`	Salasana proxy-palvelimelle.
`local_cert`	HTTPS asiakassovelluksen autentikoinnissa käytettävä sertifikaatti.
`passphrase`	Kulkutunniste HTTPS-autentikointia varten.
`compression`	Käytettävä pakkausmenetelmä. Arvo on mikä tahansa yhdistelmä seuraavista: `SOAP_COMPRESSION_ACCEPT`, `SOAP_COMPRESSION_GZIP` ja `SOAP_COMPRESSION_DEFLATE` yhdistettynä käyttäen ”or”-operaattoria. Esimerkiksi: `SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP`.
`encoding`	Käytettävä kooditus merkkijonoja palautettaessa.
`classmap`	Sallii joidenkin WSDL-tyyppien muodostamisen PHP-luokiksi. Tyypiltään assosiatiivinen taulukko, jossa käytetään WSDL-tyyppejä ja avaimia, sekä PHP-luokkia arvoina.
`trace`	Sallii debuggauksen. Asettamalla tämän arvoksi 1, voidaan käyttää __getLastXXX()- metodeja.
`exceptions`	Oletuksena SOAP-virheet heitetään poikkeuksina. Asettamalla tälle arvo 0, otetaan virhepoikkeukset pois käytöstä.

Kuvassa 25 on esitetty muutamia mahdollisia esimerkkitapauksia `SoapClient`-ilmentymän muodostamisesta käyttäen WSDL-dokumenttia ja ilman sitä.

```
$client = new SoapClient("http://www.example.com/example.wsdl");

$client = new SoapClient("http://www.example.com/example.wsdl",
              array('login'=>"username", 'password'=>"password"));

$client = new SoapClient("http://www.example.com/example.wsdl",
              array('proxy_host'=>"localhost",'proxy_port'=> 8080));

$client = new SoapClient(null,
              array('location'=>"http://www.example.com/soap.php",
                    'uri'=>"http://www.example.com/"));

$client = new SoapClient(null,
              array('location'=>"http://www.example.com/soap.php",
                    'uri'=>"urn:ExampleAPI",
                    'style'=>SOAP_DOCUMENT,
                    'use'=>SOAP_LITERAL));

class cPerson {
   public $first;
   public $last;
}

$client = new SoapClient("http://www.example.com/example.wsdl",
              array('classmap' => array('Person' => "cPerson")));
```

Kuva 25. Esimerkkitapauksia `SoapClient`-ilmentymän muodostamisesta (Richards, 2006).

4.2.2 *Palvelun tarkastelu*

Eräs hyöty WSDL-pohjaisen SOAP:in käytöstä on se, että tietotyyppien ja funktioiden tarkastelu on helppoa. Ei tarvitse kahlata läpi pitkiä dokumentaatioita tai WSDL-tiedostoja, vaan niihin voidaan päästä käsiksi suoraan käyttämällä `__getTypes()` ja `__getFunctions()`-metodeja. Kuvan 26 esimerkki käyttää Richardsin (2006) esittämää muunnospalvelua, joka tarjoaa funktioita numeerisen syötteen muuntamiseksi sanalliseksi ilmaisuksi.

```
<?php
/* Asetetaan WSDL-tiedoston sijainti */
$wsdl='http://www.dataaccess.com/webservicesserver/conversions.wso?WSDL';

try {
    $xConverter = new SoapClient($wsdl);
    echo "Types:\n";
    if ($xTypes = $xConverter->__getTypes()) {
        foreach ($xTypes AS $type) {
            echo $type."\n\n";
        }
    }
    echo "Functions:\n";
    if ($xTypes = $xConverter->__getFunctions()) {
        foreach ($xTypes AS $type) {
            echo $type."\n\n";
        }
    }
} catch (SoapFault $e) {
    var_dump($e);
}
?>
```

Kuva 26. Esimerkki palvelun tarkastelusta (Richards, 2006).

Kuvassa 26 kumpikin hakumetodi palauttaa merkkijonotaulukon, joka sisältää joko tyyppi- tai funktiotunnuksen perustuen kutsuttuun metodiin. Luokan `SoapClient` ilmentymä luodaan käyttäen WSDL-tiedostoa ja se sijoitetaan `$xConverter`-muuttujaan. Asiakassovellus tekee kutsun kummallekin metodille ja käy läpi palautetut taulukot tulostaen niiden sisällön. Kuvassa 27 on esitelty tuloste, joka seuraa kuvan 26 sovelluksen suorituksesta.

Jos palvelu ei sisällä WSDL-tiedostoa, palvelun tarkastelua ei voida tehdä. Tällöin ainut tapa, jolla voi selvittää sen, mitä palvelu tarjoaa, on käydä läpi jonkinlainen dokumentaatio, jos sellainen on saatavilla.

```
Types:

struct NumberToWords {
  unsignedLong ubiNum;
}

struct NumberToWordsResponse {
  string NumberToWordsResult;
}

struct NumberToDollars {
  decimal dNum;
}

struct NumberToDollarsResponse {
  string NumberToDollarsResult;
}
struct TitleCaseWords {
  string sText;
  string sToken;
}

struct TitleCaseWordsResponse {
  string TitleCaseWordsResult;
}

Functions:

NumberToWordsResponse NumberToWords(NumberToWords $parameters)

NumberToDollarsResponse NumberToDollars(NumberToDollars $parameters)

TitleCaseWordsResponse TitleCaseWords(TitleCaseWords $parameters)
```

Kuva 27. Tuloste, joka seuraa kuvan 26 sovelluksen suorituksesta (Richards, 2006).

4.2.3 *Sijainti*

On täysin mahdollista, että palvelu ei ole saatavissa, jos vaikka palvelin on tilapäisesti pois käytöstä. Kun asiakassovellus tekee kutsun, tämä johtaa SOAP-virheeseen ja 404-virheeseen HTTP-vastauksena. Palvelu voi silti olla saatavilla toisissa osoitteissa. Esimerkiksi WSDL-dokumentissa palvelulla voi olla useita portteja, jotka jakavat saman porttityypin. Ainoa ero on `soap:address`-attribuutti, joka voi osoittaa, että vaihtoehtoisia osoitteita samalle palvelulle

on saatavilla. Luokan `SoapClient` ilmentymä ei automaattisesti yhdistä näihin osoitteisiin kun yhteysvirhe havaitaan.

Sijainnin muuttamiseksi on mahdollista luoda uusi `SoapClient`-ilmentymä käyttäen vaihtoehtoista osoitetta, joka määritellään sijaintitietojen perusteella. Tällöin tulee asettaa samat määritykset lukuunottamatta sijaintitietoa (`location`). Olemassaolevaa `SoapClient`-ilmentymää voidaan käyttää uudelleen ja muuttaa sijainti kutsumalla `__setLocation()`-metodia. Tälle metodille tulee antaa parametrina merkkijono, joka sisältää URL:n, joka osoittaa uuteen sijaintiin.

4.2.4 *Asiakaskutsujen tekeminen*

Kutsujen tekemiseen on olemassa monia tapoja. Tapa riippuu paljolti siitä, käytetäänkö WSDL-pohjaista SOAP:ia vai ei, sekä käytetystä tyylistä (`document/literal`, `rpc/literal` tai `rpc/encoded`). Tässä kohdassa käsitellään kutsumista kahdella eri tekniikalla: yksinkertaisen tyypin RPC-kooditettu kutsu (Simple Type RPC-Encoded Call) ja monimutkaisen tyypin dokumentin literaalikutsu (Complex Type Document Literal Call).

Tarkastellaan *yksinkertaisen tyypin RPC-kooditetun* tyypin osalta Richardsin (2006) esittämää esimerkkipalvelua, joka sisältää funktiot `FahrenheitTOCelsius()` ja `CelsiusTOFahrenheit()` ja jonka WSDL-palvelun oletetaan sijaitsevan osoitteessa:

```
http://java.hpcc.nectec.or.th:1978/axis/TemperatureConvert.jws?wsdl
```

Kuvan 28 esimerkki demonstroi, kuinka `$temp_celsius` muuttujassa määritelty 5° Celsiusta konvertoidaan Fahrenheit-asteiksi.

```
<?php
  /* Lämpötila Celsius-asteina */
  $temp_celsius = 5;

  /* WSDL-tiedoston sijainti */
  $wsdl=
    'http://java.hpcc.nectec.or.th:1978/axis/TemperatureConvert.jws?wsdl';
  $sClient = new SoapClient($wsdl);

  /* Tulostetaan lämpötila Fahrenheit-asteina */
  print $sClient->CelsiusTOFahrenheit($temp_celsius)
?>
```

Kuva 28. RPC-kooditetun tyypin kutsuminen (Richards, 2006).

Kuvan 28 esimerkistä nähdään, että etäfunktiota voidaan kutsua `SoapClient`-olion natiivi-metodina. Funktiota `CelsiusTOFahrenheit()` kutsutaan `$sClient`-muuttujasta, ikäänkuin se olisi `SoapClient`-luokan oikea metodi. Tämän tyyppinen kutsuminen on mahdollista myös ilman WSDL-tiedoston käyttöä. Kuvassa 29 on esimerkki saman toiminnallisuuden sisältävän palvelun kutsumisesta ilman WSDL-tiedoston käyttöä.

```
<?php
  $temp_celsius = 5;
  try {
    /* Asetetaan sijainti ja URI  */
    $location =
         'http://java.hpcc.nectec.or.th:1978/axis/TemperatureConvert.jws';
    $uri =
     'http://java.hpcc.nectec.or.th:1978/axis/TemperatureConvert.jws';

    /* Luodaan asiakas ilman WSDL-tiedostoa */
    $sClient = new SoapClient(NULL,array('location' => $location,
                                         'uri' => $uri,
                                         'style' => SOAP_RPC));
    print $sClient->CelsiusTOFahrenheit($temp_celsius)."\n";
    } catch (SoapFault $e) {
        echo $e->faultstring;
    }
?>
```

Kuva 29. RPC-kooditetun tyypin kutsuminen ilman WSDL-dokumenttia (Richards, 2006).

Ilman WSDL-tiedoston käyttöä asiakassovellus muodostaa parametrin tyypin käyttäen omia oletuksiaan. Tästä syystä on varmistettava se, että välitettävä parametri on oikeaa tyyppiä.

Esimerkiksi kuvan 29 tapauksessa päädytään virhetilanteeseen, jos palvelu vaatii syötteeksi `float`-tyypin.

Monimutkaista tyyppiä olevia literaalikutsuja voidaan muodostaa monilla eri tavoilla. Esimerkiksi kuvan 27 funktiot vastaanottavat ja palauttavat monimutkaista tyyppiä olevaa tietoa:

```
NumberToDollarsResponse NumberToDollars(NumberToDollars $parameters)
TitleCaseWordsResponse TitleCaseWords(TitleCaseWords $parameters)
```

Käytettävät tyypit ovat kuvan 27 mukaisesti rakenteita. PHP:n termistössä tämä tarkoittaa joko assosiatiivista taulukkoa tai oliota. Oletetaan, että halutaan kutsua funktiota `NumberToDollars()`. Oliota käytettäessä voidaan kuvan 27 `NumberToDollars`-rakenteelle määritellä seuraavanlainen luokka:

```
class NumberToDollars {
  public $dNum;
}
```

Edellä luotua luokkaa voidaan käyttää funktion kutsumiseen, kuten kuvasta 30 ilmenee.

```
$wsdl =
  'http://www.dataaccess.com/webservicesserver/conversions.wso?WSDL';
try {
    $xConverter = new SoapClient($wsdl);
        $param = new NumberToDollars();
        $param->dNum = 123456;
        $retVal = $xConverter->NumberToDollars($param);
        print $retVal->NumberToDollarsResult."\n";
    } catch (SoapFault $e) {
        var_dump($e);
    }
```

Kuva 30. Funktion kutsuminen käyttäen parametrina luokan ilmentymää (Richards, 2006).

Kuvan 30 koodin suorituksesta voisi palvelun toteutuksesta riippuen palautua esimerkiksi sanallinen ilmaisu: ”one hundred and twenty three thousand four hundred and fifty six

dollars”. Ilmentymän tuottavan luokan nimi ei ole tärkeä tässä tapauksessa, koska sille voidaan antaa mikä nimi tahansa, vaikkakin sen nimeäminen samanlaiseksi muodostetun rakenteen kanssa helpottaa sen tunnistamista ja vertaamista rakenteen määritelmän kanssa. Luokan ominaisuudet ovat tärkeitä, sillä niiden tulee käyttää samoja nimiä kuin mitä rakenteen osat käyttävät. Kun `SoapClient`-luokka luo SOAP-viestin, rakenteen osien nimiä käytetään elementtien niminä. Palautusarvo tässä esimerkissä on myös monimutkaista tyyppiä. Monimutkaisen tyypin arvot palautetaan olioina, jotka perustuvat PHP:n `stdClass`-luokkaan. Kuvassa 27 listattuihin funktioihin perustuen funktio `NumberToDollars` palauttaa `NumberToDollarsResponse`-tyypin, joka on rakenteeltaan seuraavanlainen:

```
struct NumberToDollarsResponse {
  string NumberToDollarsResult;
}
```

Nyt kun tiedetään, että rakenteet ja PHP-oliot ovat rinnastettavissa toisiinsa, voidaan sanoa että palautettu olio sisältää `NumberToDollarsResult`-osan ja se on `string`-tyyppinen. Tämän perusteella kuvan 30 esimerkki antaa vastauksena rakenteen osan sisältämän arvon. Yhtälailla voidaan käyttää taulukkoa syöteparametrina. Assosiatiivisen taulukon avaimet toimivat samoin kuin olion ominaisuudet. Taulukkoa käyttämällä on mahdollista saada sama tulos kuin kuvan 30 tapauksessa:

```
$param = array('dNum'=>123456);
$retVal = $xConverter->NumberToDollars($param);
```

Vaikka taulukkoa käytetään syötteenä, palautetaan silti olio, kuten edellisessä esimerkissäkin olion ollessa syötteenä.

4.2.5 *SOAP-otsikkojen lisääminen*

PHP:n SOAP-laajennus on sisältänyt versiosta 5.0.5 lähtien yksinkertaisen otsikonasetusmetodin `__setSoapHeaders(array SoapHeaders)`, jolla voidaan lisätä SOAP-otsikkoja. Jos käytössä on mikä tahansa aikaisempi versio, täytyy otsikot asettaa käyttämällä matalan tason kutsuja. Otsikonasetusmetodi asettaa asiakassovelluksen otsikoita, jotka ovat voimassa niin kauan kuin sovelluksen ilmentymäkin. Tämä tarkoittaa sitä, että kun metodia kerran käytetään, seuraavat asiakassovelluksen tekemät käskyt lisäävät otsikot SOAP-viestiin. Metodi hyväksyy joko `SoapHeader`-olioista koostuvan taulukon tai arvon NULL. Asetetut otsikot korvataan uusilla metodille välitetyillä arvoilla ja arvo NULL poistaa otsikon. Kuvan 31 esimerkki demonstroi tavan lisätä autentikointia SOAP-viestiin. Käyttäjänimi ja salasana asetetaan `Header`-entiteettiin vastaanottajan prosessoitavaksi.

```
/* Luodaan authentication-olio, jolla on muuttujat username ja password */
class authentication {
  public $username;
  public $password;
}
$auth = new authentication();
$auth->username = 'username';
$auth->password = 'password';

/* Kooditetaan olio */
$authVar = new SoapVar($auth, SOAP_ENC_OBJECT);
$header = new SoapHeader('urn:ExampleAPI', "Authentication",
                         $authVar, TRUE, SOAP_ACTOR_NEXT);

/* Asetetaan uudet otsikot käytettäväksi SOAP-viestejä luotaessa */
$sClient->__setSoapHeaders(array($header));
```

Kuva 31. Autentikoinnin lisääminen SOAP-otsikkoon (Richards, 2006).

Kun viesti lähetetään, ensimmäisen vastaanottajan (tässä tapauksessa viesti lähetetään vielä eteenpäin) tulee prosessoida otsikkoentiteettiä, joka sisältää kirjautumistiedot. Jos tämä ei onnistu, palautetaan `SoapFault`. Syy siihen, miksi näin toimitaan on se, että `mustUnder-`

stand-attribuutti on asetettu ja actor-attribuutti on tunnistettu seuraavaksi viestin vastaanottajaksi.

4.2.6 *Matalan tason kutsut*

Jos halutaan määritellä tarkemmin yksityiskohtia tietystä kutsusta, joka on asetettu käyttäen SoapClient-luokkaa, voidaan asetukset määrittää suoraan SoapClient-luokan kautta. Jos muutosten halutaan koskevan vain yhtä kutsua siten, että SoapClient-ilmentymään ei tehdä mitään muutoksia, voidaan käyttää __soapCall()-metodia:

```
__soapCall(string function_name [, array arguments [, array options [,
           mixed input_headers [, array &output_headers]]]])
```

Taulukko 5 kuvaa mahdollisia parametreja, joita __soapCall()-metodille voidaan syöttää.

Taulukko 5. Parametreja, joita __soapCall()-metodille voidaan syöttää (Richards, 2006).

Parametri	**Kuvaus**
function_name	Kutsuttavan funktion nimi.
arguments	Funktiolle välitettävien argumenttien taulukko.
options	Asetusten taulukko. Ainoat asetukset, jotka voidaan asettaa tämän parametrin kautta, ovat location, uri ja soapaction.
input_headers	Asettaa SOAP-otsikon. Parametrin arvo voi olla joko SoapHeader-olioista koostuva taulukko tai yksittäinen SoapHeader-olio.
output_headers	Muuttuja, johon mikä tahansa vastaanotetuista otsikoista tallennetaan.

Kuvan 32 esimerkki demonstroi, kuinka __soapCall()-metodille voidaan käyttää useita parametreja. Kun parametrit on koottu, SoapClient tekee palvelulle pyynnön doSearch-funktion suorittamiseksi.

```
$xConverter = new SoapClient(NULL,
              array('location'=>'http://www.example.com/exampleAPI',
                    'uri'=>'urn:ExampleAPI'));

/* Luodaan SoapParam-ilmentymä */
$param = array(new SoapParam('PHP XML', 'search_term'));

/* Luodaan options-taulukko */
$options =
array('location'=>'http://www.example.com/alternateExampleAPI',
                  'uri'=>'urn:AlternateExampleAPI');

/* Luodaan SoapHeader-ilmentymä */
$soapHeader = new SoapHeader("http://www.example.com",
                             "Transaction","Sisältö", TRUE,
                             SOAP_ACTOR_NEXT);

/* Kutsutaan etäfunktiota ja haetaan vastausotsikkoja */
$ret = $xConverter->__soapCall('doSearch', $param, $options,
                               $header, $response_headers);
```

Kuva 32. Esimerkki `__soapCall()`-metodin käytöstä (Richards, 2006).

4.2.7 *Viestin modifiointi*

Tarvittaessa SOAP-viestiä voidaan modifioida ennen sen lähettämistä. Käyttämällä aliluokkaa, `__doRequest()`-metodi voidaan ylikirjoittaa. Olio `SoapClient` kutsuu `__doRequest()`-metodia, kun funktiokutsu suoritetaan. Ensiksi se kokoaa SOAP-viestin ja sitten suorittaa kutsun tälle metodille, jotta viesti lähetettäisiin. Tekemällä `SoapClient`-luokasta aliluokan, on mahdollista, että tämä metodi muutetaan niin, että viestiin voidaan tehdä muutoksia ja sen lähetystapaa voidaan muuttaa. Seuraavassa on esitelty `__doRequest()`-metodin syntaksi:

```
__doRequest(string request, string location, string action, int version)
```

Parametri `request` on sarjallistettu SOAP-viesti. Käyttämällä jotakin XML-laajennusta, kuten DOM:ia, viesti voidaan ladata puurakenteeksi ja sitä voidaan muokata tämän rakenteen kautta. Parametri `location` on URL, jota kutsutaan palvelua tavoiteltaessa. Käyttämällä tätä arvoa voidaan muuttaa sijaintia, johon kutsu kohdistetaan. Parametri `action` on SOAP:in `action`-

attribuutti, joka on asetettu muuttujaan. Tätä arvoa käytetään HTTP-pyynnön `SOAPAction Header`-elementissä. Parametri `version` määrittää käytetyn SOAP-version (`SOAP_1_1` tai `SOAP_1_2`).

Jotta viestin modifiointi onnistuisi, täytyy luoda luokka, joka perii `SoapClient`-luokan. Kuvan 33 aliluokka `mySoapClient` keskittyy niiden pyyntöjen modifiointiin, joissa kutsutaan kohdassa 4.2.4 käsiteltyä `CelsiusToFahrenheit()`-funktiota.

```
Class mySoapClient extends SoapClient {
  function __doRequest($request, $location, $action, $version) {
    /* Ladataan pyyntö DOMDocument-olioksi */
    $dom = new DOMDocument();
    $dom->loadXML($request);

    /* Etsitään temp-elementti ja asetetaan sen arvoksi 20C */
    $nodeList = $dom->getElementsByTagName('temp');
    if ($nodeList->length == 1) {
      $nodeList->item(0)->firstChild->nodeValue = "20";
    }

    /*  Sarjallistetaan puu ja lähetetään pyyntö yläluokan metodille */
    $request = $dom->saveXML();
    return parent::__doRequest($request, $location, $action,$version);
  }
}
```

Kuva 33. Esimerkki pyyntöjen modifioinnista (Richards, 2006).

Metodi `__doRequest()` on nyt lisätty aliluokkaan. Kun tätä kutsutaan, se lataa pyynnön `DOM-Document`-olioon, etsii `temp`-elementin ja muuttaa sen arvoksi 20, ja sitten lähettää modifioidun pyynnön yliluokalle käsiteltäväksi. Kun aliluokka on määritelty, voidaan luoda asiakassovellus, joka perustuu tähän uuteen luokkaan (kuva 34).

```
$wsdl =
"http://java.hpcc.nectec.or.th:1978/axis/TemperatureConvert.jws?wsdl";
$sClient = new mySoapClient($wsdl);

/* Asetetaan alkuperäinen konvertoitava lämpötila: */
$temp_celsius = 5;
$tempVar = new SoapVar($temp_celsius, XSD_FLOAT);

/* Lopuksi kutsutaan funktiota normaalisti */
print $sClient->CelsiusTOFahrenheit($tempVar)."\n";
```

Kuva 34. Asiakassovellus, joka käyttää modifioitua `SoapClient`-luokkaa (Richards, 2006).

Tulos on erilainen verrattuna tuloksiin, jotka seuraavat kuvien 28 ja 29 modifioimattoman version suorittamisesta. Sen sijaan, että palautettaisiin arvo 41, palautetaan 68. Tässä `__doRequest()`-metodin versiossa SOAP-viestiä modifioidaan ennen kuin se lähetetään, eli muutetaan arvoksi 20 alkuperäisen 5:n sijaan.

Metodi `__doRequest()` tarjoaa mahdollisuuden useiden teknologioiden käytölle. Esimerkiksi SOAP-laajennuksella ei ole yhtään natiivia ominaisuutta web-palveluiden turvallisuuden takaamiseksi. Lisäämällä edellisen esimerkin tavoin turvallisuutta parantavia tekijöitä, voidaan varmistua siitä, että SOAP-viestien väärinkäyttö ei ole mahdollista.

4.2.8 *Asiakaskutsujen debuggaus*

SOAP:ia käytettäessä debuggaus voi olla hankalaa. SOAP:in `Fault`-elementti tarjoaa paljon informaatiota, mutta silti tästä informaatiosta on selvitettävä virheen aiheuttaja. Kun `SoapClient`-ilmentymä luodaan käyttäen `trace`-optiota arvolla 1, on käytössä neljä metodia, joilla voidaan tarkastella eri osia SOAP-viesteistä. Ilman `trace`-optiota nämä metodit ovat kyllä käytössä, mutta ne palauttavat vain tyhjiä merkkijonoja. Kuvatut metodit ovat `__getLastRequest()`, joka palauttaa asiakkaan lähettämän sarjallistetun `SOAP-ENV:Envelope`-dokumentin merkkijonona, `__getLastRequestHeaders()`, joka palautta HTTP-pyynnön otsikot

sisältävän merkkijonon, `__getLastResponse()`, joka palauttaa palvelimelta tulleen sarjallistetun `SOAP-ENV:Envelope`-dokumentin ja `__getLastResponseHeaders()`, joka palauttaa HTTP-vastauksen otsikot sisältävän merkkijonon.

Debuggauskutsujen hyödyntäminen onnistuu kutsumalla sopivia metodeja ilman välitettäviä argumentteja, jolloin palautuva merkkijono sisältää halutun tiedon. Kuvan 35 esimerkissä kutsutaan funktiota, joka tulee aiheuttamaan SOAP-virheen. Kun `SoapFault` otetaan kiinni, saatu merkkijono ja pyyntöviesti, sekä HTTP-vastauksen otsikot saadaan vastauksena.

```
try {
  $response = $sPublish->getPeopleByFirstLastName(1,2, 3);
} catch (SoapFault $e) {
  echo "Fault: ".$e->faultstring."\n\n";
  echo $sPublish->__getLastRequest()."\n\n";
  echo $sPublish->__getLastResponseHeaders();
  }
-------------------------------------------------------------
Fault: Error cannot find parameter

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV = "http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1="urn:ExampleAPI">
    <SOAP-ENV:Body>
      <ns1:getPeopleByFirstLastName/>
        <param1>2</param1>
        <param2>3</param2>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

HTTP/1.1 500 Internal Service Error
Date: Fri, 02 Dec 2005 08:29:32 GMT
Server: Apache/2.0.53 (Win32) PHP/6.0.0-dev
X-Powered-By: PHP/6.0.0-dev
Content-Length: 294
Connection: close
Content-Type: text/xml; charset=utf-8
```

Kuva 35. Funktion kutsu, SOAP-pyyntö ja HTTP-vastaus (Richards, 2006).

Virhemerkkijonosta voidaan päätellä, että palvelimelle välitettävissä parametreissa on jotain vikaa. Vastausotsikko ei sisällä tässä tapauksessa lisätietoja virheen yksityiskohdista. SOAP-pyynnöstä nähdään, että rakenne on väärä. Elementtiin `Body` tulisi sisältyä vain yksi `ns1:getPeopleByFirstLastName`-elementti, joka sisältää kaksi muuta elementtiä. Sen sijaan kuvassa

35 suoritettava funktion kutsu muodostaa SOAP-pyynnön, jonka `Body`-elementti sisältää kolme lapsielementtiä, mikä ei täsmää liitteen 1 WSDL-dokumentissa määritellyn syöteviestin kanssa. Ilman SOAP-viestin tarkastelua voidaan joutua määrittämään monia muutoksia ennen kuin todellinen syy selviää.

4.3 Palvelin

Jos käytetään WSDL-dokumenttia, SOAP-palvelimen muodostaminen onnistuu helposti. Kuten asiakassovelluksen, myös palvelinsovelluksen kannalta päätepisteet, sidonnat, operaatiot ja tietotyypit voidaan asettaa käyttämällä WSDL-dokumenttia. Vaikein osa SOAP-palvelimen kirjoittamisessa onkin WSDL-dokumentin kirjoittaminen. Seuraavaksi käydään läpi esimerkki web-palvelun toteutuksesta, jonka lähdekoodi löytyy liittestä 2 (palvelin-, sekä asiakassovelluksen lähdekoodit). Toteutettava sovellus perustuu liitteen 1 WSDL-dokumenttiin ja se toteuttaa funktion, joka mahdollistaa asiakassovellukselle henkilötietojen hakemisen etu- ja sukunimeen perustuen. Osittaiset nimet sallitaan käyttämällä *-merkkiä ”jokerimerkkinä”. Esimerkiksi jos etsitään j-kirjaimella alkavaa henkilön nimeä, kirjoitetaan hakukriteeriksi j*.

Luokan `SoapServer` ilmentymää luotaessa konstruktori on seuraavanlainen:

```
__construct(mixed wsdl [, array options])
```

Parametri `wsdl` määrittää URI:n palvelimen käyttämälle WSDL-dokumentille. Jos ei käytetä WSDL-dokumenttia, tämän parametrin arvoksi asetetaan NULL. Parametri `options` määrittää palvelimen käyttämät asetukset. Tämä parametetri on valinnainen työskenneltäessä WSDL-dokumentin kanssa. Taulukossa 7 on listattu mahdolliset asetukset ja kuvassa 36 on esimerkkejä SOAP-palvelimen luomisesta eri tavoilla.

Taulukko 7. Mahdollisia parametreja `SoapServer`-luokan konstruktorille (Richards, 2006).

Parametri	**Kuvaus**
`actor`	Attribuutti `actor` SOAP-palvelimelle.
`classmap`	Assosiatiivinen taulukko, joka liittää WSDL-tyypit PHP-luokiksi. Taulukon avaimet ovat WSDL-tyyppejä ja arvot luokkia.
`encoding`	Käytettävä kooditus merkkijonoja palautettaessa.
`soap_version`	Määrittää sen, käytetäänkö SOAP 1.1 vai SOAP 1.2 versiota. Voi saada joko arvon `SOAP_1_1` tai `SOAP_1_2`.
`uri`	SOAP-palvelimen kohdenimiavaruus.

```
/* Luodaan palvelin käyttämällä WSDL-tiedostoa ja
   määrittelemällä soap-versio, sekä actor:in URI  */

$server = new SoapServer("mywsdl.wsdl",
                         array('soap_version' => SOAP_1_2,
                         'actor' => "http://www.example.com/actorA"));

/* Luodaan palvelin käyttämällä WSDL-tiedostoa ja
   liitetään siihen Person-luokka                  */

class Person {
   public $id;
   public $firstName;
   public $lastName;
}

$server = new SoapServer("mywsdl.wsdl",
                         array('classmap' => array('book' => "Person")));

/* Luodaan palvelin ilman WSDL-tiedostoa ja asetetaan URI */
$server = new SoapServer(null, array('uri' => "urn:ExampleAPI"));
```

Kuva 36. Esimerkkejä SOAP-palvelimen luomisesta (Richards, 2006).

Liitteen 2 esimerkissä käsiteltävä palvelin muodostetaan käyttämällä WSDL-dokumenttia exampleapi.wsdl syötteenä pelkästään dokumentin sijainti:

```
$sServer = new SoapServer("exampleapi.wsdl");
```

Palvelimen rekisteröimät palvelun sisältämät toiminnot voidaan toteuttaa funktioina tai metodeina.

4.3.1 *Palvelun toiminnoista vastaavat funktiot*

Kun `SoapServer`-ilmentymä on luotu, saapuvien kutsujen käsittelystä tulee pitää huolta. Käsittelijöiden kirjoittaminen käyttäen funktioita eroaa hiukan tavallisten PHP-funktioiden kirjoittamisesta. Tulee varmistua siitä, että palautettava data on oikeaa tyyppiä. Käytettäessä WSDL-dokumenttia tämä on helppoa, koska useimmissa tapauksissa data tyypitetään ja koo-ditetaan oikein. Tyypin tarkistus onnistuu kätevästi käyttämällä asiakassovellusta siten, että sillä listataan kaikki tyypit ja funktiot ja sitten verrataan tulostetta odotettuihin arvoihin. Tätä tarkistustapaa demonstroidaan Edustaja-suunnittelumallin toteutusta käsittelevässä luvussa 5.

Funktiot tulee nimetä samoin kuin WSDL-operaatioiden nimet. Käyttämällä esimerkkinä liitteen 1 WSDL-dokumenttia vain yksi operaatio, `getPeopleByFirstLastName`, määritellään. Se ottaa vastaan yhden parametrin, joka on tyypiltään `getPeopleByFirstLastName`-rakenne, joka sisältää merkkijonot etu- ja sukunimelle. Tämä on määritelty `getPeopleBy-FirstLastName`-viestin osien määrityksessä. Funktio saa viestin osat parametrina. Liitteen 1 esimerkin WSDL-dokumentti käyttää `document/literal`-tyyppiä SOAP-viestille, jolloin funktio yleensä ottaa vastaan viestin osat yhtenä parametrina, jolloin funktiossa pitää viitata tämän rakenteen sisältämiin osiin. Kun käytetään tyyppiä `rpc/encoded`, palvelimen funktio ottaa vastaan useita parametreja, yhden jokaista osaa kohti.

Luotavan funktion tulee olla nimeltään `getPeopleByFirstLastName` ja sen tulee vastaanottaa yksi parametri, joka voi myös olla nimeltään `getPeopleByFirstLastName`, vastaten WSDL-dokumentin tyypin nimeä:

```
function getPeopleByFirstLastName($getPeopleByFirstLastName) {}
```

Käytettävän funktion nimi tulee olla sama kuin WSDL-dokumentissa on määritelty, mutta parametrin voi kuitenkin nimetä vapaasti. On tosin helpompaa pitää nimet samana kuin

WSDL-dokumentissa, jotta parametrien ja rakenteiden käsittely olisi helpompaa palvelua kehitettäessä.

Syöteparametrit ja palautettava data perustuvat WSDL-dokumentttiin. Syöteparametri on `getPeopleByFirstLastName`-rakenne. Tämä rinnastuu PHP:n `stdClass`-olioon ominaisuuksin `first` ja `last`. WSDL-dokumentista johtuen palautustyyppi käytettäessä `SoapClient`-olion `__getFunctions()`-metodia näytetään `ArrayOfPerson`-tyyppisenä, eikä `getPeopleByFirstLastNameResponse`-tyyppinä. Syy tähän on se, että elementti `getPeopleByFirstLastNameResponse` viittaa `ArrayOfPerson`-tyyppiin. Kuvan 37 koodi demonstroi kuinka syöteparametreja käytetään.

```
$people =
  array(array('id'=>1, 'firstName'=>'John', 'lastName'=>'Smith'),
  array('id'=>2, 'firstName'=>'Jane', 'lastName'=>'Doe'));

/* Asetetaan asiakassovellukselta saadut arvot first ja last */
$firstSearch =
  str_replace('*', '([a-z]*)', $getPeopleByFirstLastName->first);
$lastSearch =
  str_replace('*', '([a-z]*)', $getPeopleByFirstLastName->last);
$matching = array();

/* Etsitään kaikki täsmäävät tiedot */
foreach($people AS $person) {
  if (empty($firstSearch) ||
      preg_match('/^'.$firstSearch.'$/i', $person['firstName']))
    if (empty($lastSearch) ||
        preg_match('/^'.$lastSearch.'$/i', $person['lastName'])) {
       /* Match found - Add the record to our return list */
       $matching[] = $person;
    }
}
```

Kuva 37. Esimerkki syöteparametrien käytöstä (Richards, 2006).

Kuvan 37 sovellus luo taulukon, joka sisältää tiedot kahdesta henkilöstä. Koodi konvertoi käyttäjän pyynnön säännöllisiksi lausekkeiksi ja käy läpi taulukon henkilöt. Syötteenä saadun tiedon perusteella se joko lisää henkilön tuloslistaan tai hyppää sen yli siirtyen seuraavaan taulukon henkilöön.

Kun kaikki kriteerejä vastaavat tiedot on löydetty ja kerätty `$matching`-muuttujaan, ne täytyy palauttaa asiakassovellukselle. Palautus voidaan toteuttaa muutamalla eri tavalla. Suoraviivaisin tapa on yksinkertaisesti palauttaa taulukko (`return $matching;`). Kun käytetään WSDL:ää, palvelin koodaa palautusarvon SOAP-viestiin. Jos ei käytetä WSDL-dokumenttia tai asiakassovellus ei käytä WSDL-dokumenttia tai jos WSDL-dokumentti sisältää kirjoitusvirheen, voidaan palautettava data kirjoittaa käyttämällä `SoapVar`-oliota. Kuvan 38 esimerkki perustuu kuvan 37 henkilötietotaulukkoon `$matching`, joka sisältää hakutulokset `$people`-taulukosta.

```
/* Alustetaan palautustaulukko */
$retval = array();

/* Käydään läpi palautettavaa taulukkoa ja luodaan jokaiselle henkilölle
   SoapVar-ilmentymä, joka lisätään palautettavaan taulukkoon.          */
foreach ($matching as $person) {
    $retval[] =
       new SoapVar($person, SOAP_ENC_ARRAY, "Person", "urn:ExampleAPI");
}

/* Palautetaan lopullinen data */
return $retval;
```

Kuva 38. Esimerkki `SoapVar`-olion käytöstä palautusviestin luomiseksi (Richards, 2006).

Kuten kuvan 38 esimerkistä nähdään, joudutaan työskentelemään manuaalisesti hiukan enemmän, mutta tämä on kuitenkin turvallista, koska varmistutaan siitä, että data kooditetaan tarkalleen sellaiseksi kuin sen halutaan SOAP-viestin kannalta olevan. Jos hakutuloksien välittämiseen käytetään oliotaulukkoa, ainoa mahdollisuus olisi koodittaminen käyttäen `SoapVar`-konstruktoria, jolloin `SOAP_ENC_OBJECT` olisi oikea tyyppi koodikselle.

Kun kaikki tarvittavat funktiot on määritelty, ne täytyy rekisteröidä `SoapServer`-oliota käyttäen. Tähän tarkoitukseen voidaan käyttää `addFunction()`-metodia. Funktioita voidaan lisätä kolmella eri tavalla. On mahdollista lisätä yksittäisiä funktioita yksi kerrallaan kutsumalla `addFunction()`-metodia ja välittämällä parametrina funktion nimi merkkijonona. Toi-

nen tapa on käyttää merkkijonotaulukkoa, jossa jokainen merkkijono on funktion nimi. Kolmas tapa on funktioiden rekisteröinti kutsumalla metodia `addFunction()` ja välittämällä parametrina `SOAP_FUNCTIONS_ALL`-vakio. Tämä vakio ohjeistaa `SoapServer`-oliota lisäämään jokaisen skriptissä määritellyn funktion. Tämä on kätevää silloin kun skripti sisältää vain SOAP-pyyntöjen käsittelyyn tarkoitettuja funktioita. Jos skripti on iso ja sisältää monia asiaan liittymättömiä funktioita, on turvallisempaa rekisteröidä funktiot yksi kerrallaan tai taulukkoa hyväksi käyttäen. Liitteen 2 esimerkissä oleva palvelin koostuu yhdestä kutsuttavasta funktiosta, joten se lisätään nimen perusteella: `$sServer->addFunction('getPeopleByFirstLastName')`.

4.3.2 *Palvelun toiminnoista vastaava luokka*

Sen sijaan, että täytyisi rekisteröidä useita funktioita, voidaan käyttää luokkaa, joka sisältää kaikki metodit SOAP-pyyntöjen käsittelyyn. Luokan metodeja voidaan kirjoittaa samalla tavalla kuin funktioita voidaan kirjoittaa SOAP-pyyntöjen hallintaan. Ero näiden kahden tavan välille muodostuu niiden rekisteröintitavasta. Toisin kuin funktiot, jotka täytyy eksplisiittisesti rekisteröidä käyttäen `addFunction()`-metodia, luokan metodit rekisteröidään kokonaisuudessaan käyttämällä `setClass()`-metodia:

```
setClass(string class_name [, mixed args [, mixed ...]])
```

Parametri `class_name` on `SoapServer`:iin rekisteröitävän luokan nimi. Vain yhtä luokkaa voidaan käyttää, joten tämän metodin kutsuminen uudelleen eri luokalla korvaa alunperin käytetyn luokan uudella. Jäljellä olevat argumentit tälle metodille ovat sellaisia, jotka välitetään, kun ilmentymä luodaan.

Palvelimelle muodostetun olion ilmentymän elinaika on sama kuin pyynnöllä. Tämä tarkoittaa sitä, että joka kerta kun asiakassovellus tekee kutsun palvelimelle, palvelimen tulee muodostaa uusi ilmentymä rekisteröidystä luokasta. On kuitenkin mahdollista, että asiakassovellus säilyttää olion tilan tekemällä olion pysyväksi käyttämällä `setPersistence()`-metodia. Metodi ottaa vastaan yksittäisen parametrin, joka voi olla oletusta vastaava `SOAP_PERSISTENCE_REQUEST`-vakio, jolloin tilaa ei säilytetä. Sen sijaan vakio `SOAP_PERSISTENCE_SESSION` sallii olion tilan säilyvyyden kutsujen välillä PHP-session aikana. Kun luokka on rekisteröity palvelimelle, kutsutaan sitä seuraavalla tavalla:

```
$sServer->setPersistence(SOAP_PERSISTENCE_SESSION);
```

Oliosta muodostetaan ilmentymä ensimmäisen asiakassovellukselta tulleen pyynnön yhteydessä ja se (ilmentymä) tallennetaan session muistiin. Peräkkäisissä kutsuissa olio palautetaan sessiosta, eikä uutta ilmentymää luoda.

4.3.3 *Asiakaspyyntöjen käsittely*

Kun `SoapServer` on saatu toimimaan, eli kaikki funktioiden käsittelijät on rekisteröity tai luokka ja sen pysyvyys on asetettu, pitää vielä käsitellä asiakassovellukselta tulevat pyynnöt. Tätä varten on olemassa `handle()`-metodi:

```
handle([string soap_request])
```

Useimmissa tapauksissa tätä metodia kutsutaan ilman argumentteja. Palvelut suoritetaan yleensä web-palvelimella ja `SoapServer` hakee SOAP-viestin sieltä, käsittelee sen ja palauttaa vastauksen asiakassovellukselle. Esimerkiksi liitteen 2 palvelun suorittamiseksi palvelin

tekee `$sServer->handle()`-kutsun, joka käsittelee syötteen ja palauttaa viestin asiakassovellukselle.

Operaatiot eivät kuitenkaan toimi aina virheettömästi. Esimerkiksi jos pyynnön vastaanottavan funktion tulee päästä käsiksi tietokantaan ja tietokanta sattuu olemaan epäkunnossa, tai kysely epäonnistuu jostain tuntemattomasta syystä, halutaan luultavasti informoida asiakasta siitä, että jonkinlainen virhe on havaittu. Luokkaa `SoapFault` käsittelevässä kohdassa 4.1.4 esiteltiin tämän tyyppisten olioiden luomisessa käytettävää syntaksia. Virhe palautetaan asiakassovellukselle siten, että kutsuttava funktio joko heittää tai palauttaa `SoapFault`-olion. Tarkasteltava liitteen 2 palvelinsovellus sisältää kaksi nimettyä WSDL-dokumentissa `getPeopleByFirstLastName`-operaatiolle määriteltyä virhettä. Nämä ovat nimeltään `nodb` ja `sysmaint`.

Palvelua ei pitäisi koskaan sulkea kokonaan. Web-palvelimen tulisi aina olla käynnissä ja palvelun tulisi aina hyväksyä pyyntöjä. Tyypillisesti järjestelmän ylläpito aiheuttaa kuitenkin käyttökatkoja. Globaalia, koko systeemin laajuista muuttujaa `$SYS_STATUS`, käytetään ilmaisemaan onko järjestelmä käytössä vai ei. Asettamalla arvoksi `FALSE` ilmaistaan, että palvelu on poissa käytöstä ja kun näin on, palvelun tulisi palauttaa `sysmaint`-virhe asiakassovellukselle. Tämä tarjoaa asiakkaalle informaatiota siitä, mitä on meneillään, sekä ajan, kuinka kauan tulisi odottaa ennen seuraavan kutsun tekemistä, kuten kuvan 39 esimerkissä tehdään.

```
function getPeopleByFirstLastName($getPeopleByFirstLastName) {
  if (isset($GLOBALS['SYS_STATUS']) && $GLOBALS['SYS_STATUS'] == FALSE)
  {
    /* Asetetaan details-rakenne */
    $details = array("SysMessage"=>"System Maintenance",
                     "RetryInMinutes"=>60);
   /* Heitetään uusi SoapFault-poikkeus */
    throw new SoapFault("SYSError", "System Unavailable",
                        "urn:ExampleAPI", $details, "sysmaint");
  }
  /* Funktion koodi tähän */
}
```

Kuva 39. Esimerkki toiminnasta, kun palvelu on poissa käytöstä (Richards, 2006).

Muuttujaa `$details` käytetään kuvan 39 koodissa yksityiskohtia sisältävänä parametrina. Virhekoodin nimi `sysmaint` välitetään `SoapFault`-konstruktorille ja se asettaa WSDL-dokumentissa määritellyn rakenteen `SystemMaintenance` tälle virheelle sisältäen `details`-muuttujan tiedot. Poikkeuksen `SoapFault` yksityiskohdat voidaan selvittää asiakassovelluksessa katsomalla sopivia SOAP-viestin osia kutsumalla `SoapFault`-ilmentymälle `var_dump()`-metodia, josta seuraa kuvan 40 mukainen tuloste.

```
["faultstring"]=>
string(18) "System Unavailable"
["faultcode"]=>
string(12) "ns1:SYSError"
["faultactor"]=>
string(14) "urn:ExampleAPI"
["detail"]=>
object(stdClass)#2 (1) {
  ["SystemMaintenance"]=>
  object(stdClass)#3 (2) {
     ["SysMessage"]=>
     string(8) "DB Error"
     ["RetryInMinutes"]=>
     string(2) "60"
  }
}
```

Kuva 40. Metodin `var_dump()` kutsusta seuraava tuloste (Richards, 2006).

4.3.4 *SOAP-otsikoiden käsittely*

Jotta SOAP-otsikoita voitaisiin käsitellä kunnolla, täytyy määritellä `soap:header`-elementti `wsdl:operation`-elementtiin, joka sijaitsee `wsdl:binding`-elementin alla. Jos `soap:header`-elementtiä ei ole sisällytetty WSDL-dokumentttiin ja asiakassovellus lähettää otsikon `mustUnderstand`-attribuutin kanssa, `SoapServer` palauttaa virheen, vaikka otsikoiden käsittelijä olisi asetettu oikein palvelimen sovellukseen. Otsikon käsittely toteutetaan samalla tavalla kuin funktioiden käsittely. Voidaan kirjoittaa joko funktio tai luokan metodi. Tärkeitä aspekteja ovat funktion tai metodin nimi ja `SoapServer`-ilmentymään määritelty `actor`-attribuutti.

Seuraavassa on esitetty SoapHeader-konstruktori, jota SoapClient käyttää otsikoiden asettamiseen:

```
__construct(string namespace, string name [, mixed data [,
            bool mustUnderstand [, mixed actor]]])
```

Parametri actor määrittää sen palvelimen URI:n, jonka tarkoitus on käsitellä otsikkoa. Jos tätä parametria ei ole määritelty tai se on sama kuin SOAP_ACTOR_NEXT-vakio, SoapServer-ilmentymä yrittää käsitellä otsikkoa.

Otsikkoelementtiä käsittelevä funktio tai metodi nimetään soap:header-elementin käyttämän part-elementin mukaan. Arvo riippuu käytetystä koodituksesta, joka riippuu siitä, mitä käytetään WSDL-dokumentissa. Se voi olla esimerkiksi rpc/encoded tai document/literal. Välitetyt parametrit perustuvat elementin datatyyppiin (document/literal) tai part-elementin datatyyppiin (rpc/encoded). Tämä sama asia on helpommin ymmärrettävissä WSDL-dokumentin kautta. Esimerkiksi jos käytetään document/literal-tyyppiä, voidaan kuvan 41 mukaisesta WSDL-dokumentista määritellä toteutettavien funktioiden nimet.

```
<definitions ....>
  <types>
    <xsd:schema ...>
      <!- Muut tyypit -->
      <xsd:element name="headerfunc" type="xsd:string"/>
    </xsd:schema>
  </types>
  <message name="headermsg">
    <part name="param1" element="tns:headerfunc"/>
  </message>
  <binding ...>
    <operation ...>
      <soap:operation .../>
      <input>
        <soap:header message="tns:headermsg" part="param1" use="literal"/>
          <soap:body .... />
      </input>
    </operation>
    <!-- Muut operaatiot -->
  </binding>
</definitions>
```

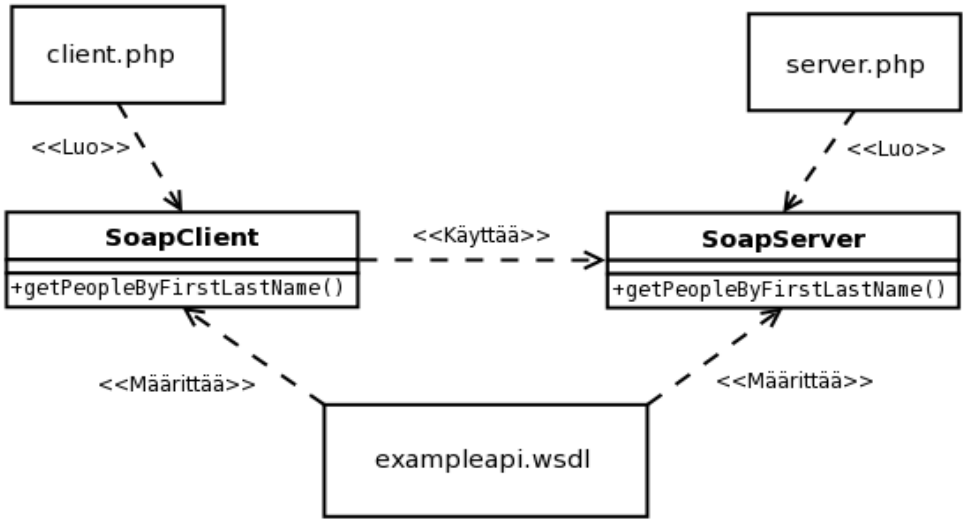
Kuva 41. WSDL-dokumentti, josta ilmenee toteutettavien funktioiden nimet (Richards, 2006).

Aloittamalla kuvan 41 WSDL-tiedoston `operation`-elementin sisällä olevasta `soap:header`-elementistä nähdään, että `message`-attribuutin arvo on `tns:headermsg` ja `part`-attribuutin arvo on `param1`. Kun WSDL-dokumentista on paikannettu tietty `part`-attribuutti `message`-parametria varten, voidaan todeta, että elementti on nimeltään `tns:headerfunc`. Käyttämällä tätä nimeä, toteutettavaa funktiota kutsutaan nimellä `headerfunc()`. Elementti on `xsd:string`-tyyppinen, joten funktio hyväksyy yksittäisen merkkijonoparametrin. Lopullinen prototyyppi toteutettavalle funktiolle tai metodille näyttää seuraavalta:

```
headerfunc(string $param)
```

4.4 Esimerkkisovellus

SOAP-palvelimen muodostamista on nyt käsitelty monesta näkökulmasta. Tässä vaiheessa voidaan koota palaset yhteen ja luoda toimiva SOAP-palvelin, jonka lähdekoodi löytyy liitteestä 2 ja WSDL-dokumentti, johon sovellus perustuu, löytyy liitteestä 1. Kuvassa 42 on kuvattu liitteen 2 sisältämien asiakas- ja palvelinsovelluksien muodostamaa kokonaisuutta.



Kuva 42. Asiakas- ja palvelinsovelluksen muodostama kokonaisuus.

Se osa koodista, joka ei käsittele SOAP:ia, tekee paljon oletuksia, joita ei todellisessa sovelluksessa tehtäisi, mutta tärkeintä on huomata se, kuinka SOAP:ia käytetään yhdistämään palaset toisiinsa. SOAP:in osuus ohjelmakoodista koostuu palvelun muodostamisesta ja funktion käyttöönotosta, sekä sen kutsumisesta asiakassovelluksella. Liitteessä 2 on esitetty kokonaisuudessaan palvelinsovelluksen koodi (server.php) sekä asiakassovelluksen koodi (client.php). Liitteen 2 asiakassovellus tekee palvelimelle kutsun, jossa metodia `getPeopleByFirstLastName()` kutsutaan parametrilla, joka on taulukkorakenne `array('first'=>'jo*', 'last'=>'*')`, eli haetaan henkilöitä, joiden etunimi alkaa kirjaimilla ”jo” ja sukunimi voi olla mikä tahansa.

4.5 Ulkoisia SOAP-kirjastoja

Natiivi SOAP-tuki versiossa PHP 5 oli suuri harppaus kohti XML-perustaisten teknologioiden hyödyntämistä. Ennen kuin natiivia tukea oli saatavilla, ilmestyi vaihtoehtoisia PHP:llä kirjoitettuja toteutuksia, kuten PEAR SOAP ja NuSOAP. Vaikka edellä mainitut ovat varteenotettavia vaihtoehtoja, natiivi SOAP-tuki on kirjoitettu C-kielellä, joten se on nopeampi ja luotettavampi (Richards, 2006).

Jotta PHP:n natiivi SOAP-tuki saataisiin käyttöön, tulee PHP asentaa siten, että SOAP-tuki otetaan mukaan. Jos palvelimella olevassa PHP-asennuksessa ei ole sisällytettynä SOAP-tukea, eikä sen asentaminen jälkikäteen ole mahdollista, voidaan turvautua ulkoisiin SOAP-kirjastoihin. Saatavilla on PHP:n oman SOAP-laajennuksen lisäksi muutamia ulkoisia SOAP-kirjastoja. Varteenotettavia vaihtoehtoja ovat esimerkiksi PEAR SOAP ja NuSOAP.

PEAR SOAP (PHP Group, 2007a) sisältää SOAP-palvelimen ja asiakassovelluksen toteutuksen, jotka on kirjoitettu kokonaan PHP:llä. Kirjasto on ollut saatavilla jo useita vuosia,

mutta silti se on vielä beta-vaiheessa. Suurin ongelma tässä on se, että document/literal-tyyppisten viestien luominen ei onnistu. Asiakassovelluksen puolella tämä onnistuu, mutta palvelimen puolella ei. Tämä aiheuttaa melkoisen ongelman, koska document/literal-tyyppisiä palveluja käytetään melko paljon, jopa useammin kuin RPC/encoded-tyyppisiä (Richards, 2006).

NuSOAP (2007) on kokoelma PHP-luokkia, jotka mahdollistavat SOAP-viestien lähettämisen ja vastaanottamisen HTTP-protokollan yli. NuSOAP, joka tunnettiin aikaisemmin nimellä SOAPx4, on julkaistu GNU LGPL -lisenssin alla. SOAPx4-kirjastoa on käytetty perustana monessa PHP:n web-palvelu-paketissa, kuten esimerkiksi edellä mainitussa PEAR SOAP:issa (Ayala et al., 2002).

5 Edustaja-suunnittelumallin soveltaminen SOAP-protokollaa käytettäessä

Suunnittelumalli (Design Pattern) on kirjoitettu kuvaus, joka tarjoaa suunnitelmatason ratkaisun johonkin toistuvasti esiintyvään sovelluksen toteutusongelmaan. Kun jonkin ongelman ratkaisu on kirjoitettu ylös, monet muut voivat käyttää samaa ideaa uudelleen (Braam, 2004). Suunnittelumalli on käytännöllinen apuväline ratkaistaessa annettua ongelmaa. Se ei ole koodikirjasto, jota käytetään ohjelmassa, vaan malli, joka kertoo miten koodin rakenne kannattaa muodostaa. Koodikirjastoa ja suunnittelumallia käytetään varsin eri tavoin (Sweat, 2005).

5.1 Mallin rakenne

Suunnittelumalleja PHP:lle on esitellyt esimerkiksi Sweat (2005) ja toteuttanut testauslähtöisesti esimerkiksi Rikala (2006). Sweatin (2005) esittämä Edustaja-suunnittelumallin PHP-toteutus tarjoaa oliolle edustajan (proxy), joka sijaitsee asiakassovelluksen ja kohdeolion välillä. *Edustaja* voi mahdollistaa esimerkiksi laiskan ilmentämisen (lazy instantiation), pääsyn hallinnan (access control) tai pelkästään kutsujen eteenpäin välittämisen. Edustajaa, joka toimii paikallisesti, kutsutaan nimellä *virtuaaliedustaja* (virtual proxy). Etäpalveluiden kanssa toimivaa edustajaa kutsutaan nimellä *etäedustaja* (remote proxy).

Jos tarvitsee esimerkiksi viivyttää olion ilmentymän luomista, koska se käyttää kallisarvoisia resursseja ja jos vain osan koodista tarvitsee käyttää SOAP-protokollaa tai jos oliolle tulee tarjota erilaisia käyttöoikeuksia eri käyttäjäryhmille, voidaan käyttää Edustaja-mallia. Edellä mainitut ongelmat ovat hyvin yleisiä ja ne johtavat isompaan ongelmaan: kuinka tarjota yhtenäinen rajapinta oliolle, joka saattaa muuttua, tai jota ei vielä ole edes olemassa (Sweat, 2005). SOAP-protokollan käyttöön tämä malli tarjoaa erittäin hyvän lähestymistavan. Käyttämällä

Edustaja-mallia huolehditaan mahdollisesta palvelun puuttumisesta (esimerkiksi kaatunut palvelin) sekä palvelun muutoksista (esimerkiksi metodien nimet vaihtuvat, tai niitä tulee lisää). Tärkein Edustaja-mallin käytöstä saatava hyöty on se, että tarjotaan oliolle yhtenäinen rajapinta, jonka kautta palvelua voidaan käyttää luotettavasti.

Edustaja-mallin päätarkoitus on säilyttää ilmentymämuuttujassa viittausta kohdeolioon ja välittää edustajaluokan kutsut kohdeoliolle. Seuraavaksi käsitellään Edustaja-malli yksinkertaistetussa muodossa. Kuvassa 43 on esitetty kohdeluokka.

```
class Subject {
  function someMethod() {
    sleep(1);
  }
}
```

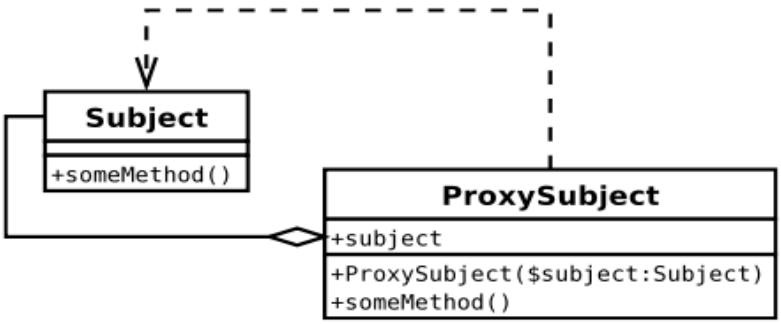
Kuva 43. Yksinkertaistettu kohdeluokan toteutus (Sweat, 2005).

Seuraavaksi tarvitaan edustajaluokka (proxy class), jolla täytyy olla ilmentymä kohdeluokasta. Edustajaluokan tulee tarjota kaikki julkiset metodit, jotka kohdeluokalla on. Tässä tapauksessa metodi on `someMethod()`-metodi. Kuvassa 44 `ProxySubject` kutsuu kohdeluokkaa käyttämällä `$this->subject->someMethod()`-metodia. Edustajaluokalla saattaa siis olla metodeja, jotka välitetään suoraan eteenpäin tai joiden parametreihin kohdistetaan joitakin operaatioita ennen kutsujen eteenpäin välittämistä.

```
class ProxySubject {
  var $subject;
  function ProxySubject() {
    $this->subject =& new Subject;
  }
  function someMethod() {
    $this->subject->someMethod();
  }
}
```

Kuva 44. Edustajaluokka, johon on lisätty metodin toteutus (Sweat, 2005).

Kuvassa 45 on esitetty kuvien 43-44 esimerkki UML-kaaviona.



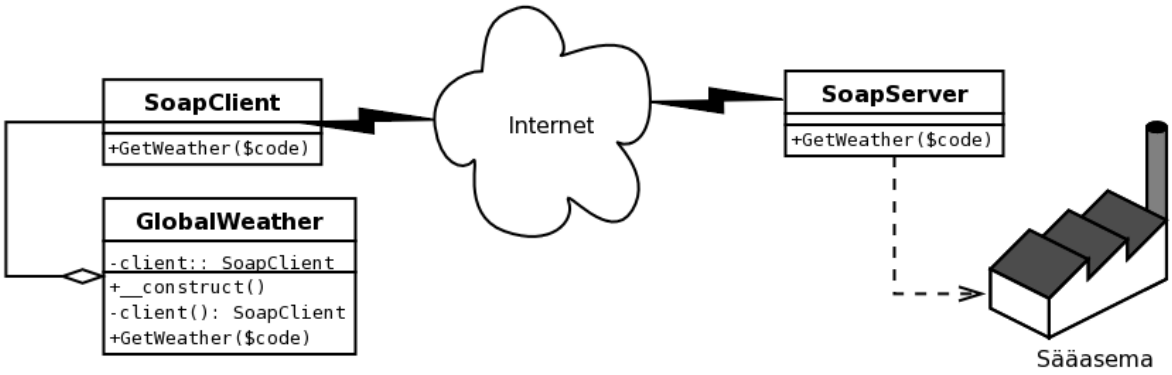
Kuva 45. Kuvien 43-44 esimerkki UML-kaaviona (Sweat, 2005).

5.2 Edustaja-toteutuksen vaihtoehdot

Seuraavassa esimerkissä esitetään Sweatin (2005) tarjoamaa Edustaja-mallia mukaileva SOAP-protokollaa hyödyntävä sovellus, joka käyttää Generic Objects Technologies Ltd:n (2007) tarjoamaa SOAP-säätietopalvelua. Esimerkissä käsitellään etäedustaja-tyyppistä sovellusta, sekä laiskaa ja dynaamista edustaja-toteutusta. Etäedustajaa käsittelevässä esimerkissä toteutettavaan luokkaan on sisällytetty Bakerin et al. (2005) toteuttamaa testiympäristöä käyttäviä testejä, joilla voidaan seurata käytetyn SOAP-palvelun toimivuutta ja mahdollisia muutoksia.

Esiteltävien Edustaja-mallin toteutuksien lähdekoodit löytyvät liitteestä 3. Liitteenä olevissa sovelluksissa ilmennetään mallin toimintaa joko käyttämällä Bakerin et al. (2005) yksikkötestauskirjastoa tai tulostamalla säätietoja ja kaupunkeja, joihin säätietoja on saatavilla. Jotta yksikkötestauskirjastoa voitaisiin käyttää, tulee se ladata osoitteesta www.simpletest.org.

Kuvassa 46 on esitetty laiskan edustajan toteutusta vastaava kaavio, joka kuvaa Edustaja-mallin toimintaa. Seuraavaksi esiteltävät Edustaja-mallin toteutukset noudattavat pääpiirteittäin kuvan 46 mukaista rakennetta, jossa edustajaluokka (kuvassa GlobalWeather) pitää hallussaan `SoapClient`-ilmentymää ja välittää kutsun eteenpäin säätietopalvelulle.



Kuva 46. Edustajaluokka sovelluksen osana.

5.2.1 *Etäedustaja*

Luokkaa `SoapClient` voidaan käsitellä etäedustaja-luokkana, koska se toimii välittäjänä sovelluksen ja palvelun välillä. Ensimmäisenä tulee tietysti luoda `SoapClient`-luokan ilmentymä käyttäen WSDL-tiedostoa:

```
$client =
  new soapclient('http://www.webservicex.com/globalweather.asmx?WSDL');
```

Jotta voitaisiin suojautua PHP:n tulevista päivityksistä aiheutuvilta muutoksilta tarvitaan testi, joka ilmaisee muutoksen, jos metodeja on lisätty tai poistettu. Testillä varmistutaan myös siitä, että käytetty PHP:n versio sisältää SOAP-tuen. Voidaan käyttää esimerkiksi metodia `get_class_methods(get_class($this->client))`, joka listaa luokan ilmentymän sisältä-

mät metodit. Kuvassa 47 on testifunktio, joka tarkistaa saatavilla olevien ja oletettujen metodien vastaavuuden. Tämä metodien vertailu voitaisiin kohdistaa myös pelkästään käytettyihin metodeihin, silloin käytettäisiin `in_array`-vertailua `assertEqual`-vertailun sijaan.

```
function TestMethodsOfSoapClient() {
    $soap_client_methods = array(
      '__construct',
      '__call',
      '__soapCall',
      '__getLastRequest',
      '__getLastResponse',
      '__getLastRequestHeaders',
      '__getLastResponseHeaders',
      '__getFunctions',
      '__getTypes',
      '__doRequest',
      '__setCookie',
      '__setLocation',
      '__setSoapHeaders');
    $this->assertEqual(
      $soap_client_methods,
      get_class_methods(get_class($this->client)));
}
```

Kuva 47. Testifunktio, joka varmistaa, että käytettyjen metodien lista on tiedossa.

SOAP-palvelun sisältämien ominaisuuksien tarkkailuun voidaan käyttää `SoapClient`-ilmentymän `__getFunctions()`-metodia, joka listaa palvelussa saatavilla olevat funktiot. Kuvassa 48 on esitetty testitapaus, jossa verrataan tiedossa olevaa SOAP-palvelun funktioiden listaa palvelun todellisuudessa sisältämiin funktioihin. Metodi `__getFunctions()` palauttaa merkkijonotaulukon, joka esittää web-palvelun rajapinnan, eli jokaisen metodin palautustyypin, metodin nimen ja odotettujen parametrien tyypin. Tämänkin testitapauksen tarkoitus on ilmaista muutokset web-palvelun rajapinnassa. Käyttämällä tällaista testiä voidaan välttyä virheen etsinnältä, kun web-palvelu odottamatta lopettaa toimintansa johtuen palvelun muuttuneesta rajapinnasta, josta ei olla tietoisia etukäteen.

```
function TestSoapFunctions() {
  $globalweather_functions = array(
    'GetWeatherResponse GetWeather(GetWeather $parameters)',
    'GetCitiesByCountryResponse GetCitiesByCountry(GetCitiesByCountry
                                                   $parameters)',
    'string GetWeather(string $CityName, string $CountryName)',
    'string GetCitiesByCountry(string $CountryName)',
    'string GetWeather(string $CityName, string $CountryName)',
    'string GetCitiesByCountry(string $CountryName)'
  );
  $this->assertEqual(
    $globalweather_functions,
    $this->client->__getFunctions());
}
```

Kuva 48. Testimetodi, joka varmistaa että palvelun tarjoamat funktiot ovat tiedossa.

Esimerkkikoodi RemoteProxy.php on esitelty liitteessä 3.

5.2.2 *Laiska edustaja*

Joskus saatetaan kohdata tilanne, jossa `SoapClient`-ilmentymän luomista ja WSDL-tiedoston prosessointia halutaan viivästyttää, kunnes ollaan varmoja siitä, että oliota todella tarvitaan. Metodin `GetCitiesByCountry()` tulisi muodostaa edustajametodi `$client`-ilmentymämuuttujan sisältämästä `GetCitiesByCountry()`-metodista. Kuitenkaan `SoapClient`-luokan ilmentymää ei ole vielä luotu ja talletettu `$client`-muuttujaan, koska tarkoituksena on viivästyttää WSDL-tiedoston prosessointia, kunnes sitä todella tarvitaan. Luokan `SoapClient` ilmentymän muodostamista voidaan viivästyttää asettamalla väliin laiskan latauksen koodia ennen asiakaskutsun suorittamista. Kuten kuvasta 49 ilmenee, metodi `lazyLoad()` luo `SoapClient`-ilmentymän pyynnöstä.

```
class GlobalWeather {
  private $client;
  private function lazyLoad() {
    if (! $this->client instanceof SoapClient) {
       $this->client = new
         SoapClient('http://www.webservicex.com/globalweather.asmx?WSDL');
    }
  }
  public function GetCitiesByCountry($code) {
    $this->lazyLoad();
    return $this->client->GetCitiesByCountry($code);
  }
}
```

Kuva 49. Laiskan ilmentämisen toteutus.

Voidaan myös käyttää PHP:n ominaisuutta palautettujen olioiden metodikutsujen ketjuttamiseksi, jotta säästytään kaikkien edustajaluokkaan sisällytettävien metodien luomiselta ja `$this->lazyLoad()`-rivin lisäämiseltä jokaiseen edellä mainittuun metodiin. Annetaan metodille `lazyLoad()` nimeksi `client()` attribuutin `$client` sijaan. Kuvassa 50 on esimerkki oikeaoppisesta laiskan ilmentämisen toteutuksesta. Kuvassa 46 esitetty kaavio vastaa tämän tyyppistä laiskan ilmentämisen toteutusta.

```
class GlobalWeather {
  private function client() {
    if (! $this->client instanceof SoapClient) {
       $this->client = new
        SoapClient('http://www.webservicex.com/globalweather.asmx?WSDL');
    }
    return $this->client;
  }
  public function GetWeather($code) {
    return $this->client()->GetWeather($code);
  }
}
```

Kuva 50. Vaihtoehtoinen toteutus laiskalle ilmentämiselle.

Esimerkkikoodi LazyProxy.php on esitelty liitteessä 3.

5.2.3 *Dynaaminen edustaja*

PHP tarjoaa ominaisuuksia, jotka mahdollistavat edustajaluokan luomisen ilman, että jokaista metodia tarvitsee kirjoittaa eksplisiittisesti (kuva 51).

```
class GenericProxy {
  protected $subject;
  public function __construct($subject) {
    $this->subject = $subject;
  }
  public function __call($method, $args) {
    return call_user_func_array(array($this->subject, $method),$args);
  }
}
```

Kuva 51. Dynaamisen edustajan toteutus.

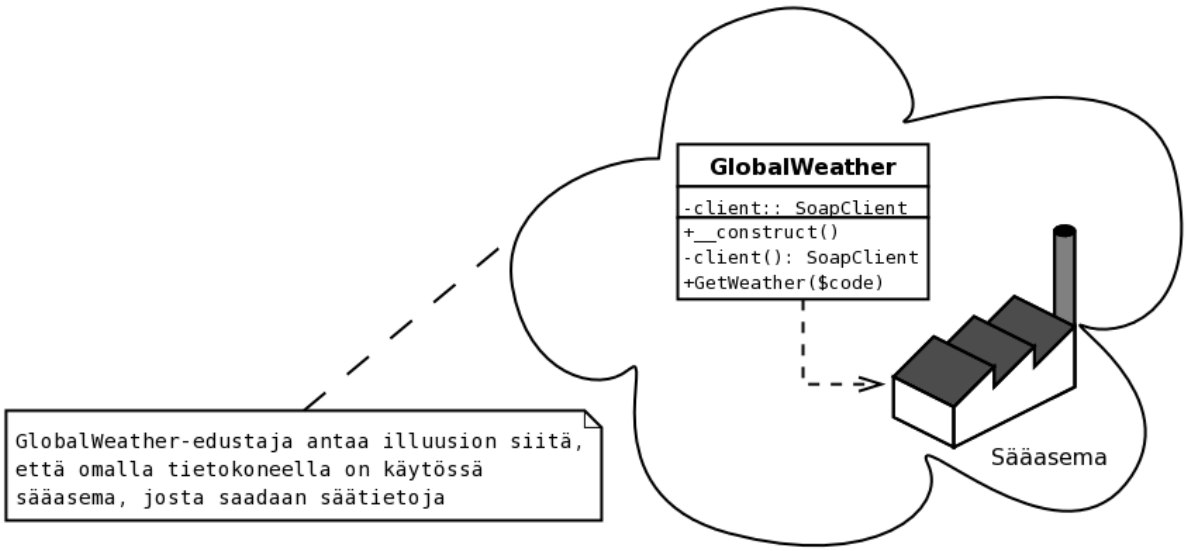
Avainasemaassa on `__call()`-metodi, joka mahdollistaa jokaisen edustajaluokan kutsun uudelleenohjauksen `$subject`-muuttujalle. Koska `__call()`-metodilla on matalampi prioriteetti kuin todellisilla metodeilla, voidaan määrittää edustajaluokan metodi, joka suoritetaan `__call()`-metodin sijaan. Tätä rakennetta voidaan käyttää perustana edustajaluokalle ja lisätä tarvittavat ominaisuudet Edustaja-mallin toteutusta varten.

Esimerkkikoodi GenericProxy.php on esitelty liitteessä 3.

5.3 Vaikutus ohjelmoijan näkökulmaan

Kuten edellä demonstroiduissa edustajaluokkien toteutuksissa, Edustaja-mallin avulla voidaan käyttää etäpalveluita aivan kuin ne sijaitsisivat paikallisesti omalla tietokoneella. Tällainen näkökulma helpottaa ohjelmoijan työtä, koska ei tarvitse miettiä liikaa web-palveluiden toteutuksen tai rajapintojen yksityiskohtia. Tämän tyyppinen toiminta on yleensäkin suotavaa,

kun toimitaan olio-ohjelmoinnin periaatteiden mukaan. Kuvan 52 kaavio kuvastaa edustaja-luokan vaikutusta ohjelmoijan näkökulmaan kuvan 46 sovellusta tarkasteltaessa.



Kuva 52. Palvelu ohjelmoijan näkökulmasta Edustajan avulla ilmaistuna (Sweat, 2005).

6 Yhteenveto

WSDL tarjoaa palvelun kehittäjälle tavan kuvata palvelun rakenteen tärkeimmät ominaisuudet siten, että palvelun käyttäjät voivat käyttää niitä kohtuullisen vähällä vaivalla. Luvussa 2 käytiin läpi tärkeimmät WSDL-kielen ominaisuudet ja niiden soveltaminen SOAP-protokollan yhteydessä. Vaikka WSDL-kieli saattaa vaikuttaa alkuun hyvinkin sekavalta, ei siitä tule hämmentyä, sillä WSDL-tiedostojen luomiseen on tarjolla sovelluksia, jotka helpottavat palvelujen kuvaamista. On olemassa myös paljon esimerkkidokumentteja, joiden pohjalta omaa sovellusta on helppo lähteä rakentamaan.

SOAP on protokolla, joka mahdollistaa sovellustenvälisen viestien välittämisen verkon yli ja joka oikein käytettynä on tehokas väline toimintojen hajauttamiseen ja web-palveluiden toteutukseen. Luvussa 3 käytiin läpi SOAP-viestien rakennetta, sekä virhetilanteiden käsittelyä. Koska SOAP on alustariippumaton protokolla, voidaan eri ohjelmointikielillä toteutettuja palveluja käyttää millä tahansa SOAP-protokollaa hyödyntävällä sovelluksella. Vaikka SOAP:ia voidaankin käyttää ohjelmointikielien tarjoamien rajapintojen kautta, on hyvä tietää protokollan toiminnasta myös viestien tasolla siltä varalta, jos joskus joudutaan muuttamaan SOAP-viestejä manuaalisesti. Protokollan tuntemus viestien tasolla on erittäin tärkeää myös palveluita kehitettäessä ja virheitä jäljitettäessä, koska pienikin virhe SOAP-viestissä voi estää ohjelmointikielen tarjoaman rajapinnan toiminnan.

WSDL-kielen tuntemus helpottaa huomattavasti SOAP-protokollan käyttöä, sillä WSDL-dokumentissa on määritelty tärkeimmät tiedot, joita onnistuneeseen SOAP:in käyttöön vaaditaan. Kun sekä asiakas- että palvelinsovellus käyttävät WSDL-dokumenttia palvelun rajapinnan määrittämiseen, varmistutaan siitä, että kumpikin osapuoli käyttää samaa rajapintaa SOAP-viestien lähettämiseen ja vastaanottamiseen. Jos ollaan tekemisissä palvelinsovelluksen kanssa, on WSDL:n tuntemus välttämätöntä, mutta jos ollaan toteuttamassa asiakassovellusta, pärjätään myös ilman WSDL-kielen osaamista. Jos käytetään todella yksin-

kertaista palvelua, voidaan rajapinta määritellä jonkin muun kuin WSDL-kielisen dokumentaation perusteella, mutta tällaisessa tapauksessa tulee palvelun tarjoajalla olla todella hyvä perustelu WSDL-dokumentin puuttumiselle.

Jotta SOAP:ia voitaisiin hyödyntää tehokkaasti PHP:llä, tulee tietää joitakin SOAP:iin, WSDL-kieleen, sekä PHP:n SOAP-tuen käyttöön liittyviä asioita. Luvussa 4 käytiin läpi SOAP:in käyttöä PHP:llä sekä palvelin- että asiakassovelluksen näkökulmasta. SOAP-protokollan hyödyntäminen PHP-ohjelmoinnissa on melko tuore ilmiö, jos PHP:tä verrataan muihin ohjelmointikieliin SOAP-tuen osalta. PHP:n natiivi SOAP-tuki tarjoaa helppokäyttöisiä välineitä palvelun käyttöön, tarkasteluun sekä virheiden jäljitykseen, joten onnistuneeseen SOAP:in käyttöön riittää, kun osaa soveltaa muutamaa tärkeintä toiminnallisuutta. Vaikka palvelimelle asennetussa PHP-versiossa ei olisi natiivia SOAP-tukea mukana, voidaan käyttää ulkoisia kirjastoja, jotka mahdollistavat SOAP-protokollan käytön.

Luvussa 5 esitettiin SOAP-protokollaa hyödyntävä Edustaja-suunnittelumallin toteutus. Suunnittelumallit liittyvät vahvasti olio-ohjelmointiin. Esitellyn Edustaja-mallin tarkoituksena oli tarjota tapa SOAP-protokollan hyödyntämiseen olioperustaisessa PHP-ohjelmoinnissa. Vaikka PHP onkin alunperin tarkoitettu yksinkertaiseksi skriptauskieleksi, jolla voidaan luoda dynaamiikkaa web-sivuille, on siitä muotoutunut kieli, joka mahdollistaa olio-ohjelmoinnin periaatteiden mukaisen ohjelmoinnin.

Koska SOAP on käytettävästä ohjelmointikielestä riippumaton, voidaan sen avulla sovellusten yhteentoimivuutta parantaa ja näin saavuttaa tehokkaampia kokonaisuuksia. Web-palvelut ovat tulleet jäädäkseen, joten esimerkiksi termeihin SOAP, WSDL ja UDDI tutustuminen on ohjelmistotuotannon ammattilaisille vähintäänkin suotavaa.

Viitteet

Ayala D., Browne C., Chopra V., Sarang P., Apshankar K., McAllister T. (2002) *Professional Open Source Web Services.*Wrox Press.

Baker M., Fuecks H., Sweat J. E. (2006) *Simple Test PHP Unit Test Framework.* WWW-sivusto, http://www.simpletest.org/ (31.5.2007).

Braam P. (2004) *Design Patterns applied to Web Programming in PHP.* http://www.cs.vu.nl/~pajbraam/Essay_OOP.pdf (31.5.2007).

Butek R. (2005) *Which style of WSDL should I use?* WWW-sivusto, http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/ (3.6.2007).

Erl T. (2004) *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services.* Prentice Hall.

Generic Objects Technologies Ltd (2007) *webserviceX.NET.* WWW-sivusto, http://www.webservicex.net (31.5.2007).

Gilmore W. J. (2006) *Beginning PHP and MySQL 5: From Novice to Professional.* Apress.

Goel S., Seshan P. (2006a) *Web services integration patterns for Java applications using open source frameworks, Part 1: Implementing invoke patterns.* WWW-sivusto, http://www.ibm.com/developerworks/webservices/library/ws-pattern-open1.html (29.6.2007).

Goel S., Seshan P. (2006b) *Web services integration patterns for Java applications using open source frameworks, Part 2: Implementing receive patterns.* WWW-sivusto, http://www.ibm.com/developerworks/webservices/library/ws-pattern-open2.html (29.6.2007).

Newcomer E. (2002) *Understanding Web Services- XML, WSDL, SOAP and UDDI.* Addison-Wesley Professional.

Newcomer E., Lomow G. (2005) *Understanding SOA with Web Services.* Addison-Wesley Professional.

NuSOAP (2007) WWW-sivusto, http://dietrich.ganx4.com/nusoap (31.5.2007).

PHP Group (2007a) *PEAR - PHP Extension and Application Repository.* WWW-sivusto, http://pear.php.net/package/SOAP (31.5.2007)

PHP Group (2007b) *PHP-manual.* WWW-sivusto, http://www.php.net/manual/en/ (31.5.2007).

Richards R. (2006) *Pro PHP XML and Web Services.* Apress, USA.

Rikala P. (2006) *PHP ja suunnittelumallit.* Kandidaatintutkielma. Joensuun yliopisto, Tietojenkäsittelytieteen laitos.

Sweat J. E. (2005) *PHP-Architect's Guide to PHP Design Patterns*. Marco Tabini & Associates, Inc. Canada, Toronto.

W3C (2000) *Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000*. WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (31.5.2007)

W3C (2001) *Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001.* WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/wsdl (31.5.2007)

W3C (2004) *Web Services Architecture, W3C Working Group Note 11 February 2004.* WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/ws-arch/ (31.5.2007)

W3C (2007a) *SOAP Version 1.2.* WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/soap12 (31.5.2007)

W3C (2007b) *XML Schema.* WWW-sivusto, World Wide Web Consortium, http://www.w3.org/XML/Schema (31.5.2007)

Weerawarana S., Curbera F., Leymann F., Storey T, Ferguson D. F. (2005) *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall PTR.

WS-I (2006) *Web Services Interoperability Organization.* WWW-sivusto, http://www.ws-i.org/ (31.5.2007)

LIITE 1: WSDL-dokumentti

WSDL-dokumentti (Richards, 2006): exampleapi.wsdl

```
<?xml version="1.0"?>
<definitions targetNamespace="urn:ExampleAPI" xmlns:tns="urn:ExampleAPI"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
      <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="urn:ExampleAPI">
         <xsd:element name="getPeopleByFirstLastName">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="first" type="xsd:string"/>
                  <xsd:element name="last" type="xsd:string"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>

         <xsd:complexType name="Person">
            <xsd:all>
               <xsd:element name="id" type="xsd:int"/>
               <xsd:element name="lastName" type="xsd:string"/>
               <xsd:element name="firstName" type="xsd:string"/>
            </xsd:all>
         </xsd:complexType>

         <xsd:complexType name="ArrayOfPerson">
            <xsd:complexContent>
               <xsd:restriction base="soapenc:Array">
                  <xsd:attribute ref="soapenc:arrayType"
                                 wsdl:arrayType="tns:Person[]"/>
               </xsd:restriction>
            </xsd:complexContent>
         </xsd:complexType>

         <xsd:element name="getPeopleByFirstLastNameResponse"
                      type="tns:ArrayOfPerson"/>

         <xsd:element name="DBUnavailableFault">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="DBMessage" type="xsd:string"/>
                  <xsd:element name="RetryInMinutes" type="xsd:int"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>

         <xsd:element name="SystemMaintenance">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="SysMessage" type="xsd:string"/>
                  <xsd:element name="RetryInMinutes" type="xsd:int"/>
               </xsd:sequence>
```

```
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>

<!-- Syöteviesti -->
<message name="getPeopleByFirstLastName">
   <part name="parameters" element="tns:getPeopleByFirstLastName"/>
</message>
<!-- Tulosteviesti -->
<message name="getPeopleByFirstLastNameResponse">
   <part name="result" element="tns:getPeopleByFirstLastNameResponse"/>
</message>
<!-- Virheviestit -->
<message name="DBUnavailableFault">
   <part name="DBUnavailableFault" element="tns:DBUnavailableFault"/>
</message>
<message name="SystemMaintenance">
   <part name="SystemMaintenance" element="tns:SystemMaintenance"/>
</message>

<!-- Portti esimerkkirajapinnalle -->
<portType name="ExamplePortType">
   <operation name="getPeopleByFirstLastName">
      <input message="tns:getPeopleByFirstLastName"/>
      <output message="tns:getPeopleByFirstLastNameResponse"/>
      <fault name="nodb" message="tns:DBUnavailableFault"/>
      <fault name="sysmaint" message="tns:SystemMaintenance"/>
   </operation>
   <!-- Muut operaatiot -->
</portType>

<!-- Sidonta esimerkkirajapinnalle. Kooditus: document/literal,
     SOAP-viestit välitetään HTTP:n yli                          -->
<binding name="ExampleBinding" type="tns:ExamplePortType">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="getPeopleByFirstLastName">
      <soap:operation soapAction="getPeopleByFirstLastName"/>
      <input>
         <soap:body use="literal"/>
      </input>
      <output>
         <soap:body use="literal"/>
      </output>
      <fault name="nodb">
         <soap:fault name="nodb" use="literal"/>
      </fault>
      <fault name="sysmaint">
         <soap:fault name="sysmaint" use="literal"/>
      </fault>
   </operation>
   <!-- Muut operaatiot -->
</binding>

<!-- Päätepisteet esimerkkirajapinnalle -->
<service name="ExampleService">
   <port name="ExamplePort" binding="tns:ExampleBinding">
      <soap:address location="http://www.example.com/ExampleService"/>
   </port>
```

```
   </service>
</definitions>
```

LIITE 2: Asiakas-palvelin-sovellus

Palvelimen koodi (Richards, 2006): server.php

```
<?php
/* Systeemin tila - TRUE : Normaali tila          /
                   FALSE : Palvelu pois käytöstä */
$SYS_STATUS = TRUE;

function getPeopleByFirstLastName($getPeopleByFirstLastName) {
   /* Jos systeemi on pois käytöstä heitetään SoapFault */
   if (isset($GLOBALS['SYS_STATUS']) && $GLOBALS['SYS_STATUS'] == FALSE) {
      $details = array("SysMessage"=>"Sys Error", "RetryInMinutes"=>60);
      throw new SoapFault("SYSError", "System Unavailable",
                          "urn:ExampleAPI",$details, "sysmaint");
   }

   /* Asetetaan henkilöiden tiedot */
   $people = array(array('id'=>1, 'firstName'=>'John','lastName'=>'Smith'),
                   array('id'=>2, 'firstName'=>'Jane', 'lastName'=>'Doe'));

   $firstSearch = str_replace('*', '([a-z]*)',
                              $getPeopleByFirstLastName->first);
   $lastSearch = str_replace('*', '([a-z]*)',
                             $getPeopleByFirstLastName->last);

   $retval = array();

   foreach($people AS $person) {
      /* Tarkistetaan täsmääkö etunimi */
      if (empty($firstSearch) || preg_match('/^'.$firstSearch.'$/i',
                                            $person['firstName']))
      {
         /* Tarkistetaan täsmääkö sukunimi */
         if (empty($lastSearch) || preg_match('/^'.$lastSearch.'$/i',
                                              $person['lastName']))
         {
            /* Lisätään löydetyt tiedot SoapVar-kooditettuna */
            $retval[] = new SoapVar($person, SOAP_ENC_ARRAY, "Person",
                                    "urn:ExampleAPI");
         }
      }
   }

   return $retval;
}

/* Luodaan palvelin käyttäen WSDL-tiedostoa ja määritellään actor- /
   attribuutille URI                                                */
$sServer = new SoapServer("exampleapi.wsdl",
array('actor'=>'urn:ExampleAPI'));

/* Rekisteröidään getPeopleByFirstLastName-funktio */
$sServer->addFunction("getPeopleByFirstLastName");

/* Käsitellään SOAP-pyyntö */
$sServer->handle();
```

```
?>
```

Asiakassovelluksen koodi (Richards, 2006): client.php

```
<?php
try {
   $sClient = new SoapClient('exampleapi.wsdl');

   /* Asetetaan hakuparametrit */
   $params = array('first'=>'jo*', 'last'=>'*');

   /* Tehdään pyyntö ja tulostetaan vastaus */
   $response = $sClient->getPeopleByFirstLastName($params);
   var_dump($response);
} catch (SoapFault $e) {
   /* Tulostetaan kiinni otettu poikkeus */
   var_dump($e);
  }
?>
```

LIITE 3: Edustaja-suunnittelumallin soveltaminen

Esimerkit tehty Sweatin (2005) esittämien esimerkkien pohjalta.

RemoteProxy.php

```
<?php


require_once 'simpletest/unit_tester.php';
require_once 'simpletest/reporter.php';

class ProxyTestCase extends UnitTestCase {
  const WSDL = 'http://www.webservicex.com/globalweather.asmx?WSDL';
  private $client;
  function setUp() {
    $this->client = new SoapClient(ProxyTestCase::WSDL);
  }

  function TestMethodsOfSoapClient() {
    $soap_client_methods = array(
      '__construct',
      '__call',
      '__soapCall',
      '__getLastRequest',
      '__getLastResponse',
      '__getLastRequestHeaders',
      '__getLastResponseHeaders',
      '__getFunctions',
      '__getTypes',
      '__doRequest',
      '__setCookie',
      '__setLocation',
      '__setSoapHeaders');
    $this->assertEqual(
      $soap_client_methods,
      get_class_methods(get_class($this->client)));
  }

  function TestSoapFunctions() {
    $globalweather_functions = array(
      'GetWeatherResponse GetWeather(GetWeather $parameters)',
      'GetCitiesByCountryResponseGetCitiesByCountry(
                                    GetCitiesByCountry $parameters)',
      'string GetWeather(string $CityName, string $CountryName)',
      'string GetCitiesByCountry(string $CountryName)',
      'string GetWeather(string $CityName, string $CountryName)',
      'string GetCitiesByCountry(string $CountryName)' );

    $this->assertEqual(
      $globalweather_functions,
      $this->client->__getFunctions());
  }
}

// ajetaan testi
```

```
$test = new ProxyTestCase('Proxy - SOAP - Sovellus');

$test->run(new HtmlReporter());

$client = new
soapclient('http://www.webservicex.com/globalweather.asmx?WSDL');

$class_methods = get_class_methods(get_class($client));
$client_functions = $client->__getFunctions();
$client_types = $client->__getTypes();


echo "<b>Luokan SOAP metodit: <br></b>";
foreach ($class_methods as $method_name) {
  echo "$method_name <br>";
}

echo "<b><br>Palvelun funktiot: <br></b>";
foreach ($client_functions as $function_name) {
 echo "$function_name <br>";
}

echo "<b><br>Palvelun tyypit: <br></b>";
foreach ($client_types as $function_type) {
 echo "$function_type <br>";
}

$haku1 = array(CountryName => 'finland', CityName=> 'joensuu');

try{
  $saa = $client->GetWeather($haku1);
} catch (SoapFault $e) {
        echo $e->faultstring;
  }

echo "<b><br> Joensuun sää: <br></b>";
echo "$saa->GetWeatherResult <br>";


$haku2 = array(CountryName => 'finland');

try{
  $saa2 = $client->GetCitiesByCountry($haku2);
} catch (SoapFault $e) {
        echo $e->faultstring;
  }

echo "<b><br>Kaupungit, joihin säätiedot on saatavilla:<br></b>";
echo "$saa2->GetCitiesByCountryResult";
```

LazyProxy.php

```
<?php

class GlobalWeather {
  private $client;

  private function client() {
    if (! $this->client instanceof SoapClient) {
      $this->client = new SoapClient(
                    'http://www.webservicex.com/globalweather.asmx?WSDL');
    }
    return $this->client;
  }

  public function GetCitiesByCountry($code) {
    return $this->client()->GetCitiesByCountry($code);
  }

  public function GetWeather($code) {
    return $this->client()->GetWeather($code);
  }
}


$gw = new GlobalWeather();

$haku1 = array(CountryName => 'finland', CityName=> 'joensuu');

try{
  $saa = $gw->GetWeather($haku1);
} catch (SoapFault $e) {
        echo $e->faultstring;
  }

echo "<b><br> Joensuun sää: <br></b>";
echo "$saa->GetWeatherResult <br>";

$haku2 = array(CountryName => 'finland');

try{
  $saa2 = $gw->GetCitiesByCountry($haku2);
} catch (SoapFault $e) {
        echo $e->faultstring;
  }

echo "<b><br>Kaupungit, joihin säätiedot on saatavilla:<br></b>";
echo "$saa2->GetCitiesByCountryResult";
```

GenericProxy.php

```
<?php

class GenericProxy {
  protected $subject;

  public function __construct($subject) {
    $this->subject = $subject;
  }

  public function __call($method, $args) {
    return call_user_func_array(array($this->subject, $method),$args);
  }
}


$gw = new GenericProxy(new SoapClient(
                    'http://www.webservicex.com/globalweather.asmx?WSDL'));

$haku1 = array(CountryName => 'finland', CityName=> 'joensuu');

try{
  $saa = $gw->GetWeather($haku1);
} catch (SoapFault $e) {
        echo $e->faultstring;
  }

echo "<b><br> Joensuun sää: <br></b>";
echo "$saa->GetWeatherResult <br>";

$haku2 = array(CountryName => 'finland');

try{
  $saa2 = $gw->GetCitiesByCountry($haku2);
} catch (SoapFault $e) {
        echo $e->faultstring;
  }

echo "<b><br>Kaupungit, joihin säätiedot on saatavilla:<br></b>";
echo "$saa2->GetCitiesByCountryResult";
```