Web-sovelluksen laajentaminen ulkoisilla web-palveluilla

Mika Kinnunen

12.6.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Internet-sivustot tarjoavat monia web-palveluja, joita voidaan hyödyntää web-sovelluksissa. Valmiit web-palvelut tarjoavat keinoja laajentaa omien web-sovellusten toimintaa sellaisilla toiminnoilla ja palveluilla, joiden toteuttaminen voisi muodostua muuten hankalaksi. Tutkielmassa esitellään Googlen, Amazonin ja eBayn tarjoamia web-palvelurajapintoja. Web-palvelurajapintojen käyttöä havainnollistetaan Googlen Google Data API -rajapinnan avulla.

Web-palvelujen suuri määrä aiheuttaa myös ongelmia, mikäli niitä halutaan hyödyntää samanaikaisesti. Tällöin joudutaan huomioimaan jokaisen web-palvelun erilaiset rajapinnat ja toimintaperiaatteet. Tämä lisää helposti asiakassovellusten ohjelmointiin tarvittavaa työmäärää ja rajoittaa näin ollen web-palvelujen hyödyntämistä. Tämän ongelman ratkaisua tarkastellaan välityspalvelurajapintamallin kautta, joka konkretisoidaan esiteltävässä WEB-API-web-palvelussa.

ACM-luokat (ACM Computing Classification System, 1998 version): C.2.4, D.2.11, D.3.2, H.3.5

Avainsanat: Web-palvelu, Web API, Google Data API, välityspalvelurajapinta

Sisältö

1 Johdanto

Web-sovellukset tarjoavat mahdollisuuden tuottaa Internet-selaimilla käytettäviä yhteiskäyttöön soveltuvia sovelluksia. Web-sovellukset ovat olleet itsenäisiä kokonaisuuksia, jotka hallinnoivat oman kontekstinsa itsenäisesti. Web-sovellusten rinnalle on tullut myös web-palveluja (Weerawarana & al., 2005), jotka tarjoavat palvelujaan web-sovelluksille tietoverkon yli. Tämä mahdollistaa hajautetut palvelukeskeiset *SOA* (Service-Oriented Architectures) -arkkitehtuurit (Bih J., 2006), missä web-sovellukset voivat tarjota käyttäjilleen myös web-palvelujen tuottamaa toiminnallisuutta.

Monet Internet-sivustot tarjoavat web-palveluja (Gosnell & Denise, 2005), joita voidaan hyödyntää työpöytä- ja web-sovelluksissa. Palvelujen tarjonta on runsasta ja Internetistä löytyykin web-palveluja useisiin tarkoituksiin. Tällaisia ovat muun muassa erilaiset uutispalvelut, karttapalvelut ja kaupankäyntipalvelut. Valmiit web-palvelut tarjoavat keinoja laajentaa omien web-sovellusten toimintaa sellaisilla toiminnoilla ja palveluilla, joiden toteuttaminen voisi muodostua työlääksi tai jopa mahdottomaksi, mikäli ei ole käytettävissä riittäviä resursseja tai riittävää tietotaitoa. Tässä mielessä valmiit web-palvelut voivat tarjota helpohkon ja nopean tavan täydentää web-sovellusten tarjoamaa toiminnallisuutta. Esimerkiksi verkkokaupan rakentaminen helpottuu, kun maksuliikenne ja jopa itse verkkokaupan toiminnallisuus voidaan toteuttaa valmiilla palveluilla. Tutkielmassa esitellään Googlen, Amazonin ja eBayn tarjoamia web-palveluja ja palvelurajapintoja, joista tutustutaan tarkemmin Googlen tarjoamiin palveluihin.

Erilaisten web-palvelujen suuri määrä aiheuttaa myös ongelmia, mikäli niitä halutaan hyödyntää samanaikaisesti. Tällöin joudutaan huomioimaan jokaisen web-palvelun erilaiset rajapinnat ja toimintaperiaatteet. Tämä lisää helposti asiakassovellusten ohjelmointiin tarvittavaa työmäärää ja rajoittaa näin ollen web-palvelujen hyödyntämistä. Tämän ongelman ratkaisua tarkastellaan tässä tutkielmassa koostepalvelun (Kung-Kiu & Cuong, 2007) ja ns. koostettavan välityspalvelurajapinnan kautta, joka tarjoaa ulkopuolisten web-palvelujen palveluja koostetusti oman web-palvelurajapintansa kautta. Tutkielmassa esitellään myös tähän liittyvä välityspalvelurajapintamallia hyödyntävä WEB-API-web-palvelu (Kinnunen, 2008). Ennen kuin web-palvelujen käyttöä voidaan esitellä tulee olla tiedossa niihin liittyvät perusteet. Tästä syystä esitellään aluksi web-palvelujen perusteita ja web-palveluihin läheisesti liittyviä tekniikoita.

Luvussa 2 esitellään yleisesti web-palvelujen toimintaperiaatetta sekä web-palvelujen käyttämiä tekniikoita. Ensimmäiseksi tarkastellaan web-palvelun perusteita sekä web-palveluihin liittyviä yleisiä termejä. Tämän jälkeen kuvataan web-palvelujen määrityksessä käytettävää WSDL 1.1. -suositusta (W3C, 2001). Tämän jälkeen tarkastellaan web-palveluihin läheisesti liittyvää SOAP-viestimuotoa, josta tarkastellaan SOAP 1.1 -versiota (W3C, 2000). WSDL:n ja SOAP:n jälkeen kerrotaan, miten SOAP-protokolla liitetään osaksi WSDL-määritystä. Lopuksi kuvataan lyhyesti eri menetelmiä, joilla web-palveluja voidaan kutsua.

Luvussa 3 tuodaan esille valittuja web-palveluja ja niiden tarjoamia web-palvelurajapintoja. Tämän lisäksi esitellään myös erityinen välityspalvelurajapintamalli, jolla voidaan koostaa web-palveluja. Luvussa selvitetään ensin web-palvelurajapintakäsitettä yleisellä tasolla. Tämän jälkeen kuvataan Googlen, Amazonin ja eBayn tarjoamia web-palvelurajapintoja sekä vertaillaan niiden hyödyntämiseksi tarjottuja aputoimintoja. Tämän jälkeen esitellään tarkemmin Googlen tarjoamat web-palvelurajapinnat, jonka jälkeen niiden käyttöä havainnollistetaan käytännön esimerkeillä. Esittelyssä käytetään apuna Googlen tarjoamia asiakaskirjastoja. Lopuksi kerrotaan yleisesti web-palvelujen koostamisesta välityspalvelurajapintamallilla. Samalla tuodaan esille välitypalvelurajapintapalvelujen koostamiseen liittyviä ongelmia sekä tarjotaan joitakin Busslerin (2006) esittelemiä menetelmiä niiden ratkaisemiseksi.

Luvussa 4 esitellään *JAX-WS* (Java API for XML Web Services) -tekniikalla (Chinnici & al., 2005) toteutettu välityspalvelurajapinnan mukainen web-palvelu: WEB-API. WEB-API-web-palvelusta esitellään sekä palvelun rakenne että sen tarjoamat palvelut. Ensiksi kuvataan WEB-API-palvelun rakenne käsitteellisellä tasolla, joka konkretisoidaan toiminnan kannalta olennaisilla osilla. Palvelun rakenteen jälkeen kuvausta tarkennetaan WEB-API:n palvelujen toteutusteknisillä piirteillä, jotka muodostavat WEB-API:n perustan. Lopuksi kuvataan WEB-API:n tarjoamat palvelut ja niiden käyttämiseksi tarjotut web-palvelurajapinnat. Näistä esitellään ensiksi Googlen *tunnistuspalvelu* (authentication service), jota käytetään Googlen palveluihin tunnistauduttaessa. Tämän jälkeen kuvataan Googlen Picasa Web Albums -palvelu.

Luvussa 5 tehdään yhteenveto tutkielmassa käsitellyistä asioista. Samalla tarkastellaan sekä API- että välitypalvelurajapintojen käyttökelpoisuutta ja tuodaan esille näihin liittyviä mahdollisia ongelmakohtia. Tarkastelua konkretisoidaan muun muassa Googlen palvelujen ja WEB-API-välityspalvelurajapinnan avulla.

2 Web-palvelut

Tässä luvussa esitellään web-palvelu ja web-palveluun läheisesti sidoksissa olevat standardit ja protokollat. Ensiksi tarkastellaan web-palvelun määritelmää abstraktimmalla tasolla, samalla kartoittaen web-palveluun liittyviä taustatekijöitä. Tämän jälkeen esitellään web-palveluun läheisesti kytköksissä olevat WSDL, SOAP ja SOAP-sidonta. Näistä kuvataan ensin WSDL-dokumentin ja SOAP-viestin rakenteet ja niiden sisältämät elementit. Tämän jälkeen esitellään SOAP-sidonta, jolla SOAP-protokolla sidotaan osaksi web-palvelun WSDL-määritystä. Lopuksi esitellään vielä lyhyesti REST-menetelmän käyttöä.

2.1 Web-palvelun määritelmä

Tässä kohdassa esitellyt tiedot pohjautuvat W3C:n web-palvelujen arkkitehtuurikuvaukseen (W3C, 2004a) tai ovat siitä johdettuja.

W3C määrittää web-palvelun seuraavasti: Web-palvelulla tarkoitetaan ohjelmistojärjestelmää, joka on suunniteltu tukemaan yhteentoimivaa koneelta koneelle vuorovaikutusta tietoverkon yli. Web-palvelu sisältää koneluettavassa muodossa olevan *rajapinnan* (interface). Muut järjestelmät kommunikoivat web-palvelun kanssa käyttäen *SOAP* (Simple Object Access Protocol) -viestejä. Viestien välityksessä käytetään tyypillisesti *HTTP* (HyperText Transport Protocol) -protokollaa ja *XML (eXtensible Markup Language) -sarjoitusta* (XML serialization) yhdistettynä muihin web-standardeihin.

Kun web-palvelua tarkastellaan abstraktina käsitteenä, se voidaan tulkita resurssiksi, joka määrittää web-palvelun tarjoamat toiminnot. Web-palvelun toiminnot toteuttavaa ohjelmistoa tai tietokonelaitteistoa kutsutaan *konkreettiseksi agentiksi* (concrete agent). Web-palvelu muodostaa siis käsitteenä abstraktin rajapinnan ja agentti rajapinnan toteuttavan toteutuksen. Täten yhtä web-palvelua kohden voi olla useita eri toteutuksia, jotka kaikki toteuttavat saman toiminnallisuuden.

Web-palvelun tarkoituksena on tarjota toiminnallisuutta omistajansa puolesta, joka voi olla henkilö tai organisaatio. Web-palvelun agentin tarjoavaa henkilöä tai organisaatiota kutsutaan käsitteellisesti *tarjoaja-entiteeksi* (provider entity). Web-palvelua käyttävää henkilöä tai organisaatiota kutsutaan puolestaan *pyytäjä-entiteetiksi* (requester entity). Tarjoaja-entiteetin viestinvälityksestä vastaa *tarjoaja-agentti* (provi-

der agent), joka kommunikoi pyytäjä-entiteetin viestinvälityksestä vastaavan *pyytäjä-agentin* (requester agent) kanssa. Edellisen nojalla voimme kutsua tarjoaja-entiteettiä yleisemmin *palvelun tarjoajaksi* ja pyytäjä-entiteettiä *palvelun asiakkaaksi*. Vastaavasti voimme kutsua tarjoaja-agenttia *web-palveluksi* ja pyytäjä-agenttia *web-palvelun asiakkaaksi*. Jatkossa entiteettien ja agenttien sijasta käytetään näitä termejä, ellei haluta ilmaista nimenomaisesti palveluun tai sen käyttöön liittyvää omistajuussuhdetta, jota entiteeteillä ja agenteilla on tavoiteltu. Jotta palvelun asiakas voisi käyttää web-palvelun tarjoamaa toiminnallisuutta, täytyy seuraavien ehtojen toteutua:

1. Palvelun tarjoaja ja palvelun asiakas ovat tietoisia toisistaan.
2. Palvelun tarjoaja ja palvelun asiakas jakavat palvelun kuvauksen ja semantiikat, jotka käsittävät web-palvelun ja web-palvelun asiakkaan välisen vuorovaikutuksen.
3. Web-palvelu ja web-palvelun asiakas toteuttavat palvelun kuvauksen ja semantiikat.
4. Web-palvelu ja web-palvelun asiakas vaihtavat viestejä suorittaen samalla tehtäviä palvelun tarjoajan ja asiakkaan puolesta.

Edellä mainituista ehdoista osa voidaan automatisoida, kun taas toiset joudutaan suorittamaan manuaalisesti. Lisäksi ehtoja saatetaan joutua tarkentamaan, jotta eri skenaariot saataisiin kuvattua riittävällä tarkkuudella. Näiden ehtojen pohjalta voimme johtaa web-palvelun käyttämiseksi tarvittavat konkreettiset toimenpiteet:

1. *Web-palvelun löytäminen*. Asiakkaan tulee löytää ja tulla tietoiseksi palvelun tarjoajasta ja sen tuottamasta web-palvelusta.
2. *Web-palvelun toimintamekanismin tiedostaminen*. Asiakkaan ja palvelun tarjoajan tulee jakaa yhteinen tieto web-palvelun rajapinnasta, tiedonsiirtoprotokollasta sekä tieto- ja viestityypeistä.
3. *Web-palvelun ja web-palvelun asiakkaan toteutus*. Web-palvelun rajapinnalle täytyy löytyä toteutus, jota web-palvelun asiakkaan toteutukset voivat kutsua.
4. *Web-palvelun asiakkaan tekemät web-palvelukutsut*. Edellisten vaiheiden jälkeen web-palvelun asiakas voi tehdä kutsuja web-palveluun ja hyödyntää web-palvelun tarjoamaa toiminnallisuutta.

Web-palvelun löytäminen voi tapahtua joko manuaalisesti tai koneellisesti. Manuaalinen etsintä tapahtuu aina ihmisen toimesta. Asiakas voi käyttää esimerkiksi hakupalvelua, jolla voidaan paikallistaa ja valita web-palveluja, jotka toteuttavat halutun toiminnallisuuden. Kun web-palveluja etsitään automatisoidusti, etsintä suoritetaan koneellisesti. W3C esittää kolme johtavaa näkemystä kuinka *web-palvelujen hakupalvelu* (discovery service) tulisi muodostaa. Nämä ovat rekisteri, indeksi ja *vertaisverkko* (peer-to-peer). Rekisteri-menetelmässä web-palvelujen tiedot tallennetaan rekisteriin, missä rekisteri-palvelun tarjoaja vastaa rekisterin ylläpidosta. *UDDI* (Universal Description, Discovery and Integration) voidaan nähdä yhtenä rekisteripohjaisena menetelmänä. Indeksi toimii kokoavana viitteenä tai ohjeena informaation löytämiseksi, joka sijaitsee jossain muualla. Indeksiä ei hallita keskitetysti, vaan indeksejä voi olla useita, joihin indeksien omistajat voivat kerätä haluamiensa web-palvelujen tietoja. Google-hakukonetta voidaan pitää yhtenä esimerkkinä indeksi-menetelmän käytöstä. Vertaisverkkomenetelmässä web-palvelut voidaan nähdä keskenään tasavertaisina tietoverkon solmuina muiden tietoverkon olioiden kanssa. Tällöin web-palvelun asiakas etsii web-palveluja naapurisolmuistaan, jolloin se joutuu myös päättelemään ovako kyseiset solmut web-palveluja ja tarjoavatko ne haluttua toiminnallisuutta.

Web-palvelun toimintamekanismin tiedostamisessa palvelun tarjoajan ja palvelun asiakkaan on sovittava keskenään sekä viestivälityksen yhteisestä semantiikasta että käytetyistä menetelmistä, jotta web-palvelujen viestinvälitys olisi mahdollista. Viestivälityksen menetelmät dokumentoidaan web-palvelun kuvausdokumenteissa, joissa kuvataan sekä web-palvelun rajapinta että semantiikat. *WSD-kuvaus* (Web Service Description) on koneellisesti käsiteltävissä oleva spesifikaatio web-palvelun rajapinnasta, joka on kirjoitettu *WSDL* (Web Service Description Language) -kielellä. Täten WSD-kuvauksen sisältävää dokumenttia kutsutaankin *WSDL-dokumentiksi*. WSD-kuvauksessa määritellään viestityypit, tietotyypit, tiedonsiirtoprotokollat ja tiedonsiirron sarjoitusmuodot, joita tulisi käyttää web-palvelun ja palvelun asiakkaan välisessä viestivälityksessä. WSD-kuvauksessa määritetään myös yksi tai useampi verkko-osoite, missä web-palvelua voidaan kutsua ja mistä voi saada informaatiota palvelun käyttämästä *viestinvälitysmallista* (message exhange pattern). WSD-kuvaus edustaa siis palvelun kanssa vuorovaikutuksessa olevia menetelmiä. Palvelun semantiikat edustavat puolestaan tämän vuorovaikutuksen tarkoitusta ja palvelun odotettua käyttäytymismallia. Semantiikat ilmaisevat siten web-palvelun ja web-palvelun asiakkaan välistä sopimusta, jossa sovitaan yhteiset pelisäännöt osapuolten välillä.

Web-palveluun liittyvä rajapinta ja semantiikat eivät vielä itsessään tarjoa konkreettista toiminnallisuutta. Voimme tulkita nämä toteutuksen ohjeeksi, jota on noudatettava toiminnallisuuden toteutuksessa. W3C:n mukaan web-palvelun määritykset voidaankin yhdistää mihin tahansa ohjelmointikieleen, alustaan, oliomalliin tai viestitysjärjestelmään. Web-palveluja voidaan kutsua suoraan sovelluksista tai selaimen välityksellä. Käytetyllä toteutusmenetelmällä ei ole merkitystä niin kauan kun sekä lähettäjä että vastaanottaja toteuttavat WSDL-tiedostossa kuvatun palvelun rajapinnan sisältämät vaatimukset. Web-palveluja ja niitä käyttäviä asiakassovelluksia voidaankin toteuttaa useilla ohjelmointikielillä ja alustoilla. Tarkoitukseen sopivia ohjelmointikieliä ovat muun muassa *BPEL* (Business Execution Programming Language), Visual C#.NET ja Java.

2.2 WSDL

WSDL on XML-pohjainen kuvauskieli, jota käytetään web-palvelun kuvauksessa. WSDL määrittää XML-kieliopin, jossa tietoverkkopalvelut kuvataan viestien vaihtoon kykenevinä *päätepisteiden* (endpoints) kokoelmina. WSDL-dokumentti on puolestaan WSDL-kielellä kuvattu XML-muotoinen dokumentti, joka koostuu joukosta WSDL-kielen mukaisia määritelmiä. Tämä osuus käsittelee W3C:n WSDL 1.1 -suositusta (W3C, 2001).

WSDL-dokumentti muodostuu siten, että abstraktit viestit ja abstraktit päätepisteiden määritykset erotetaan konkreettisista vastineistaan. Tällä mahdollistetaan abstraktien *viestien* (message) ja *porttityyppien* (porttype) uudelleenkäyttö. Käytännössä tämä tapahtuu siten, että WSDL-dokumetissa olevat toiminnot ja viestit kuvataan ensin abstraktilla tasolla, jonka jälkeen ne sidotaan konkreettisiin tietoverkkoprotokolliin ja viestimuotoihin. Tietoverkkoprotokollat ja viestimuodot määrittävät näin ollen web-palvelun *konkreettiset päätepisteet*, jotka yhdistyvät *absrakteihin päätepisteisiin* — palveluihin.

WSDL-dokumentti sisältää web-palvelun viestityypit, tietotyypit, tiedonsiirtoprotokollat ja tiedonsiirron sarjoitusmuodot. Näiden lisäksi WSDL-dokumentti sisältää myös web-palvelun paikallistamiseen tarvittavat verkko-osoitteet, joihin web-palvelukutsut voidaan kohdistaa. WSDL-dokumentin sisältämät viestityypit ovat abstrakteja kuvauksia päätepisteiden välillä siirrettävästä tiedosta. Päätepisteitä kuvaavat porttityypit ovat

puolestaan abstrakteja toimintokokoelmia. Porttityypin konkreettinen protokolla ja viestimuotomääritykset muodostavat uudelleenkäytettävän *sidonnan* (binding), joka voidaan yhdistää verkko-osoitteeseen. Tällöin saadaan määritettyä *portti*. Kun portteja yhdistetään kokoelmaksi, saadaan määritettyä *palvelu*. Täten voidaankin sanoa, että palvelu koostuu kokoelmasta portteja, jotka puolestaan muodostuvat porttityyppien protokollista, tietomuotomäärityksistä ja verkko-osoitteesta. WSDL-dokumentti käyttää palvelun määrittämiseksi seuraavia elementtejä:

- *types*-elementti kokoaa web-palvelun käyttämät tietotyyppimääritykset käyttäen XML-skeemasyntaksia.
- *message*–elementti määrittää web-palvelun välittämän abstraktin viestin, joka voi koostua yhdestä tai useammasta osasta.
- *operation*–elementti sisältää abstraktin kuvauksen yksittäisestä palvelun toiminnosta
- *portType*-elementti kokoaa abstraktin toimintojoukon, joka liittyy yhteen tai useampaan päätepisteeseen.
- *binding*–elementti määrittää viestimuodon ja protokollan yksityiskohdat kullekin porttityypille.
- *port*–elementti määrittää yksittäisen päätepisteen, jossa sidonta yhdistetään verkko-osoitteeseen.
- *service*–elementti kokoaa toisiinsa liittyvät päätepisteet palveluksi.

WSDL-dokumentti noudattaa hierarkkista rakennetta, missä dokumentin juurena toimii *definitions*-elementti (kuva 1). Definitions-elementille voidaan asettaa myös vapaavalintaiset name- ja targetNamespace-attribuutit. Nimiavaruus saadaan yhdistettyä WSDL-dokumenttiin *import*-elementillä, jolle voidaan asettaa namespace- ja location-attribuutit. Definitions-elementti voi sisältää kuusi päätason elementtiä, joiden avulla palvelu voidaan määrittää. Nämä ovat types-, message-, portType-, binding-, port- ja service-elementit. Näiden lisäksi WSDL tarjoaa vapaavalintaisen *document*-elementin, johon voidaan liittää ihmisen luettavaksi tarkoitettu dokumentaatio. Toisin kuin muut elementit, document-elementti voidaan sijoittaa minkä tahansa muun elementin sisään.

```
<definitions name="nmtoken"?
             targetNamespace="uri"?>
  <import namespace="uri" location="uri"/>*
  ...
</definitions>
```

Kuva 1: Definitions- ja import-elementtien XML-määritys, missä *nmtoken* (name token) -tyyppi (W3C, 2006) on nimimerkkijono, joka koostuu joukosta nimimerkkejä, joita voivat olla kirjaimet, numerot ja tietyt erikoismerkit.

Types-elementti (kuva 2) kokoaa tietotyyppimääritykset, joita tarvitaan viestien vaihdossa. WSDL-dokumentti voi sisältää nolla tai yksi kappaletta types-elementtejä, joka puolestaan voi sisältää 0–n kappaletta xsd:schema-elementtejä, joissa määritetään varsinaiset tietotyypit. XSD-skeemat voivat sisältää suorat XSD-tyyppimääritykset tai vaihtoehtoisesti tyyppimääritykset voidaan tuoda xsd:import-elementillä ulkoisesta XSD-tiedostosta. XSD mahdollistaa sekä yksinkertaisten että monimuotoisten yhdistelmätietotyyppien määrityksen. Yksinkertaisia tietotyyppejä voidaan määrittää simpleType-elementillä ja monimuotoisia tyyppejä vastaavasti complexType-elementillä, missä kaikki simpleType-elementit perustuvat XSD:n sisäänrakennettuihin tietotyyppeihin. Monimuotoisilla tietotyypeillä voidaan muodostaa laajempia koostettuja tietuetyyppejä, jotka voivat sisältää useita attribuutteja. Monimuotoiset tietotyypit mahdollistavat myös tyhjien tietueiden välittämisen.

```
<definitions .... >
  <types>
    <xsd:schema .... />*
  </types>
  ...
</definitions>
```

Kuva 2: Types-elementin XML-määritys.

Message-elementillä (kuva 3) määritetään web-palvelun välittämät viestit. Yksittäinen viesti koostuu yhdestä tai useammasta loogisesta *viestiosasta* (part), missä viestiosa

esitetään WSDL:n *part*-elementillä. Viestiosaan liittyy aina jokin tyyppi. Viestiosan tyyppiliitos tehdään *viestityypitys* (message-typing) -attribuuteilla, jotka riippuvat käytetystä *tyyppijärjestelmästä* (type system). WSDL määrittää useita viestityypitysattribuutteja, joista esimerkkeinä voidaan mainita XSD:n kanssa käytettävät element- ja type-attribuutit. Element-attribuutti viittaa XSD:n elementtiin ja type-attribuutti puolestaan XSD:n tietotyyppiin. Näiden lisäksi voidaan määrittää myös muita viestityypitysattribuutteja, kunhan attribuuttien nimiavaruus eroaa WSDL:n omasta nimiavaruudesta. Viestit ja viestiosat yksilöidään elementtien name-attribuuteissa kuvatuilla nimillä, missä viestin nimi toimii yksilöivänä tunnisteena WSDL-dokumentin sisällä ja viestiosan nimi vastaavasti viestin sisällä.

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

Kuva 3: Message-ja part-elementtien XML-määritys, missä *qname* (qualified name) -tyyppi (W3C, 2004b) muodostuu {URI, paikallinen nimi} -parista.

PortType-elementillä (kuva 4) koostetaan toisiinsa liittyvät abstraktit operaatiot ja viestit. Yksittäinen porttityyppi koostuu yhdestä tai useammasta operation-elementistä, joilla kuvataan palvelun toiminnot. Operation-elementti sisältää *lähetysprimitiivejä* (transmission primitives), joilla määritetään toiminnon viestien lähetys ja vastaanotto. Porttityypit ja toiminnot yksilöidään elementtien name-attribuuteilla, missä porttityypin nimi toimii yksilöivänä tunnisteena WSDL-dokumentin sisällä ja toiminnon nimi vastaavasti porttityypin sisällä. WSDL sisältää neljä erilaista lähetysprimitiiviä, joita päätepisteet voivat käyttää:

- *Yksisuuntaisella* (one-way) -primitiivillä ilmaistaan, että päätepiste voi vastaanottaa viestin, muttei lähetä paluuviestiä.

- *Pyyntö-vastaus* (request-response) -primitiivillä ilmaistaan, että päätepiste vastaanottaa viestin ja lähettää sen jälkeen siihen liittyvän vastausviestin.
- *Lähetys-vastaus* (solicit-response) -primitiivillä ilmaistaan, että päätepiste lähettää viestin ja vastaanottaa sen jälkeen siihen liittyvän vastauksen.
- *Ilmoitus* (notification) -primitiivillä ilmaistaan, että päätepiste lähettää viestin, muttei vastaanota viestiä.

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

Kuva 4: PortType- ja operation-elementtien XML-määritys.

WSDL määrittää oletusarvoisesti sidonnat yksisuuntaiselle- ja pyyntö-vastaus-primitiiville. Lähetys-vastaus- ja ilmoitus-primitiivejä ei tueta suoraan WSDL:n toimesta. Näitä primitiivejä tukevien protokollamääritysten tulee sisältää myös tarvittavat WSDL-sidontalaajennusosat, jotka mahdollistavat kyseisten primiriivien käytön.

Yksisuuntainen primitiivi määritetään yhdellä toiminnon sisään tulevalla *input*-elementillä. Pyyntö-vastaus-primitiivi määritetään yhdellä input- ja *output*-elementillä, missä syöteosa tulee ennen tulosteosaa. Lisäksi virheviesteille voidaan määrittää vapaavalintainen *fault*-elementti. Lähetys-vastaus-primitiivi määritetään muuttamalla syöte- ja tulosteosan järjestystä siten, että output-elementti tulee ennen input-elementtiä. Lähetys-vastaus-primitiivin yhteydessä voidaan myös käyttää fault-elementtiä. Ilmoitus-primitiivi määritetään puolestaan yhdellä output-elementillä. Primitiivien määritys on nähtävissä kuvassa 5.

Input-, output- ja fault-elementit voidaan yksilöidä name-attribuutilla, missä elementin nimi toimii yksilöivänä tunnisteena porttityypin sisällä. Mikäli input- tai output-elementeille ei annetta nimeä, nimet generoidaan automaattisesti primitiiviä hallinnoi-

van toiminnon nimen perusteella. Tästä poiketen fault-elementille on annettava aina yksilöivä tunniste, jotta abstrakti virheviesti voidaan sitoa konkreettiseen muotoon.

```
Yksisuuntainen primitiivi:
...
<operation name="nmtoken">
    <input name="nmtoken"? message="qname"/>
</operation>
...
Pyyntö-vastaus-primitiivi:
...
<operation name="nmtoken" parameterOrder="nmtokens">
    <input name="nmtoken"? message="qname"/>
    <output name="nmtoken"? message="qname"/>
    <fault name="nmtoken" message="qname"/>*
</operation>
...
Lähetys-vastaus-primitiivi:
...
<operation name="nmtoken" parameterOrder="nmtokens">
    <output name="nmtoken"? message="qname"/>
    <input name="nmtoken"? message="qname"/>
    <fault name="nmtoken" message="qname"/>*
</operation>
...
Ilmoitus-primitiivi:
...
<operation name="nmtoken">
    <output name="nmtoken"? message="qname"/>
</operation>
...
```

Kuva 5: Primitiivien XML-määritys toimintojen yhteydessä.

Binding-elementillä eli sidonnalla (kuva 6) määritetään kullekin porttityypille viestin

muoto ja protokollan yksityiskohdat. Jokaista porttityyppiä kohden voi olla vapaavalintainen määrä sidontoja. Binding-elementti sisältää name- ja type-attribuutit, missä sidonnan nimi toimii sidonnan yksilöivänä tunnisteena WSDL-dokumentin sisällä. Sidonnan tyypillä viitataan puolestaan sidonnan kohteena olevaan porttityyppiin. Sidonnan sisältämän toiminnon nimi sitoo kunkin toiminnon binding-elementissä sidotun porttityypin saman nimiseen toimintoon. Kuormitetuilla saman nimisillä toiminnoilla täytyy lisäksi määrittää toiminnon sisältämien input- ja output-elementtien nimet, jotta konkreettinen sidostoiminto saadaan sidottua yksiselitteisesti abstraktin porttityypin toiminnon kanssa. Lisäksi sidontaan liittyy myös seuraavat rajoitteet:

- Kunkin sidonnan täytyy määrittää täsmälleen yksi protokolla.
- Sidonta ei saa määrittää osoitetietoja.

Sidonnan laajennuselementeillä määritetään puolestaan syöte-, tuloste- ja virheviestien konkreettinen kielioppi. Laajennuselementtejä voidaan käyttää myös toiminto- ja sidontakohtaisesti. WSDL 1.1 sisältää sisäänrakennetut sidonnan laajennuselementit SOAP 1.1-, HTTP GET/POST- ja MIME-protokollille.

Port-elementillä (kuva 7) määritetään yksittäinen päätepiste, joka liitetään sidontaan. Port-elementti sisältää name- ja binding-attribuutit, missä portin nimi toimii yksilöivänä tunnisteena WSDL-dokumentin sisällä. Portin binding-attribuutilla viitataan porttiin kytkettävään sidontaan. Portin osoitetiedot määritetään sidonnan laajennuselementeillä. Porttiin liittyvät myös seuraavat rajoitteet:

- Portti saa määrittää maksimissaan yhden osoitteen.
- Portti ei saa määrittää osoitteen lisäksi muuta sidontatietoa.

Service-elementti (kuva 8) kokoaa toisiinsa liittyvät portit palveluksi, missä palvelu voi sisältää 0–n kappaletta portteja. Service-elementti sisältää name-attribuutin, jonka sisältämä nimi toimii yksilöivänä tunnisteena WSDL-dokumentin sisältämien palvelujen joukossa.

```
<definitions .... >
  ...
  <binding name="nmtoken" type="qname"> *
    <-- laajennuselementti (sidonta) --> *
    <operation name="nmtoken"> *
      <-- laajennuselementti (toiminto) --> *
      <input name="nmtoken"? > ?
          <-- laajennuselementti (syöte) -->
      </input>
      <output name="nmtoken"? > ?
          <-- laajennuselementti (tuloste) --> *
      </output>
      <fault name="nmtoken"> *
          <-- laajennuselementti (virhe) --> *
      </fault>
    </operation>
  </binding>
  ...
</definitions>
```

Kuva 6: Binding-elementin XML-määritys.

2.3 SOAP

Tässä kohdassa esitellään W3C:n SOAP 1.1 -protokolla, jota käytetään WSDL 1.1 -kielen tarjoamassa SOAP-sidonnassa. Tässä kohdassa kuvatut tiedot perustuvat W3C:n SOAP 1.1 -muistioon (W3C, 2000).

SOAP tarjoaa yksinkertaisen ja kevyen mekanismin rakenteisen XML-tiedon välitykseen hajautetussa ympäristössä. SOAP ei itsessään määritä sovelluksen semantiikkaa tai toteutusta. Sen sijaan SOAP määrittää yksinkertaisen mekanismin sovelluksen semantiikan ilmaisemiseksi. SOAP-protokolla koostuu kolmesta osasta:

- SOAP *kirjekuori* (envelope) -rakenne määrittää SOAP-viestin rungon, jolla kuvataan viestin sisältö, osapuolet ja pakollisuussäännöt.

```
<definitions .... >
  ...
  <service .... > *
    <port name="nmtoken" binding="qname"> *
      <-- Sidonnan laajennuselementti -->
    </port>
  </service>
  ...
</definitions>
```

Kuva 7: Port-elementin XML-määritys.

```
<definitions .... >
  ...
  <service name="nmtoken"> *
    <port .... />*
  </service>
</definitions>
```

Kuva 8: Service-elementin XML-määritys.

- SOAP *koodaussäännöt* (encoding rules) määrittää *sarjoitusmekanismin* (serialization mechanism), jota käytetään sovelluksen määrittämien tietotyyppien välitykseen.
- SOAP *RPC* (Remote Procedure Call) -esitys määrittää etäproseduurikutsujen ja -vastauksien esitystavan.

SOAP-toteutuksia voidaan optimoida eri tietoverkkojärjestelmille, mistä voidaan mainita esimerkkinä HTTP-sidonta, jonka avulla *SOAP-viestit* voidaan lähettää HTTP-protokollan välityksellä. SOAP-viestit reititetään niin kutsutussa *viestipolussa* (message path), riippumatta käytetystä protokollasta. Tämä mahdollistaa viestin prosessoinin yhdessä tai useammassa välisolmussa ennen lopullista kohdetta. Välisolmuja kutsutaan *SOAP-välittäjiksi*. SOAP-välittäjällä tarkoitetaan sovellusta, joka kykenee sekä

vastaanottamaan SOAP-viestejä että välittämään niitä eteenpäin. Täten SOAP-viestin viestipolku voi kulkea vastaanottajalle SOAP-välittäjien kautta, jolloin vastaanottaja ja välittäjä yksilöidään omilla *URI* (Uniform Resource Identifier) -osoitteillaan.

SOAP-viesti on XML-muotoinen dokumentti, joka koostuu pakollisesta SOAP-kirjekuoresta, vapaavalintaisesta SOAP-*otsakkeesta* (header) ja pakollisesta SOAP-*runko-osasta* (body). SOAP-kirjekuori toimii SOAP-dokumentin juurena. SOAP-otsake toimii vapaavalintaisena lisäominaisuuksien hallintamekanismina, jolla SOAP-viestiin voidaan lisätä uusia ominaisuuksia. Tällaisia ominaisuuksia ovat muun muassa erilaiset transaktion hallintamekanismit ja tunnistusmenetelmät. SOAP-otsake sisältää myös tiedot, joilla voidaan määrittää ne tahot, joiden tulisi käsitellä otsakkeessa hallitut ominaisuudet. Lisäksi otsakkeessa voidaan määrittää, milloin ominaisuus on pakollinen tai vapaavalintainen. SOAP-runko-osa puolestaan kapseloi lopulliselle vastaanottajalle tarkoitetun pakollisen tiedon. SOAP-dokumentti sisältää seuraavat dokumentin osia vastaavat peruselementit:

- *Envelope*-elementti muodostaa SOAP-kirjekuorirakenteen.
- *Header*-elementti muodostaa SOAP-otsakkeen.
- *Body*-elementti muodostaa SOAP-runko-osan.

SOAP-sovelluksen tulee sisällyttää SOAP-määritysten mukaiset nimiavaruudet tuottamilleen SOAP-viestin elementeille ja attribuuteille. Täten SOAP-sovelluksen tulee myös pystyä käsittelemään vastaanottamiensa viestien SOAP-nimiavaruudet. SOAP 1.1 -viestin tulee sisältää Envelope-elementti, jossa on määritetty ”http://schemas.xmlsoap.org/soap/envelope/”-nimiavaruus. Mikäli vastaanottavan SOAP-sovelluksen kirjekuori on liitetty toiseen nimiavaruuteen, tulee SOAP-sovelluksen jättää eriävän nimiavaruuden omaava viesti käsittelemättä. Mikäli SOAP-sovellus jättää viestin käsittelemättä pyyntö-vastaus -tyyppisellä protokollalla, sovelluksen täytyy lähettää ”http://schemas.xmlsoap.org/soap/envelope/”-nimiavaruutta käyttävä paluuviesti, joka sisältää VersionMismatch-virhekoodin.

SOAP-viesti sisältää myös yleisen encodingStyle-attribuutin, jota voidaan käyttää kaikissa elementeissä. Attribuuttiin asetetaan SOAP-*koodaustyyli*, jolla ilmaistaan SOAP-viestissä käytetyt *sarjoitussäännöt* (serialization rules). Attribuutin vaikutusalue ulottuu siihen elementtiin, missä se on määritelty. Vaikutusalue ulottuu myös oletusarvoisesti kaikkiin lapsielementteihin, ellei niissä ole määritelty eriäviä sarjoitussääntöjä.

EncodingStyle-attribuutti mahdollistaa yhden tai useamman sarjoitussäännön määrityksen, missä attribuutin arvo muodostuu järjestetystä listasta, joka koostuu puolestaan URI-osoitteista (kuva 9). Järjestys muodostetaan siten, että eniten merkitsevä sarjoitussääntö tulee ensimmäiseksi ja vähiten merkitsevä viimeiseksi. SOAP:ssa määritetyt sarjoitussäännöt ilmaistaan ”http://schemas.xmlsoap.org/soap/encoding/” -URI-osoitteella. Tyhjällä URI-osoitteella ilmaistaan se, ettei sarjoitussääntöjä tarvitse käyttää.

```
<ns:Envelope
   xmlns:ns="uri"
   encodingStyle="http://schemas.xmlsoap.org/
                  soap/encoding/" ...>
  ...
</ns:Envelope>
```

Kuva 9: Esimerkki sarjoitussääntöjen määrityksestä encodingStyle-attribuutilla.

Envelope-elementillä (kuva 10) muodostetaan SOAP-kirjekuorirakenne. Envelope toimii SOAP-dokumentin juurielementtinä ja se voi sisältää sekä Header- että Body-elementin tai pelkän Body-elementin. Näiden lisäksi Envelope-elementti voi sisältää myös muita vapaavalintaisia elementtejä, jotka on esiteltävä Body-elementin jälkeen. Envelope-elementti sisältää xmlns-nimiavaruusattribuutin, jonka lisäksi elementti voi sisältää myös muita vapaavalintaisia nimiavaruudella merkittyjä attribuutteja.

```
<ns:Envelope xmlns:ns="uri"? ...>
  <ns:Header.../>?
  <ns:Body.../>
  ...
</ns:Envelope>
```

Kuva 10: SOAP-Envelope -elementin XML-määritys, missä ”ns”-osa muodostaa SOAP-kirjekuoren nimiavaruuden.

Header-elementillä (kuva 11) muodostetaan vapaavalintainen SOAP-otsake. Mikäli

Header-elementti on asetettu, sen täytyy sijaita Envelope-elementin ensimmäisenä lapsielementtinä. Header-elementti voi sisältää myös vapaavalintaisia lapsielementtejä, joiden tulee sisältää oma nimiavaruutensa. SOAP-otsakkeen lapsielementtejä kutsutaan *otsakemerkinnöiksi* (header entries). SOAP-otsakeattribuuteilla määritetään miten SOAP-viestin vastaanottajan tulisi prosessoida viesti. SOAP-viestin generoivan sovelluksen tulisi asettaa SOAP-otsakkeen attribuutteja ainoastaan Header-elementin suorissa lapsielementeissä. Vastaavasti SOAP-viestin vastaanottajan täytyy jättää käsittelemättä kaikki edellä mainitusta poikkeavat SOAP-otsakeattribuutit. Otsakemerkinnät noudattavat seuraavia koodaussääntöjä:

1. Otsakemerkintä yksilöidään elementin nimellä, joka koostuu nimiavaruus-URI-osoitteesta ja paikallisesta nimestä.
2. SOAP-koodaustyyliä voidaan käyttää osoittamaan otsakemerkintöjen sarjoitussäännöt.
3. SOAP mustUnderstand- ja actor-attribuutteja voidaan käyttää ilmaisemaan miten ja kuka voi prosessoida merkinnän.

Otsakkeita voidaan lähettää myös SOAP-välittäjille, jolloin kaikkia otsakkeita ei ole välttämättä tarkoitettu viestin lopulliselle vastaanottajalle. Tämän seurauksena SOAP-välittäjä ei saa lähettää eteenpäin itselleen lähetettyjä otsakkeita. SOAP-välittäjä voi kuitenkin lisätä viestiin uuden samansisältöisen otsakkeen, joka voidaan lähettää viestin mukana seuraavalle välittäjälle tai vastaanottajalle. Otsakkeen vastaanottaja määritetään globaalilla actor-attribuutilla. Attribuutin arvoksi annetaan vastaanottajan URI-osoite. Erityisellä ”http://schemas.xmlsoap.org/soap/actor/next” -osoitteella voidaan ilmaista, että otsake on tarkoitettu ensimmäiselle SOAP-viestiä prosessoivalle SOAP-sovellukselle. Vastaavasti voidaan ilmaista, että otsake on tarkoitettu lopulliselle vastaanottajalle. Täten lopullista vastaanottajaa voidaan kutsua *oletusvastaanottajaksi*. Tämä ilmaistaan jättämällä actor-attribuutin määritys pois. Actor-attribuutin lisäksi otsake voi sisältää globaalin mustUnderstand-attribuutin, jolla ilmaistaan otsakkeen käsittelyn pakollisuus. Käsittelyn pakollisuus ilmaistaan arvolla ”1” ja vapaavalintaisuus arvolla ”0”. Mikäli mustUnderstand-attribuutti jätetään pois, otsakkken käsittely tulkitaan vapaavalintaiseksi. Pakollisen otsakkeen vastaanottajan on joko käsiteltävä tai hylättävä vastaanottamansa viesti.

Body-elementillä (kuva 12) muodostetaan SOAP-runko-osa. Mikäli Header-elementti

```
<ns:Envelope...>
  <ns:Header>
    <t:Transaction
       xmlns:t="uri"
       ns:mustUnderstand="1">
       ...
    </t:Transaction>
    ...
  </ns:Header>
  ...
</ns:Envelope>
```

Kuva 11: SOAP Header-elementin XML-määritys, joka sisältää Transaction-tunnisteella merkatun lapsielementin.

on asetettu, Body-elementin täytyy sijaita välittömästi tämän jälkeen. Muuten sen täytyy sijaita Envelope-elementin ensimmäisenä lapsielementtinä. Mikäli SOAP-viesti ei sisällä otsaketta, Body-elementti tulkitaan semanttisesti ekvivalentiksi pakollisen otsakkeen kanssa, joka on osoitettu oletusvastaanottajalle. Body-elementin lapsielementtejä kutsutaan *runkomerkinnöiksi* (body entries). Jokainen runkomerkintä koodataan itsenäiseksi elementiksi Body-elementin sisällä. SOAP määrittää yhden runkomerkinnän, jota käytetään virheiden raportointiin. Tämä on *Fault*-elementti, jota kutsutaan jatkossa *SOAP-virheeksi*. Runkomerkinnät koodataan seuraavien sääntöjen mukaisesti:

1. Runkomerkintä yksilöidään elementin nimellä, joka koostuu nimiavaruus-URI-osoitteesta ja paikallisesta nimestä.
2. SOAP-koodaustyyliä voidaan käyttää osoittamaan runkomerkintöjen sarjoitussäännöt.

Fault-elementtiä (kuva 13) käytetään SOAP-viestissä virheiden ja statustiedon välitykseen. Mikäli Fault-elementti on asetettu, sen tulee sijaita Body-elementin sisällä. Body-elementti voi sisältää yhden Fault-elementin. Fault-elementti kapseloi virhekoo-

```
<ns:Envelope...>
  ...
  <ns:Body>
    ...
  </ns:Body>
  ...
</ns:Envelope>
```

Kuva 12: SOAP Body-elementin XML-määritys.

din, virhetekstin, virheen aiheuttajan ja virhekuvauksen, joita vastaavat seuraavat lapsielementit:

- *faultcode*-elementillä esitetään virheen yksilöivä tieto, josta sovellus pystyy tulkitsemaan virheen syyn koneellisesti. Elementti on pakollinen.
- *faultstring*-elementillä esitetään virheen tekstimuotoinen kuvaus. Elementti on pakollinen.
- *faultactor*-elementillä esitetään tietoa virheen aiheuttajasta, jonka avulla viestipolusta voidaan jäljittää virheen generoinut solmu. Tämän arvo vastaa actor-attribuutin URI-osoitetta sillä erotuksella, että viestin vastaanottajan sijasta ilmaistaan viestin lähettäjä. Kaikkien SOAP-välittäjien on sisällytettävä tämä elementti.
- *detail*-elementillä ilmaistaan tarkemmat SOAP-viestin runko-osaan liittyvät virhetiedot. Elementti on pakollinen, mikäli Body-elementin sisältöä ei pystytä käsittelemään. Elementtiä ei saa käyttää otsakkeisiin liittyvissä virhetilanteissa. Näitä varten tulee käyttää otsakkeita. Mikäli detail-elementtiä ei välitetä, voidaan olettaa, että virhe ei liittynyt Body-elementin käsittelyyn. Detail-elementti voi sisältää *tarkennusmerkintöjä* (detail entries), jotka noudattavat seuraavia koodaussääntöjä:
  1. Tarkennusmerkintä yksilöidään elementin nimellä, joka koostuu nimiavaruus-URI-osoitteesta ja paikallisesta nimestä.
  2. Runkomerkintöjen sarjoitussäännöt osoitetaan SOAP-koodaustyyylillä.

```
<ns:Envelope...>
  ...
  <ns:Body>
    <ns:Fault>
      <faultcode>...</faultcode>
      <faultstring>...</faultstring>
      <detail>
        <message>...</message>
        <errorcode>...</errorcode>
      </detail>
    </ns:Fault>
  </ns:Body>
  ...
</ns:Envelope>
```

Kuva 13: SOAP Fault-elementin XML-määritys.

Edellä mainittujen elementtien lisäksi Fault-elementti voi sisältää myös muita validin nimiavaruuden sisältäviä lapsielementtejä. SOAP sisältää seuraavat oletusarvoiset virhekoodit, joita voidaan käyttää ”http://schemas.xmlsoap.org/soap/envelope/” -nimiavaruuden kanssa:

- *VersionMismatch*-koodi ilmaisee, että viestin käsittelijä on löytänyt Envelope-elementistä virheellisen nimiavaruuden.
- *MustUnderstand*-koodi ilmaisee, että viestin käsittelijä ei ole pystynyt käsittelemään pakollista Header-elementtiä.
- *Client*-virheluokan koodi ilmaisee, että viesti on väärin muodostettu tai se ei sisältänyt tarvittavaa informaatiota. Esimerkiksi maksutapahtumaviestistä saattaa puuttua tarpeellisia tunnnistustietoja, joita tarvitaan maksutapahtuman varmennuksessa. Koodi ilmaisee yleisellä tasolla sen, että viesti tulisi lähettää uudestaan puuttuvilla tiedoilla täydennettyinä.
- *Server*-virheluokan koodi ilmaisee, että viestiä ei voitu käsitellä viestistä riippumattomista syistä, jolloin on mahdollista, että viestin käsittely onnistuu myö-

hempänä ajankohtana.

2.4 SOAP-sidonta

SOAP-sidonta mahdollistaa SOAP-protokollan liittämisen osaksi web-palvelun WSDL-määritystä, jolloin web-palvelu voi lähettää ja vastaanottaa SOAP-protokollan mukaisia SOAP-viestejä. Tässä yhteydessä esitellään SOAP 1.1 -protokollan sitominen WSDL 1.1 -kielen sisältämien laajennuselementtien avulla. WSDL 1.1 -suosituksessa (W3C, 2001) on kuvattu seuraavat SOAP-laajennuselementit:

- *soap:binding*-elementillä osoitetaan, että sidonta kohdistuu SOAP-protokollamuotoon.
- *soap:operation*-elementillä tarjotaan kokonaisvaltaista informaatiota toimintoa varten.
- *soap:body*-elementillä määritetään, miten viestiosat liitetään SOAP Body-elementin sisälle.
- *soap:fault*-elementillä määritetään SOAP Fault-elementin sisältämän details-elementin tiedot.
- *soap:header* ja *soap:headerfault* -elementit mahdollistavat otsakkeen määrityksen, jota käytetään SOAP-kirjekuoressa Header-elementin sisällä.
- *soap:address*-elementillä asetetaan portin URI-osoite.

Soap:binding-elementillä (kuva 14) kohdistetaan sidonta SOAP-protokollan muotoon. Elementti on pakollinen, kun käytetään SOAP-sidontaa. Soap:binding-elementti sisältää transport- ja style-attribuutit. Vaaditun transport-attribuutin arvoksi tulee URI-osoite, jolla ilmaistaan käytetty siirtoprotokolla. Esimerkiksi ”http://schemas.xmlsoap.org/soap/http”-osoitteella ilmaistaan, että käytetään SOAP-määrityksen HTTP-sidontaa. Vapaavalintaisella style-attribuutilla asetetaan tyyli, missä tyylin arvona voi olla joko ”RPC”- tai ”document”-tyyli. Tyyliasetus toimii oletuksena kaikille elementin sisältämille soap:operation-elementeille. Mikäli tyyli jätetään asettamatta, käytetään oletusarvoisesti ”document”-tyyliä.

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

Kuva 14: Soap:binding-elementin XML-määritys.

Soap:operation-elementti (kuva 15) tarjoaa kokonaisvaltaista tietoa toimintoa varten. Elementti sisältää soapAction- ja style-attribuutit. SoapAction-attribuutti määrittää toiminnolle SOAPAction-otsakkeen URI-osoitteen. Attribuutti on pakollinen HTTP-sidonnassa, mutta muissa SOAP-sidoinnoissa arvoa ei saa asettaa. Style-attribuutilla ilmaistaan käytetäänkö RPC- tai document-tyyliä. Mikäli arvoa ei aseteta, käytetään SOAP-sidonnan tyyliä.

```
<definitions .... >
  <binding .... >
    <operation .... >
      <soap:operation soapAction="uri"?
                      style="rpc|document"?>?
    </operation>
  </binding>
</definitions>
```

Kuva 15: Soap:operation-elementin XML-määritys.

Soap:body-elementillä (kuva 16) määritetään, miten eri viestiosat liitetään SOAP-runko-osan sisälle. Viestiosat voivat olla joko abstrakteja tyyppimäärityksiä tai konkreettisia skeemamäärityksiä. Mikäli viestiosat ovat abstrakteja tyyppimäärityksiä, tyypit sarjoitetaan koodaustyylissä määritetyn sääntöjoukon mukaisesti. Mikäli käytetään konkreettisia viestiosia, alkuperäinen koodaustyyli voidaan ilmaista vihjeenä. Soap:body-elementtiä käytetään sekä RPC- että document-tyyppisissä viesteissä. Käytetty tyyli vaikuttaa siihen, miten SOAP-runko-osa muodostetaan:

- RPC-tyylillä jokainen viestiosa toimii parametrina tai paluuarvona, joka kapseloidaan rungon sisällä *kääre-elementtiin* (wrapper element). Kääre-elementti nimetään identtisesti toiminnon nimen kanssa ja sen nimiavaruudeksi tulee namespace-attribuutin osoittama nimiavaruus. Viestiosat nimetään parametrien nimisiksi ja ne järjestetään kutsujärjestyksen mukaisesti.
- Document-tyylillä viestiosat tulevat suoraan Body-elementin alaisuuteen ilman erillisiä kääre-elementtejä.

```
<definitions .... >
  <binding .... >
    <operation .... >
      <input>
        <soap:body parts="nmtokens"?
                   use="literal|encoded"?
                   encodingStyle="uri-list"?
                   namespace="uri"?>
      </input>
      <output>
        <soap:body ... >
      </output>
    </operation>
  </binding>
</definitions>
```

Kuva 16: Soap:body-elementin XML-määritys.

Soap:body-elementti sisältää parts-, use-, encodingStyle- ja namespace-attribuutit. Vapaavalintaisella parts-attribuuttilla osoitetaan, mitkä viestiosat sijaitsevat SOAP-viestin runko-osassa. Mikäli parts-attribuuttia ei aseteta, kaikkien viestiosien oletetaan sijaitsevan SOAP-viestin runko-osassa. Pakollisella use-attribuutilla voidaan ilmaista koodataanko viestiosat jollakin koodaussäännöllä vai määrittääkö viestiosat konkereettisen viestiskeeman. Atribuutti voi saada arvokseen joko ”encoded”- tai ”literal”-arvon, missä ensin mainitulla ilmaistaan koodasusääntöjen käyttö ja jälkimmäisellä konkreettisen viestiskeeman käyttö. Mikäli use-attribuutin arvo on ”encoded”, encodingStyle-

ja namespace-attribuuteilla annetaan koodauksen tarvitsemaa tietoa. Mikäli käytetään ”literal”-arvoa, näitä parametreja ei tarvitse asettaa. EncodingStyle-attribuutilla voidaan kuitenkin ilmaista, että konkreettinen muoto on saatu johdettua tietyllä koodauksella, josta esimerkkinä SOAP-koodaus.

Soap:fault-elementti (kuva 17) määrittää SOAP-virheen detail-elementin sisällön. Virheviestin täytyy koostua yhdestä osasta. Elementti sisältää pakolliset name- ja use-attribuutit. Name-attribuutin nimellä viitataan WSDL:ssä toiminnolle määritettyyn fault-elementtiin. Use-, encodingStyle- ja namespace-attribuutteja käytetään samaan tapaan kuin soap:body-elementissä.

```
<definitions .... >
  <binding .... >
    <operation .... >
      <fault>*
        <soap:fault name="nmtoken"
                    use="literal|encoded"
                    encodingStyle="uri-list"?
                    namespace="uri"?>
      </fault>
    </operation>
  </binding>
</definitions>
```

Kuva 17: Soap:fault-elementin XML-määritys.

Soap:header- ja soap:headerfault-elementit (kuva 18) mahdollistavat otsakkeen määrityksen, jota käytetään SOAP-kirjekuoressa Header-elementin sisällä. Elementit sisältävät message-, part-, use-, encodingStyle- ja namespace-attribuutit, missä use-, encodingStyle- ja namespace-attribuuteja käytetään samaan tapaan kuin soap:body-elementissä. Message- ja part-attribuutit viittaavat yhdessä viestiosaan, joka määrittää otsakkeen tyypin. Part-attribuutti viittaa skeemaan, joka voi sisältää soap:actor- ja soap:mustUnderstand-attribuutit, mikäli use-attribuutissa käytetään ”literal”-arvoa. Soap:header-elementti voi sisältää valinnaisia soap:headerfault-elementtejä, jotka mahdollistavat otsakkeiden virhetiedon lähetyksessä käytettyjen otsaketyyppien mää-

rityksen.

```
<definitions .... >
  <binding .... >
    <operation .... >
      <input>
        <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"?/>*
        <soap:header>
      </input>
      <output>
        <soap:header ... >*
          <soap:headerfault ... />*
        <soap:header>
      </output>
    </operation>
  </binding>
</definitions>
```

Kuva 18: Soap:header- ja soap:headerfault-elementtien XML-määritys.

Soap:address-elementtiä (kuva 19) käytetään portin URI-osoitteen määrityksessä. Jokaiselle SOAP-sidontaa käyttävälle portille on määritettävä yksi osoite. Soap:adress-elementti sisältää pakollisen location-attribuutin, joka sisältää SOAP-sidonnassa käytetyn URI-osoitteen. URI-osoitteen on vastattava SOAP-sidonnan määrittämää lähetystapaa.

```
<definitions .... >
    <port .... >
        <binding .... >
           <soap:address location="uri"/>
        </binding>
    </port>
</definitions>
```

Kuva 19: Soap:address-elementin XML-määritys.

2.5 REST

Web-palvelut voivat vastaanottaa ja lähettää SOAP-viestejä, joiden lähetyksessä voidaan käyttää HTTP-GET- ja HTTP-POST-metodeja. Web-palveluja voidaan kutsua myös ilman SOAP-protokollaa, jolloin HTTP-GET- ja HTTP-POST-metodeilla lähetetyn tiedon muoto ei noudata SOAP-rakennetta. Web-palvelukutsuja voidaan täten tehdä myös *HTTP-GET-* ja *HTTP-POST-menetelmillä*, joista HTTP-GET-menetelmää kutsutaan *REST* (Representational State Transfer) -menetelmäksi (Prescord, 2002).

REST-menetelmässä web-palvelukutsu muodostetaan *URL* (Uniform Resource Locator)-osoitteella ja siihen liittyvillä URL-parametreilla. REST-menetelmässä web-palvelukutsu tehdään HTTP-GET-pyynnöllä, joka palauttaa web-palvelun tuottaman XML-dokumentin. Seuraavassa esimerkissä on nähtävissä Google Picasa -web-palveluun kohdistuva REST-menetelmää käyttävä web-palvelukutsu (Kinnunen, 2008):

```
http://picasaweb.google.com/data/feed/api/all?
kind=photo&q=Joensuu
```

Yllä olevassa kutsussa on nähtävissä, että pyyntö koostuu staattisesta palvelukohtaisesta alkuosasta ja kutsua tarkentavista parametreista, joilla voidaan vaikuttaa tulosdokumentin sisältöön. Mikäli web-palvelu tukee REST-menetelmää, web-palvelukutsujen tekeminen on helppoa, koska kutsu voidaan muodostaa ilman erityiskomponentteja. Riittää kun suorittaa HTTP-GET-pyynnön muodostettuun osoitteeseen. Tämä voidaan

suorittaa joko ohjelmallisesti tai manuaalisesti Internet-selaimella, missä jälkimmäinen tarjoaa helpon menetelmän testata web-palveluja.

REST-menetelmässä web-palvelujen kutsuminen voidaan Gosnellin ja Denisen (2005) mukaan tiivistää seuraaviin toimenpiteisiin:

1. Tunnista haluamasi web-palvelu ja sen hyväksymät parametrit
2. Muodosta URL-osoite, joka sisältää palvelun osoitteen ja vaaditut parametrit.
3. Testaa muodostettu URL-osoite Internet-selaimella, jotta voit varmistua osoitteen toimivuudesta.
4. Tee valitsemallasi ohjelmointikielellä HTTP-GET-pyyntö muodostettuun URL-osoitteeseen.
5. Jäsennä pyynnön vastauksena saatu XML-dokumentti halutulla tavalla.

Kuten edellä on nähtävissä, REST-menetelmä tarjoaa helpon ja yksinkertaisen tavan suorittaa web-palvelukutsuja. REST-menetelmä sisältää myös joitakin rajoituksia. REST-menetelmä ei sovellu hyvin arkaluonteisen tiedon välitykseen, koska muodostettu URL-osoite on tekstimuotoista ja helposti luettavaa tietoa. Esimerkiksi henkilötunnuksen tai salasanan välittäminen URL-osoitteen mukana voisi olla hyvin vahingollista. Lisäksi URL-osoitteen koko on rajoitettu, mikä rajoittaa siihen liitettävän parametritiedon määrää. REST-menetelmä ei sovellu myöskään datan lähetykseen, koska se käyttää tiedon hakuun tarkoitettua GET-metodia. REST mahdollistaa kuitenkin rajoitetun tietojen lähetyksen, kunhan pysytään URL-osoitteen maksimikoon rajoissa. Tyagin (2006) mukaan REST soveltuu seuraaviin tilanteisiin:

- Web-palvelut ovat täysin tilattomia.
- Palvelun suoritusta voidaan tehostaa *välimuistilla* (cache).
- Palvelun tarjoajalla ja asiakkaalla on yhteinen käsitys kontekstista ja välitettävästä sisällöstä.
- Käytettävissä olevalla *kaistanleveydellä* (bandwidth) on erityinen merkitys ja sitä joudutaan rajoittamaan.

- REST-tyyppiset web-palvelut voidaan liittää olemassaoleviin WWW-sivuistoihin.

HTTP-GET-menetelmän lisäksi voidaan käyttää myös HTTP-POST-menetelmää, joka on samankaltainen HTTP-GET-menetelmän kanssa. HTTP-POST-menetelmässä lähetetään URL-osoitteen sijasta XML-dokumentti, joka sisältää web-palvelun kutsumisessa tarvitut tiedot. Tämän jälkeen web-palvelu lähettää takaisin HTTP-POST-vastauksen, joka sisältää vastausdokumentin. HTTP-POST soveltuu siten sekä tietojen lähetykseen että vastaanottoon. Tämä mahdollistaa myös REST-menetelmää turvallisemman tavan lähettää arkaluontoisia tietoja.

HTTP-POST-menetelmässä web-palvelujen kutsuminen voidaan Gosnellin ja Denisen (2005) mukaan tiivistää seuraaviin toimenpiteisiin:

1. Tunnista haluamasi web-palvelu ja sen hyväksymät parametrit
2. Muodosta XML-dokumentti, joka sisältää parametrit
3. Tee valitsemallasi ohjelmointikielellä HTTP-POST-kutsu lähettääksesi muodostetun XML-dokumentin halutulle web-palvelulle.
4. Jäsennä vastauksena saatu XML-dokumentti halutulla tavalla.

3 Sovelluksen laajentaminen web-palvelurajapinnoilla

Tässä luvussa esitellään Internet-sivustojen tarjoamien ulkoisten web-palvelurajapintojen käyttöä. Luvun alkupuolella esitellään yleisemmin verkosta löytyviä web-palvelurajapintoja. Näistä esitellään tarkemmin Google-, Amazon- ja eBay-sivustojen web-palvelurajapinnat. Samalla esitellään myös mitä toimenpiteitä näiden rajpintojen hyödyntäminen vaatii. Tämän jälkeen esitellään Google API -palvelurajapintaa ja sen hyödyntämistä omissa web-sovelluksissa. Lopuksi tarkastellaan menetelmää, jolla voidaan toteuttaa web-palveluja, jotka yhdistävät useita web-palveluja saman palvelun alle.

3.1 Web-palvelujen rajapinnat

Internet-verkossa on useita sivustoja, jotka tarjoavat toiminnallisuutensa hyödyntämistä web-palvelurajapintojen kautta. Näitä rajapintoja voidaan kutsua yleisemmin *Web API* (Application Programming Interface) -rajapinnoiksi. Täten web-palvelurajapintoja tarjoavat palvelut sisältävät kukin oman Web API -rajapintansa. Tässä yhteydessä esitellään Googlen, Amazonin ja eBayn Web API -rajapintoja ja niiden sisältämää toiminnallisuutta.

Gosnell ja Denise (2005) esittävät Web API -rajapinnan seuraavasti: Web API -rajapinnat ovat Internetin kautta käytettäviä palvelurajapintoja, joita sovellukset voivat kutsua Internetin välityksellä. Web API voidaan siten mieltää kokoelmaksi web-palveluja, joista kukin palvelu sisältää yhden tai useamman kutsumetodin, joita voidaan kutsua Internetin kautta. Web-palvelut ja Web API -rajapinnat ovat erillisiä käsitteitä, mutta niihin viitataan usein ristiin, koska ne liittyvät toisiinsa hyvin läheisesti.

Useat Internet-palvelujen tarjoajat ovat omaksuneet web-palvelukonseptin ja tarjoavat omia Web API -rajapintojaan, jotka mahdollistavat sivustojen ydintoimintojen hyödyntämisen omissa sovelluksissa. Esimerkkeinä tällaisista palvelun tarjoajista voidaan mainita Google-, Amazon- ja eBay-sivustot.

Monet Web API -rajapinnat vaativat *kehittäjätunnisteen* (developer token), jolla kontrolloidaan web-palvelujen käyttöä. Tunnisteet mahdollistavat myös palvelu- ja transaktio-kohtaiset veloitukset, joita voidaan hyödyntää liiketoiminnassa. Useimmat Web API -rajapinnat sisältävät joko ilmaisen rajoitetun lisenssin tai määräaikaisen ko-

keiluajanjakson. Osa palveluista on käytettävissä maksua vastaan.

3.1.1 Google Data API

Google tarjoaa API-rajapinnat lähes kaikille palveluilleen. Tämä kattaa niin työpöytäsovellukset, mobiililaitteet kuin myös web-palvelutkin. Tässä tutkielmassa esitellään Google API -rajapinnan sisältämää Google Data API -rajapintaa (Google Data API, 2008), joka on suunnattu ensisijaisesti web-palveluille. Google Data API:n sisältämät web-palvelut ovat suurimmaksi osaksi ilmaisia ja ne liittyvät usein suoraan Googlen tarjoamiin vastaaviin palveluihin. Google Data API sisältää API-rajapinnat (ks. kohdat 3.2 ja 3.3) seuraaville Google-palveluille:

- *Google Apps* -palvelu tarjoaa *palvelin* (domain) -ylläpitoa, jossa voidaan hyödyntää Googlen tarjoamia web-palveluja. Palvelu on tarkoitettu ensisijaisesti Premier- ja Education Edition -asiakkaille ja se sisältää sekä rajoitetun ilmaisen että maksullisen version.
- *Google Base* -palvelussa voidaan julkaista Google-hakuun listattavaa informaatiota.
- *Blogger* -palvelussa voidaan ylläpitää ja julkaista omia *blogeja* (blogs).
- *Google Calendar* -palvelussa voidaan ylläpitää ja jakaa kalenteritietoja.
- *Google Code Search* -palvelulla voidaan etsiä julkista lähdekoodia Internetistä.
- *Google Contacts* -palvelulla voidaan hallita yhteystietoja.
- *Google Health* -palvelussa voidaan hallita keskitetysti omaan terveyden tilaan liittyviä tietoja.
- *Google Notebook* -palvelulla voidaan hallita omaa Google-muistikirjaa.
- *Google Spreadsheets* -palvelulla voidaan käsitellä Google-dokumentit -palveluun tallennettuja laskentataulukkoasiakirjoja.
- *Picasa Web Albums* -palvelussa voidaan varastoida ja julkaista omia valokuvaalbumeja.

- *Google Documents List* -palvelulla voidaan tallentaa ja listata Google-dokumentit -palvelussa olevia erimuotoisia toimistoasiakirjoja.
- *YouTube* -palvelussa Google-käyttäjät voivat ladata ja katsella videotiedostoja.

Google Data API -rajapinnan käyttö vaatii ilmaisen Google-käyttäjätilin luomisen, koska Google-palvelut vaativat Google-tilin avulla tehtävän *tunnistautumisen*. Google-käyttäjätili saadaan luotua seuraavasti:

1. Siirry Google Accounts -tilipalveluun sivulle: ”https://www.google.com/accounts/Login”.
2. Valitse ”luo ilmainen tili”- tai ”luo tili” -toiminto.
3. Täytä lomakkeen tiedot ja hyväksy lomake. Tilin luontiin tarvitaan olemassaoleva sähköpostiosoite.

Vaihtoehtoisesti Google-käyttäjätili saadaan luomalla Gmail-sähköpostiosoite, jolloin Google-tili aktivoidaan sähköpostin luonnin yhteydessä. Google Data API:n käyttöönottoon ei tarvita muita toimenpiteitä.

Google Data API -rajapinta sisältää useita *asiakaskirjastoja* (client APIs), joita voidaan ladata Google Code -sivuston (”http://code.google.com/”) kautta. Google tarjoaa valmiita asiakaskirjastoja muun muassa Java-, PHP- ja .NET-ohjelmointikielille. Asiakaskirjastojen käyttö on vapaavalintaista, jolloin Google Data API:a voidaan käyttää halutessa myös kokonaan ilman asiakaskirjastoja. Google Code -sivustolta löytyy myös web-palvelukohtaista dokumentaatiota ja ohjelmakoodiesimerkkejä. Näiden lisäksi Google Code tarjoaa Google-keskusteluryhmät -yhteisöpalvelun, missä voi pyytää apua tai keskustella toisten Google Data API:n käyttäjien ja Googlen henkilökunnan kanssa.

Googlen Data API -web-palvelut tukevat *Google data API* -protokollan mukaisia web-palvelukutsuja, missä kutsupyynnöt tehdään HTTP-protokollalla REST-menetelmää mukaillen. REST-menetelmästä poiketen, Google data API -protokolla mahdollistaa myös HTTP-protokollan POST-, PUT- ja DELETE-metodien käytön, joilla voidaan suorittaa luontiin, päivitykseen ja poistoon liittyviä tapahtumia. Google-palvelut palauttavat *ATOM* (Nottingham & Sayre, 2005) ja *RSS* (Really Simple Syndication) -muotoisia (Winer, 2003) XML-dokumentteja hyödyntäen HTTP-POST-menetelmää.

Google Data API -web-palvelut eivät tue SOAP-muotoisien viestien lähetystä tai vastaanottoa.

3.1.2 Amazon Web Services API

Amazonin web-palveluissa keskeisenä lähtökohtana on liiketoiminnan harjoittaminen. Tämä näkyy myös tarjotuissa web-palveluissa. Amazon web-palvelut tarjoavat suoraan liiketoimintaan ja kaupankäyntiin liittyvää toiminnallisuutta sekä näitä täydentäviä tukitoimintoja. Amazon tarjoaakin web-palveluja liittyen sen omaan liiketoimintaan, hajautettuun laskentaan, hakutoimintoihin ja erilaisiin aputoimintoihin. Amazonin web-palvelujen käyttö on suurimmaksi osaksi maksullista, missä maksu perustuu yleisimmin *transaktio*-pohjaiseen laskutukseen. Amazon tarjoaa web-palvelujensa hyödyntämiseksi Amazon Web Services (AWS) API -rajapinnan (Amazon Web Services API, 2008), joka kattaa seuraavat web-palvelut:

- *Amazon Associates Web Service* -palvelu tarjoaa Amazonin tuotetieto- ja e-liiketoimipalvelujen toiminnallisuuden.
- *Amazon DevPay* -palvelu tarjoaa yksinkertaisen laskutus- ja tilinhallintapalvelun, jota voidaan käyttää Amazonin web-palveluita käyttävissä sovelluksissa. Palvelu on maksullinen.
- *Amazon Elastic Compute Cloud (Amazon EC2)* -palvelu on beta-asteella oleva web-palvelu, joka tarjoaa skaalautuvaa laskentakapasiteettia hajautetussa *pilvi* (cloud) -ympäristössä (Fitzgerald, 2008). Palvelu on maksullinen.
- *Amazon Flexible Payments Service (Amazon FPS)* -palvelu on rajoitettu beta-asteella oleva web-palvelu, joka tarjoaa kehittäjille maksupalvelutoiminnallisuutta. Palvelu on maksullinen.
- *Amazon Fulfillment Web Service (Amazon FWS)* -palvelu tarjoaa kauppiaille pääsyn Amazonin *täydennyspalveluihin* (fulfillment services), missä kauppiaat voivat lähettää tilaustietoja Amazonille, joka pakkaa ja lähettää tuotteet kauppiaan puolesta asiakkaalle.
- *Amazon Mechanical Turk* -palvelu on beta-asteella oleva palvelu, joka toimii kauppapaikkana ihmisälyä vaativalle työlle. Web-palvelu mahdollistaa yrityksille ohjelmallisen pääsyn Amazon Mechanical Turk -kauppapaikkaan.

- *Amazon SimpleDB* -palvelu on rajoitettu beta-asteella oleva web-palvelu, joka mahdollistaa reaaliaikaisten kyselyjen tekemisen rakenteisesta tietovarastosta. Palvelu toimii läheisessä kytköksessä Amazon Simple Storage Service- ja Amazon Elastic Compute Cloud -palvelujen kanssa. Palvelu on maksullinen.

- *Amazon Simple Storage Service (Amazon S3)* -palvelu toimii varastona Internetistä saatavalle tiedolle. Palvelu toimii läheisessä kytköksessä Amazon Elastic Compute Cloud -palvelun kanssa. Palvelu on maksullinen.

- *Alexa Site Thumbnail* -palvelu tarjoaa ohjelmallisen pääsyn Alexa-palvelun sisältämiin *miniatyyrikuviin* (thumbnails), jotka on kerätty WWW-sivustojen sisältämiltä kotisivuilta.

- *Alexa Top Sites* -palvelu tarjoaa pääsyn Alexa Traffic Rank -palvelun keräämiin WWW-sivustojen liikennemäärätietoihin ja niistä koostettuihin järjestyslistoihin. Palvelu on maksullinen.

- *Alexa Web Information Service* -palvelu tarjoaa pääsyn Alexa-palvelun verkkoliikenteestä ja verkon rakenteesta keräämiin tietovarastoihin. Palvelu on maksullinen.

- *Alexa Web Search* -palvelu tarjoaa ohjelmallisen pääsyn Alexan *web-hakumoottoriin* (web search engine), missä asiakassovellukset voivat hyödyntää hakumoottorin tuottamia hakutuloksia. Palvelu on maksullinen.

AWS API:n käyttö vaatii ilmaisen Amazon Web Services developer -käyttäjätilin. Amazon Web Services developer -tili saadaan luotua seuraavasti:

1. Siirry Amazon Web Services developer -tilipalveluun sivulle: ”https://aws-portal.amazon.com/gp/aws/developer/registration/index.html”

2. Syötä sähköpostiosoitteesi sähköpostiosoitteelle varattuun kenttään. Voit syöttää myös salasanasi, mikäli sinulla on olemassa oleva Amazon-käyttäjätili ja olet syöttänyt sähköpostikenttään kyseisen käyttäjätilin sähköpostiosoitteen. Muussa tapauksessa jätä salasasanakenttä tyhjäksi. Valitse tarpeen mukaan joko ”No, I am a new customer” tai ”Yes, I have a password” ja paina ”continue”-painiketta.

3. Mikäli sinulla ei ole aiempaa Amazon-käyttäjätunnusta, täytä kolmiosainen rekisteröintilomake suorittaaksesi rekisteröitymisen.

Amazon Web Services -tilin lisäksi osa palveluista vaatii myös palvelukohtaisen kirjautumisen, johon tarvitaan Amazon-käyttäjätunnus. Tällaisia palveluja ovat esimerkiksi Alexa Site Thumbnail-, Alexa Top Sites- ja Alexa Web Search -palvelut.

AWS API -rajapinta sisältää joitakin asiakaskirjastoja ja esimerkkiohjelmakoodeja, joita voidaan ladata AWS developer connection -sivuston (”http://developer.amazonwebservices.com/connect/index.jspa”) alaisuudessa sijaitsevasta resurssikeskuksesta. AWS tarjoaa asiakaskirjastoja ja esimerkkiohjelmakoodeja Java-, PHP-, Ruby- ja C#-ohjelmointikielille. Lisäesimerkkejä voi etsiä myös sivuston alaisesta yhteisökoodipalvelusta, jonne AWS:n käyttäjät voivat ladata omia ohjelmakoodejaan. Jokaiselle palvelulle löytyy myös palvelukohtainen tekninen dokumentaatio. AWS developer connection -sivusto sisältää edellä mainittujen lisäksi myös keskustelufoorumin, missä voidaan keskustella muiden AWS-kehittäjien kanssa.

Amazonin web-palvelut tukevat yleisesti REST-menetelmää ja SOAP-muotoisia viestejä. Palvelujen lähettämät paluuviestit ovat vastaavasti SOAP- tai HTTP-POST-menetelmällä tuotettuja XML-viestejä. Palveluissa esiintyy kuitenkin keskinäisiä eroavaisuuksia yhteystapojen välillä ja niiden tuki saattaa vaihdella palvelukohtaisesti. Esimerkiksi Amazon SimpleDB -palvelu mahdollistaa pyynnöissä myös HTTP-POST-menetelmän käytön ja Amazon DevPay -palvelu kykenee vastaanottamaan tietyissä tilanteissa HTML-lomakkeita.

3.1.3 Ebay API

Ebay laajentaa tarjontaansa myös web-palveluilla. Ebayn tarjoamat web-palvelut liittyvät suoraan eBayn liiketoimintaan ja täten kohdistuvat eBay-kauppapaikkaan. Ebay sisältää sekä ilmaisia että maksullisia API-rajapintoja. Ebay API -rajapinta (Ebay API, 2008) tarjoaa kolme toisiaan täydentävää web-palvelua:

- *eBay Shopping Web Services* -palvelu tarjoaa ostajan näkymän eBayn dataan, missä ostajan näkymä käsittää lukuoikeuden eBayn julkiseen dataan. Palvelu mahdollistaa esimerkiksi erilaiset tuote- ja jäsenprofiilihaut. Palveluun tehtävien kutsujen määrä on rajattu jokaista *IP* (Internet Protocol) -osoitetta kohden 5000 kappaleeseen päivää kohden.

- *eBay Trading Web Services* -palvelu tarjoaa myyjän näkymän eBayn dataan, missä myyjän näkymä käsittää tunnistetun pääsyn yksityiseen eBay-dataan. Palvelu mahdollistaa muun muassa tuotteiden listauksen, myyntitietojen hakemisen, ostotransaktioiden jälkeisen toiminnan hallinnan sekä yksityisen eBay-käyttäjätiedon hallinnan.
- *Research Web Services for eBay* -palvelu mahdollistaa eBayn historiatietojen tarkastelun. Tutkimuspalvelu tarjoaa esimerkiksi hintoihin liittyvän historiatietojen haun. Muista palveluista poiketen tämä ei ole eBayn hallinnoima palvelu. Tutkimuspalvelu sisältää ei-kaupalliseen käyttöön suunnatun ilmaisen Price Research API -rajapinnan ja kehittyneemän Advanced Research API -rajapinnan, joka vaatii maksullisen lisenssin.

Ebay API -rajapinnan käyttö vaatii ilmaisen eBay Developers Program -käyttäjätilin luonnin. Tämän lisäksi pitää luoda sovellusavaimet, joilla eBay tunnistaa web-palvelua käyttävät sovellukset. Sovellusavaimet saadaan luotua oman tilin hallinnointisivulta. Ebay Developers Program -käyttäjätili ja sovellusavaimet saadaan luotua seuraavasti:

1. Siirry eBay Developers Program -käyttäjätilin luomissivulle: ”https://developer.ebay.com/join/Default.aspx”.
2. Täytä lomake ja paina ”Join now”-painiketta.
3. Siirry ”My Account” -sivulle ja valitse ”application keys” -kohdasta joko ”Generate Sandbox Keys”- tai ”Generate Production Keys” -vaihtoehto. Valinnasta riippuen voit luoda joko testiavaimet tai tuotantoavaimet.

Ebay API -rajapinta sisältää asiakaskirjastot useille tukemilleen ohjelmointikielille, joista voidaan mainita esimerkkinä avoimen lähdekoodin eBay SDK for Java -kehityspaketti. Ebayn asiakaskirjastoja löytyy muun muassa Java-, PHP- ja .NET-ohjelmointikielille. Asiakaskirjastojen lisäksi eBay API -rajapinnalle löytyy esimerkkiohjelmakoodeja, joita on saatavilla myös useille ohjelmointikielille. Asiakaskirjastoja ja ohjelmakoodeja voidaan ladata eBayn developers program -sivustolta (”http://developer.ebay.com/”) löytyvästä kehityskeskukset palvelusta. Sivustolta löytyy myös Community-yhteisöpalvelu, joka sisältää muun muassa foorumin ja Codebase-palvelun, josta löytyy muiden yhteisöjäsenten lataamia esimerkkiohjelmakoodeja. Ebay API -rajapinnan palveluille löytyy myös erilaisia oppaita ja teknisiä dokumentteja.

Ebayn web-palvelujen käyttämät protokollat ja viestimuodot vaihtelevat käytetyn palvelun mukaan. Shopping Web Services -palvelu tukee REST- ja HTTP-POST-menetelmiä. Lisäksi palvelu mahdollistaa muun muassa XML-, SOAP- ja *JSON* (JavaScript Object Notation) -viestimuotojen käytön. Trading Web Services -palvelu tukee ainoastaan HTTP-POST-menetelmää *SSL* (Secure Sockets Layer) -suojatulla yhteydellä. Viestimuodoista tuetaan SOAP- ja XML-viestejä. Research Web Services for eBay -palvelu mahdollistaa puolestaan REST- ja HTTP-POST-menetelmien ja XML-muotoisien viestien käytön.

3.2 Google Data API -web-palvelurajapinta

Google Data API -rajapinta tarjoaa web-palvelurajapinnat useimmille Googlen palveluille. Tässä kohdassa esitellään Google Data API:n sisältämät palvelukohtaiset API-rajapinnat. Näiden lisäksi esitellään Google data API -protokollan mukaisien web-palvelukutsujen muodostaminen kuin myös protokollan tarjoamia XML-viestimuotoja, joita käytetään esiteltävissä API-rajapinnoissa. Tämä osuus käsittelee Google Data API (2008) Internet-sivustolla ja dokumentaatiossa kuvattuja tietoja.

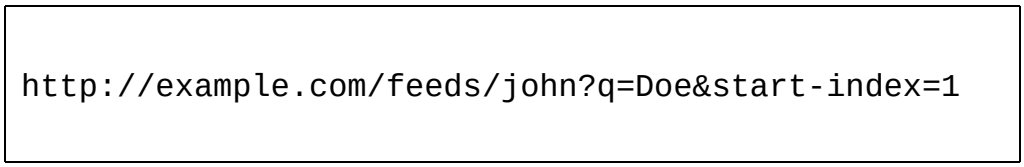
Google Data API -rajapinta koostuu sekä kaikille palveluille yhteisistä osista että palvelukohtaisista API-rajapinnoista. Google Data API sisältää seuraavat API-rajapinnat:

- *Google Apps API* -rajapinnat tarjoavat mahdollisuuden integroida ja laajentaa Googlen kommunikointi- ja yhteistoimintapalveluja. Tämä sisältää myös mahdollisuuden hyödyntää muita Google Data API:n web-palveluja.
- *Google Base data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja lisätä Google Base -palvelussa olevia tietoja.
- *Blogger data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja päivittää Blogger-palvelun sisältämiä tietoja.
- *Google Calendar data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja päivittää Google-kalenteri -palvelun kalenteritapahtumia.
- *Google Code Search data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea funktiomäärityksiä ja esimerkkilähdekoodeja Google Code Search -palvelulla.

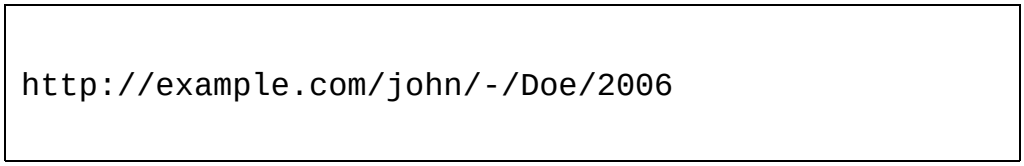
- *Google Contacts data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja päivittää Google-tilin kontaktitietoja.
- *Google Health data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja lähettää Google Health -palvelussa säilöttyä tietoa.
- *Google Notebook data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea Google-muistikirjan sisältämiä tietoja.
- *Google Spreadsheets data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja päivittää Google-dokumentit -palvelussa säilytettyjä laskentataulukkoasiakirjoja.
- *Picasa Web Albums data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja päivittää Googlen Picasa Web Albums -palvelussa olevia valokuva-albumeja ja valokuvia.
- *Google Documents List data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja ladata Google-dokumentit -palvelussa säilytettyjä toimistoasiakirjoja.
- *YouTube data API* -rajapinta tarjoaa asiakassovelluksille mahdollisuuden hakea ja lisätä YouTube-palvelussa olevia videotiedostoja.

Google Data API -protokolla mahdollistaa web-palvelujen kutsumisen REST-menetelmän mukaisesti. Tämän lisäksi tietoja voidaan luoda, päivittää ja poistaa HTTP:n POST-, PUT- ja DELETE-metodien avulla. REST-menetelmällä tehtävä web-palvelujen kutsuminen perustuu Googlen *kyselymalliin* (query model), missä lähetetään HTTP-GET-menetelmällä kyselymallin mukainen URL-osoite.

Web-palvelukutsun URI-osoite koostuu kutsuttavan web-palveluresurssin URI-osoitteesta, jota seuraa tarkentavat polkumääritteet ja kyselyparametrit. URL-osoite noudattaa tällöin muotoa: ”http://*palvelun osoite*/[*tarkentavat polkumääritteet*][*URL-parametrit*]”. Useimmat kyselyparametrit noudattavat perinteisiä URL-parametrien nimi–arvo-pareja, joiden käyttö on nähtävissä kuvan 20 esimerkissä. Poikkeuksena tähän ovat kategoriatyyppiset rajaavat parametrit, jotka muodostetaan tarkentamalla URL-osoitteen polkua kuvan 21 esimerkin mukaisesti.



Kuva 20: Esimerkki kyselyparametrien käytöstä kyselymallin mukaisessa URL-osoitteessa.



Kuva 21: Esimerkki kategoriarajoitteiden käytöstä kyselymallin mukaisessa URL-osoitteessa.

Kategoriakyselyt ilmaistaan sijoittamalla URL-osoitteen perään ”/-/”-merkkijono ennen URL-parametreja. Tämän perään tulee kategoriarajaukset muodossa: ”/*kategoria-nimi*/”. Kategoriahaut mahdollistavat useiden kategoriakriteerien käyttämisen. Kategoriahaku voidaan rakentaa seuraavien loogisten operaattorien avulla:

- AND-operaattori saadaan lisättyä yhdistämällä useita kategorioita peräkkäin muodossa: ”.../-/*kategoria1*/*kategoria2*/...”. Kategoriahaun AND-operaattorina toimii täten ”/”-merkki.
- OR-operaattori saadaan *URL-koodatulla* (URL-encoded) ”|”-merkillä. Tällöin kategoriat esitetään muodossa: ”.../-/*kategoria1*|*kategoria2*/...”. Koska ”|”-merkki on URL-koodattu, OR-operaattori näyttää URL:ssa seuraavanlaiselta: ”.../-/*kategoria1*%7C*kategoria2*/...”.
- NOT-operaattori saadaan lisäämällä ”-”-merkki välittömästi kategorian nimen eteen. Tällöin kategoriarajaus esitetään muodossa: ”../-/-*kategoria1*...”.

Kategoriakyselyjen lisäksi myös tekstihakuparametri ”q” toimii eräänlaisena erityistapauksena. Se muodostetaan tavallisena URL-parametrina, mutta sisällön rakentaminen vaatii joitakin erityistoimenpiteita, jotta tekstihakua voitaisiin käyttää tehokkaasti. Tekstihaku noudattaa Google-haun toimintaperiaatetta, missä etsitään kokonaisia hakusanoja kirjainkoosta välittämättä. Useamman hakusanan muodostamia hakulausekkeita saadaan lisäämällä lauseen alkuun ja loppuun lainausmerkit. Tällöin lauseke esitetään muodossa: ”q=”sana1 sana2”...”. Muilta osin tekstihaku voidaan rakentaa seuraavien loogisten operaattorien avulla:

- AND-operaattori saadaan lisättyä yhdistämällä useita hakusanoja peräkkäin muodossa: ”q=sana1 sana2 sana3”. Hakusanat erotetaan välilyönnillä, missä välilyönnit täytyy URL-koodata. Tällöin edellinen lauseke esitetään URL-koodatussa muodossa seuraavasti: ”q=sana1%20sana2%20sana3”.

- NOT-operaattori saadaan lisäämällä ”-”-merkki välittömästi hakusanan eteen. Tällöin hakusana esitetään muodossa: ”q=-sana1”.

Google Data API -rajapinta sisältää myös joitakin standardikyselyparametreja, jotka on kuvattu taulukossa 1. Näiden lisäksi palvelut sisältävät palvelukohtaisia parametreja, joita voidaan käyttää kyseisten web-palvelujen yhteydessä.

Google Data API -web-palvelut tukevat myös ehdollista HTTP-GET-pyyntöä, missä pyydettävän sisällön sijaista lähetetään 304 Not Modified -vastaus, mikäli pyydettävä sisältö ei ole muuttunut edellisestä pyynnöstä. Tämä tehdään siten, että web-palvelut asettavat Last-Modified-*vastausotsakkeen* (response header), jonka arvo pohjautuu web-palvelun palauttaman XML-dokumentin atom:updated-elementin arvoon. Web-palvelun asiakas voi lähettää tämän arvon takaisin If-Modified-Since-*pyyntöotsakkeen* (request header) arvona, mikäli ei haluta saada muuttumatonta sisältöä uudestaan.

Web-palveluille tehdyt kyselyt palauttavat XML-muotoisen vastauksen, joka voi sisältää Atom-syötteen, RSS-syötteen tai yksittäisen Atom-tietoelementin. Vastausten sisältö riippuu käytetystä web-palvelusta ja web-palvelukutsusta. Vastauksen muoto riippuu puolestaan pyynnön alt-parametrin arvosta. Vastauksen tiedot kapseloidaan joko *feed*- tai *channel*-elementin sisälle, riippuen käytetäänkö Atom- vai RSS-muotoa (kuva 22). Seuraavien openSearch-elementtien täytyy esiintyä välittömästi feed- ja channel-elementtien alaisuudessa:

- *openSearch:totalResults*-elementti sisältää kyselyn tuottaman tulosjoukon kokonaiskoon.

- *openSearch:startIndex*-elementti sisältää ensimmäisen tuloselementin indeksin, missä ensimmäisen indeksin arvo on yksi.

- *openSearch:itemsPerPage*-elementti sisältää sivulla näytettävien tulosten lukumäärän.

Atom-muotoiset vastaukset voivat sisältää myös ”application/atom+xml” -tyyppisen *link*-elementin, jonka rel-attribuutti voi sisältää seuraavat vakioarvot:

Taulukko 1: Google Data API -rajapinnan standardiparametrit. Kaikkien standardiparametrien arvot vaativat URL-koodauksen.

Parametri	**Tarkoitus**	**Kuvaus**
q	Tekstihaku-merkkijono	Sisältää hakusanat, joilla rajataan tulosjoukkoa.
/kategoria	Kategoriasuodatin	Rajaa tulosjoukkoa kategorioiden nimillä.
author	Tiedon laatija	Rajaa tulosjoukkoa tekijä- tai omistajatiedolla.
alt	Tiedon esitystyyli	Määrittää vastauksen palauttaman XML-dokumentin muodon. Arvolla ”rss” palauttaa RSS-muotoisen dokumentin. Arvolla ”atom” palauttaa Atom-muotoisen dokumentin. Mikäli parametria ei määritetä, palauttaa dokumentin oletusarvoisesti Atom-muotoisena.
updated-min, updated-max	Tiedon päivitysaikarajat	Parametrit sisältävät RFC 3339 -muotoiset aikaleimat, missä ensimmäinen aikaleima sisältyy hakuun, mutta jälkimmäinen ei.
published-min, published-max	Tiedon julkaisuaikarajat	Parametrit sisältävät RFC 3339 -muotoiset aikaleimat, missä ensimmäinen aikaleima sisältyy hakuun, mutta jälkimmäinen ei.
start-index	Tulosjoukon aloitusindeksi	Määrittää tulosjoukolle aloitusindeksin, joka alkaa yhdestä.
max-results	Tulosjoukon maksimikoko	Määrittää palautettavan tulosjoukon maksimikoon.
entryID	Haettavan tietoelementin yksilöivä tunnus	Yksilöi haettavan tiedon. Mikäli yksilöivä tunnus määritetään, pyynnössä ei voida käyttää muista parametreja. Kyselyssä entryID korvataan parametrin arvolla samaan tapaan kuin kategoriahaussa.

```
Atom-vastaus:

<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
  xmlns:openSearch="http://a9.com/-/
            spec/opensearchrss/1.0/">
  ...
</feed>

RSS-vastaus:

<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:atom='http://www.w3.org/2005/Atom"
     xmlns:openSearch="http://a9.com/-/spec/
           opensearchrss/1.0/"
     version="2.0">
  <channel>
    ...
  </channel>
</rss>
```

Kuva 22: Esimerkit vastausviestien kapseloinnista sekä Atom- että RSS -syötteillä, missä nimiavaruudet määräytyvät xmlns-attribuuteissa määritettyjen nimiavaruuksien mukaisesti. Atom-vastauksen elementit kapseloidaan feed-elementin sisälle. RSS-vastaukset kapseloidaan puolestaan RSS- ja channel-elementtien sisälle.

- Vakio ”http://schemas.google.com/g/2005#feed” osoittaa linkin, missä määritetään täydellisen Atom-syötteen sijainti.
- Vakio ”http://schemas.google.com/g/2005#post” osoittaa linkin, missä määritetään Atom-syötteen POST-URI-osoite, johon uudet tietoelementit voidaan lähettää.
- Vakio ”self” osoittaa kyseisen link-elementin omaavan resurssin. Tällöin type-attribuutin arvo riippuu pyydetystä muodosta. Toisen GET-pyynnön lähetys tä-

hän osoitteeseen palauttaa saman tuloksen, mikäli data ei ole muuttunut pyyntöjen välillä.

- Vakio ”previous” osoittaa linkin, missä määritetään tulosjoukon edellinen osa, mikäli tulosjoukko on pilkottu useampaan osaan.
- Vakio ”next” osoittaa linkin, missä määritetään tulosjoukon seuraava osa, mikäli tulosjoukko on pilkottu useampaan osaan.
- Vakio ”edit” osoittaa linkin, missä määritetään tietoelementin muokkausosoite, johon päivitetyt tietoelementit voidaan lähettää.

Google Data API tarjoaa myös mekanismin samanaikaisten muutosten hallintaan. Tätä voidaan kutsua myös *versioinniksi*. Joissakin tilanteissa on tärkeää, ettei asiakassovellukset pääse ylikirjoittamaan toistensa tietoja, kun samaa tietoa päivitetään yhtäaikaisesti. Tämän estämiseksi Google Data API tarjoaa mekanismin, missä päivityksen täytyy kohdistua samaan versioon kuin mitä asiakassovellus on käsittelemässä. Mikäli toinen asiakassovellus päivittää samaa tietoelementtiä ennen ensimmäistä asiakassovellusta, ensimmäisen asiakassovelluksen tekemä päivitys estetään, koska asiakassovelluksen käsittelemä versio ei ole enää sama kuin palveluun tallennettu. Versiointia tukevissa syötteissä versionumero tallennetaan kunkin tietoelementin muokkausosoitteeseen. Tämä ei vaikuta tietoelementin yksilöivään tunnistetietoon. Versiointi pätee päivityksen ohella myös poistotapahtumiin.

Mikäli syöte tukee versiointia, palvelin palauttaa PUT- ja DELETE-metodipyynnöillä konfliktitilanteissa 409 Conflict -vastauksen. Tällöin vastauksen runko sisältää konfliktin aiheuttaneesta tietoelementista validin uusimman version. Tämän jälkeen asiakassovellus voi muokata saatua tietoelementtiä ja lähettää sen uudestaan. Vastaukset voivat sisältää taulukossa 2 kuvattuja HTTP-statuskoodeja.

Google Data API mahdollistaa *istunnon* (session) hallinnoinnin sekä *evästeillä* (cookies) että *tunnistemerkkijonolla* (token), missä jälkimmäinen voidaan lähettää pyynnön mukana parametrina. Mikäli käytetään Googlen asiakaskirjastoja, asiakassovelluksen ei tarvitse välittää Google-istunnon tilasta.

3.3 Google Data API:n käyttö asiakassovelluksissa

Google Data API sisältää asiakaskirjastot useimmille Googlen web-palveluille. Tässä kohdassa esitellään Googlen web-palvelujen käyttöä Google Data API (2008)- ja Picasa Web Albums Data API (2008) -rajapintojen Java-asiakaskirjastoilla. Asiakaskirjastojen sisältämistä toiminnoista esitellään tietojen hakua, lisäystä, päivitystä ja poistoa. Asiakaskirjastojen käyttöä havainnollistetaan Java-ohjelmakoodiesimerkkien avulla, jotka pohjautuvat Googlen Java-asiakaskirjaston jakeluversioon 1.15.0.

Picasa API -rajapinnan Java-asiakaskirjasto sisältää joitakin Java-luokkia ja metodeja, joita voidaan käyttää kaikkien Picasa-palvelun sisältämien web-palvelukutsujen yhteydessä. Tärkein yksittäinen luokka on com.google.gdata.client.photos-paketin PicasawebService. Luokka perii kaikille palveluille yhteisen Service-kantaluokan, joka sijaitsee com.google.gdata.client-paketissa. Service-luokka sisältää muun muassa seuraavat julkiset kutsumetodit:

- *getFeed*-metodeilla saadaan pyydettyä syötteitä, jotka voivat sisältää tieto- tai metatietoelementtejä. Metodi lähettää pyynnön ja palauttaa pyynnön mukaisen

Taulukko 2: Google Data API:n käyttämät HTTP-statuskoodit

Koodi	**Kuvaus**
200 OK	Ei virhettä.
201 CREATED	Resurssin luominen onnistui.
304 NOT MODIFIED	Resurssi ei ole muuttunut If-Modified-Since-otsakkeessa määritetystä ajankohdasta.
400 BAD REQUEST	Virheellinen pyynnön URI-osoite, otsake tai tuntematon standardista poikkeava parametri.
401 UNAUTHORIZED	Pyyntö vaatii tunnistautumisen.
403 FORBIDDEN	Ei-tuettu standardin mukainen parametri tai tunnistautuminen epäonnistui.
404 NOT FOUND	Pyydettyä resurssia ei löydy.
500 INTERNAL SERVER ERROR	Sisäinen virhe. Tätä käytetään kaikkien tunnistamattomien virheiden kanssa.

syötteen, joka kapseloidaan jonkin syöteluokan ilmentymään.

- *getEntry*-metodeilla saadaan pyydettyä yksittäisiä Atom-tietoelementtejä. Metodi tekee pyynnön ja palauttaa pyynnön mukaisen tietoelementin, joka kapseloidaan jonkin tietoelementtiluokan ilmentymään.

- *insert*-metodilla voidaan lisätä uusi tietoelementti. Metodi lähettää pyynnön ja palauttaa lisätyn tietoelementin.

- *update*-metodilla voidaan päivittää tietoelementin tietoja. Metodi lähettää pyynnön ja palauttaa päivitetyn tietoelementin.

- *delete*-metodilla voidaan poistaa valittu tietoelementti. Metodi lähettää poistopyynnön web-palvelulle.

Palveluluokkien lisäksi asiakaskirjastot sisältävät myös syöte- ja tietoelementtiluokkia. Kaikki syöteluokat pohjautuvat com.google.gdata.data-paketin BaseFeed-luokkaan, joka periytetään jälkeläisluokille luokkahierarkian mukaisesti. Picasa API:n kanssa voidaan käyttää esimerkiksi UserFeed-, AlbumFeed- ja PhotoFeed-syöteluokkia, riippuen minkä tyyppistä tietoa ollaan hakemassa. UserFeed-luokalla voidaan hakea metatietoa käyttäjästä, AlbumFeed-luokalla saadaan valokuva-albumeihin liittyviä tietoja ja PhotoFeed palauttaa puolestaan valokuvien tietoja. Tietoelementit pohjautuvat puolestaan com.google.gdata.data-paketin BaseEntry-luokkaan, jonka jälkeläisluokat perivät sen luokkahierarkian mukaisesti. Tietoelementtiluokat kykenevät myös päivittämään ja poistamaan omia tietojaan. BaseEntry-luokka tarjoaa *update*- ja *delete*-metodit, joilla voidaan päivittää tai poistaa asianomainen tietoelementti. Metodit suorittavat tarvittavat kutsut web-palveluun, jolloin kutsuihin ei tarvitse käyttää palveluluokkia. Picasa API:n kanssa voidaan käyttää esimerkiksi UserEntry-, AlbumEntry- ja PhotoEntry-tietoelementtiluokkia.

Googlen web-palveluista voidaan hakea tietoa REST-menetelmän mukaisilla web-palvelukutsuilla. Lähetettyyn pyyntöön saadaan vastausviesti, joka sisältää parametrissa ilmaistun syötetyypin mukaisen XML-vastausdokumentin. Esimerkiksi jos pyydetään ”/myFeed” -osoitteen mukaista syötettä ”GET /myFeed” -pyynnöllä ja pyydetty syöte ei sisällä *entry*-tietoelementtejä, palvelin vastaa pyyntöön kuvan 23 esimerkin mukaisesti. Esimerkin tapauksessa palvelulta saadaan tietoa, vaikka syöte ei sisälläkään tietoelementtejä. Tämä johtuu siitä, että syöte voi sisältää tietoelementtien lisäksi

myös metatietoa. Tämä tilanne voi esiintyä esimerkiksi silloin, kun haetaan valokuvia käyttäjältä, jolla on olemassa aktivoitu Picasa-käyttäjätili, mutta siellä ei ole valokuvia.

```
200 OK

<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Foo</title>
  <updated>2006-01-23T16:25:00-08:00</updated>
  <id>http://www.example.com/myFeed</id>
  <author>
    <name>John Doe</name>
  </author>
  <link href="/myFeed" rel="self"/>
</feed>
```

Kuva 23: Esimerkki palvelun lähettämästä vastausviestistä, kun nimettyä myFeed-syötettä haetaan GET-metodilla, missä myFeed kuvaa syöte-URL-osoitetta.

Picasa API:n asiakaskirjastolla syöte saadaan haettua kuvan 24 esimerkin mukaisesti, jossa suoritetaan seuraavat toimenpiteet (Kinnunen, 2008):

1. Muodostetaan URL-olio käyttäen syötteen osoitetta.
2. Luodaan PicasawebService-luokan ilmentymä, jolle annetaan sovellustunnus.
3. Asetetaan tunnistautumista varten Google-tilin käyttäjätunnus ja salasana
4. Lähetetään pyyntö ja vastaanotetaan vastausviestin palauttama syöte, missä UserFeed-luokalla kerrotaan haettavan syöteilmentymän tyyppi. Feed-luokka on puolestaan BaseFeed-luokan perivä *geneerinen* syöteluokka, johon palautettava syöte kapseloidaan.

Googlen web-palveluihin voidaan lisätä uusia tietoelementtejä HTTP-POST-menetelmän mukaisilla web-palvelukutsuilla. Lähetettyyn XML-muotoiseen pyyntöviestiin saadaan vastausviesti, joka sisältää käytetyn syötteen mukaisen XML-

```
1: URL feedUrl =
      new URL("http://picasaweb.google.com/" +
               "data/feed/api/user/john");
2: PicasawebService myService =
      new PicasawebService("myappId");
3: myService.setUserCredentials(
                     "john",
                     "mypassword");
4: Feed myFeed =
      myService.getFeed(feedUrl, UserFeed.class);
```

Kuva 24: Esimerkki käyttäjäsyötteen hausta Picasa-web-palvelussa.

vastausdokumentin. Kun lähetetään kuvan 25 esimerkin mukainen HTTP-POST-pyyntö, palvelulta saadaan vastaukseksi kuvan 26 mukainen vastausviesti.

```
POST /myFeed

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <author>
    <name>John Doe</name>
    <email>john@gmail.com</email>
  </author>
  <title type="text">Entry 1</title>
  <content type="text">My entry</content>
</entry>
```

Kuva 25: Esimerkki palvelun asiakkaan lähettämästä HTTP-POST-pyynnöstä, kun ollaan lisäämässä uutta tietoelementtiä.

Kun asiakassovelluksesta lähetetään XML-viestejä HTTP-POST-menetelmällä, pitää huomioida, että id-, link- ja updated-elementtejä ei sisällytetä lähetettävään viestiin. Sovelluspalvelin täydentää nämä elementit vastausviestin luonnin yhteydessä.

```
201 CREATED

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>http://www.example.com/id/1</id>
  <link rel="edit"
        href="http://example.com/myFeed/1/1/"/>
  <updated>2006-01-23T16:26:03-08:00</updated>
  <author>
    <name>John Doe</name>
    <email>john@gmail.com</email>
  </author>
  <title type="text">Entry 1</title>
  <content type="text">My entry</content>
</entry>
```

Kuva 26: Esimerkki palvelun lähettämästä vastausviestistä, kun uusi tietoelementti lisättiin onnistuneesti.

Picasa API:n asiakaskirjastolla tietoelementit saadaan lisättyä kuvan 27 esimerkin mukaisesti, jossa suoritetaan seuraavat toimenpiteet (Kinnunen, 2008):

1. Muodostetaan URL-olio käyttäen syötteen osoitetta.
2. Luodaan uusi albumiolio ja asetetaan sille nimi.
3. Lähetetään lisäyspyyntö ja vastaanotetaan vastausviestinä saatu lisätty albumi.

Googlen web-palvelujen sisältämiä tietoelementtejä voidaan päivittää HTTP-PUT-metodilla. Päivityksessä täytyy käyttää palvelun luomaa tietoelementin ”<link rel=”edit” ...>”-muokkauslinkkiä. Mikäli käytetty palomuuri estää HTTP-PUT-metodin käytön, voidaan käyttää HTTP-POST-menetelmää, jolloin metodin yliajo-otsake täytyy asettaa seuraavasti: ”X-HTTP-Method-Override: PUT”. Lähetettyyn XML-muotoiseen pyyntöviestiin saadaan vastausviesti, joka sisältää käytetyn syötteen mukaisen XML-vastausdokumentin. Kun lähetetään kuvan 28 esimerkin mukai-

```
1: URL postUrl =
      new URL("http://picasaweb.google.com/" +
              "data/feed/api/user/john/" +
              "private/full");
...
2: AlbumEntry newEntry = new AlbumEntry();
   newEntry.setTitle(
            new PlainTextConstruct(
                        "My photo album" ) );
3: AlbumEntry insertedEntry =
      myService.insert(postUrl, newEntry);
```

Kuva 27: Esimerkki uuden valokuva-albumin lisäyksestä Picasa-web-palvelussa.

nen HTTP-POST-pyyntö, palvelulta saadaan vastaukseksi kuvan 29 mukainen vastausviesti. Esimerkkikuvista on myös nähtävissä, että muokkauslinkin osoite on vaihtunut vastauksessa. Osoite päättyy ”/2/”-merkkijonoon, kun pyynnössä se päättyi ”/1/”-merkkijonoon. Tämän taustalla on samanaikaisten muutosten välttämiseksi tehtävä versiointi, jolloin muuttuva luku on tietoelementin versionumero.

Picasa API:n asiakaskirjastolla tietoelementti saadaan päivitettyä kuvan 30 esimerkin mukaisesti, jossa suoritetaan seuraavat toimenpiteet (Kinnunen, 2008):

1. Haetaan muokkausosoitteen sisältävä valokuva-albumi.
2. Luodaan valokuva-albumin muokkausosoitteesta URL-olio.
3. Lähetetään muokkauspyyntö ja vastaanotetaan muokattu albumi.

Googlen web-palvelujen sisältämiä tietoelementtejä voidaan poistaa HTTP-DELETE-metodilla. Tietoelementin poistossa täytyy käyttää palvelun luomaa tietoelementin muokkauslinkkiä. Mikäli käytetty palomuuri estää HTTP-DELETE-metodin käytön, voidaan käyttää HTTP-POST-menetelmää, jolloin metodin yliajo-otsake täytyy asettaa seuraavasti: ”X-HTTP-Method-Override: DELETE”. Lähetettyyn DELETE-pyyntöön saadaan vastauksena 200 OK -statuskoodi, mikäli poisto onnistui. Muussa tapauksessa palautetaan jokin HTTP-virhekoodi.

```
PUT /myFeed/1/1/

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>http://www.example.com/id/1</id>
  <link rel="edit"
        href="http://example.com/myFeed/1/1/"/>
  <updated>2006-01-23T16:28:05-08:00</updated>
  <author>
    <name>John Doe</name>
    <email>john@gmail.com</email>
  </author>
  <title type="text">Entry 1</title>
  <content type="text">Edited entry.</content>
</entry>
```

Kuva 28: Esimerkki palvelun asiakkaan lähettämästä HTTP-PUT-pyynnöstä, kun päivitetään tietoelementtiä.

Picasa API:n asiakaskirjastolla tietoelementti saadaan poistettua kuvan 31 esimerkin mukaisesti, jossa suoritetaan seuraavat toimenpiteet (Kinnunen, 2008):

1. Luodaan valokuva-albumin muokkausosoitteesta URL-olio.
2. Lähetetään poistopyyntö web-palvelulle.

3.4 Web-palvelujen koostaminen välityspalvelurajapinnalla

Tässä kohdassa esitetyt tiedot perustuvat Busslerin (2006) esittämiin tietoihin. Luvussa 4 periaatteita sovelletaan käytäntöön rakentamalla välityspalvelurajapinta ulkoisten web-palvelujen kutsumiseksi Kinnusen (2008) esittämällä tavalla.

Yksittäisten web-palvelujen kutsuminen onnistuu palvelun tarjoamilla asiakaskirjastoilla tai suoraan palvelun WSDL-dokumentin avulla. Palvelujen hyödyntäminen käy

```
200 OK

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>http://www.example.com/id/1</id>
  <link rel="edit"
        href="http://example.com/myFeed/1/2/"/>
  <updated>2006-01-23T16:28:05-08:00</updated>
  <author>
    <name>John Doe</name>
    <email>john@gmail.com</email>
  </author>
  <title type="text">Entry 1</title>
  <content type="text">Edited entry.</content>
</entry>
```

Kuva 29: Esimerkki palvelun lähettämästä vastausviestistä, kun päivitetään tietoelementtiä.

hankalammaksi silloin, kun halutaan hyödyntää useita palveluja. Esimerkiksi yhdestä web-palvelusta voidaan varata matka, toisesta lento ja kolmannesta hotellimajoitus. Koska palvelut ovat erillisiä, niiden toteutuksessa ja rajapinnoissa saattaa olla huomattaviakin eroja. Toisaalta saatetaan haluta niputtaa toisestaan poikkeavia palveluja yhdeksi palvelukokonaisuudeksi. Esimerkiksi voidaan tarjota portaalipalvelua, josta löytyy niin uutisia, säätietoja kuin pörssikurssejakin. Asikassovellusten tekeminen voi muodostua tällöin työlääksi, kun pahimmillaan jokaisen palvelun tietotyypit ja kutsurakenteet joudutaan huomioimaan erikseen.

Edellisistä esimerkkitilanteista voimme nähdä, että kyseisiä tilanteita varten tarvittaisiin toisistaan poikkeavat ratkaisut. Matkanvaraustapausta varten voimme koostaa palveluja syvyyssuuntaisesti, jossa useita peräkkäisiä kutsuja yhdistetään yhdeksi web-palvelukutsuksi. Kutsumme tätä *orkestroinniksi*. Syvyysuuntainen koostaminen tapahtuu täten orkestroimalla palveluja, jolloin palvelukutsut tehdään tietyssä järjestyksessä halutun koostetun lopputuloksen aikaansaamiseksi. Tällöin koostaminen voidaan näh-

```
...
1: AlbumEntry albumEntry =
     myService.getEntry(entryUrl, albumEntry.class);
   albumEntry.setTitle(
          new PlainTextConstruct(
                  "My new photo album") );
...
2: URL editUrl =
      new URL(
          albumEntry.getEditLink().getHref());
3: AlbumEntry updatedEntry =
      myService.update(editUrl, albumEntry);
```

Kuva 30: Esimerkki valokuva-albumin tietojen päivityksestä Picasa-web-palvelussa.

```
...
1: URL deleteUrl =
      new URL(
          albumEntry.getEditLink().getHref());
2: myService.delete(deleteUrl);
```

Kuva 31: Esimerkki valokuva-albumin poistamisesta Picasa-web-palvelussa.

dä prosessina, missä saatu syöte muunnetaan prosessin kuluessa halutuksi tulosteeksi. Portaaliesimerkin kohdalla voimme koostaa palveluja leveyssuuntaisesti, jossa tuodaan tarjolle useita rinnakkaisia web-palveluja. Tätä kutsumme puolestaan *palvelujen koostamiseksi*. Yksittäisistä palveluista voidaan täten koostaa palvelukirjastoja. Näin ollen koostamista voidaan tehdä sekä syvyys- että leveyssuunnassa. Molemmilla koostamistavoilla saadaan tulokseksi *koostepalvelu*, joka toimii *välityspalveluna* asiakkaan ja koostettujen web-palvelujen välissä.

Busslerin mukaan web-palvelut voidaan käsittää *heterogeenisiksi*, *autonomisiksi* ja *jaetuiksi HAD* (Heterogenous, Autonomous, Distributed) -järjestelmiksi, missä HAD-järjestelmät määritetään seuraavasti:

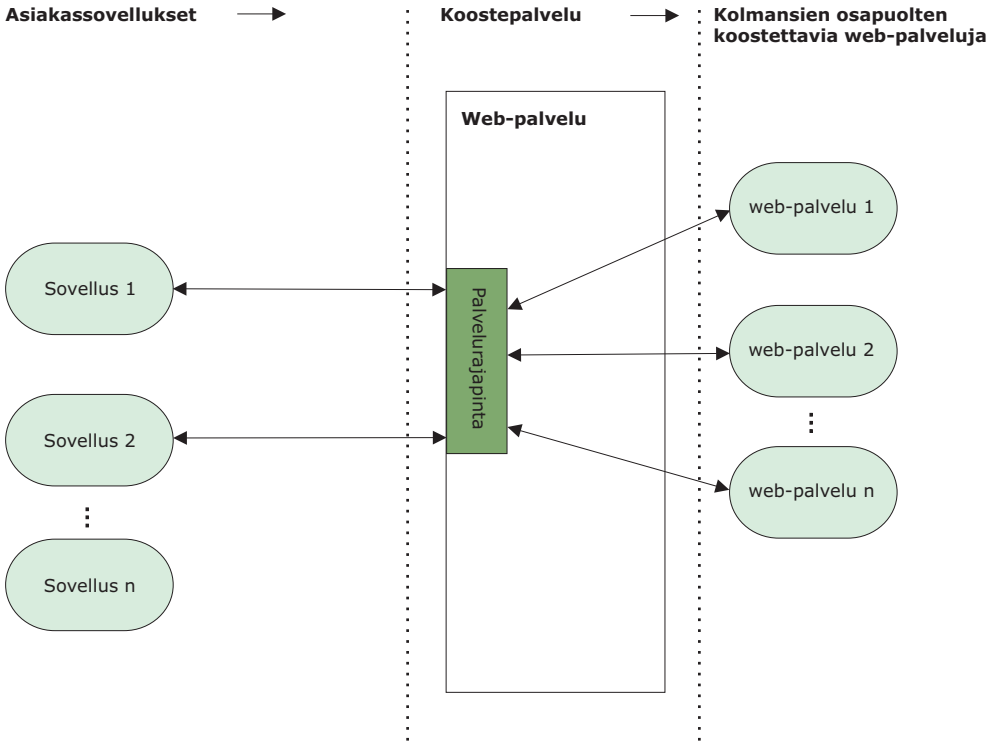
- Heterogeeniset web-palvelut määrittävät itsenäisesti omat viestiskeemansa.
- Autonomiset web-palvelut voivat muuttaa omaa rajapintaa ja semantiikkoja milloin tahansa ja ilman varoitusta.
- Jaetut web-palvelut voivat jakaa saman suoritustilan keskenään. Tämä voi tulla kysymykseen esimerkiksi silloin, kun web-palveluilla on sama palvelun tarjoaja.

Edellisten lisäksi Bussler esittää myös *homogeenisen* web-palvelun käsitteen: Yksittäinen web-palvelu voidaan määrittää homogeeniseksi, koska se on toteutettu yhden *tietomallin* (data model) mukaisesti. Homogeenisen web-palvelun kutsussa web-palvelun asiakas ja web-palvelun tarjoaja jakavat saman tietomallin, jolloin ei tarvita erillisiä tietotyyppimuunnoksia. Voimme laajentaa homogeenisen web-palvelun määrityksen koskemaan myös useita toisistaan erillisiä web-palveluja, jotka jakavat keskenään saman tietomallin. Esimerkkinä voidaan mainita Googlen Data API -rajapinta, joka koostuu useista yksittäisistä web-palveluista, jotka jakavat keskennään saman *ontologian*. *Homogeenisessa koostepalvelussa* kaikki kutsuttavat web-palvelut jakavat saman tietomallin toteuttaen palvelujen välisen ontologian käsitteet, jolloin itse koostepalvelussa ja sen koostamissa kohdepalveluissa käsitellään tietotyyppejä samalla tavalla. Esimerkiksi kaikki koostepalvelun kutsumat web-palvelut käsittelevät päiväyksiä samassa muodossa, jolloin päiväysmuodoista käytetään aina yhtä esitys- ja käsittelytapaa.

Web-palvelukutsu on heterogeeninen, mikäli web-palvelun asiakas ja web-palvelun tarjoaja noudattavat toisistaan eroavia tietomalleja. Tässä tapauksessa web-palvelun asiakkaan täytyy sovittaa viestit web-palvelun ymmärtämään muotoon. Samoin myös vastaanotetut viestit joudutaan muuntamaan sopivaan muotoon. Tämä on mahdollista, koska palvelun WSDL-dokumentti sisältää tarvittavat tiedot palvelun tietotyyppien ja viestien muodosta. Palvelun tarjoaja ei ole tietoinen tehdyistä viestimuunnoksista, koska ne on tehty ennen web-palvelukutsujen suorittamista. Tällöin palvelun tarjoaja ei joudu muuttamaan web-palvelun rajapintaa. *Heterogeeninen koostepalvelu* ei voi olettaa, että palvelun käyttämät web-palvelut käyttävät samoja tietotyyppimäärityksiä ja tyyppimääritysten tulkintoja. Esimerkiksi globaalisti toimiva matkatoimisto voi joutua tekemään varauksia useilla eri valuutoilla ja päiväysmuodoilla, riippuen mihin maihin matka suuntautuu. Tällöin myös eri maissa sijaitsevat web-palvelut voivat käyttää eriäviä valuutta- ja päiväysmuotoja. Tällöin koostepalvelun täytyy muuntaa valuutta- ja päiväysmuodot kullekin palvelulle sopiviksi. Tätä muuntamisprosessia kutsutaan *viestivälitykseksi* (message mediation). Koostepalvelun täytyy tiedostaa koostetuissa pal-

veluissa käytetyt tietotyypit. Lisäksi koostepalvelun täytyy kyetä muuntamaan erityyppiset viestimuodot ja tietotyypit, jotta se pystyisi välittämään tiedot koostetuille palveluille oikeassa muodossa. Vastaavasti koostetuista palveluista saadut vastausviestit pitää pystyä muuntamaan koostepalvelun itsensä ja mahdollisesti muiden koostettujen palvelujen ymmärtämään muotoon. Koostepalvelu koostuu täten heterogeenisista ja koostetuista web-palvelukutsuista.

Välityspalveluina toimivat koostepalvelut sekä saavat web-palvelukutsuja että tekevät niitä kolmansiin web-palveluihin. Täten koostepalveluun liittyy kaksi rajapintakerrosta, joista ensimmäinen on koostepalvelun oma asiakkaalle näkyvä palvelurajapinta. Toinen rajapintakerros muodostuu puolestaan koostepalvelun kutsumien koostettujen web-palvelujen rajapinnoista. Kuvassa 32 on esitetty koostepalvelun näkyminen välityspalveluna asiakassovelluksen ja koostettujen web-palvelujen välissä.



Kuva 32: Koostepalvelun kytkeytyminen asiakkaisiin ja koostettuihin web-palveluihin.

Orkestrointi piilottaa asiakkaalta prosessin, jolloin toimeenpiteen suoritus yksinkertaistuu, eikä asiakassovelluksen tarvitse välittää eri heterogeenisten web-palvelujen keskenään eroavista tietomalleista. Samaan päästään myös leveyssuuntaisella koostamisella. Leveyssuuntaisella koostamisella voidaan tarjota koostetuille web-palveluille yhtenäinen rajapinta, jolloin asiakassovelluksen tarvitsee käyttää ainoastaan yhtä tie-

tomallia. Koostepalvelun täytyy hallita kummassakin tapauksessa kutsumiensa web-palvelujen toisistaan eroavat tietomallit ja muuntaa ne yhtenäiseen muotoon, jota asiakassovellus voi käyttää.

Viestivälityksellä varmistetaan, että viestit voidaan välittää koostepalvelun ja koostettujen palvelujen välillä. Viestinvälitys mahdollistaa näin ollen erilaiset ontologiat omaavien heterogeenisten web-palvelujen koostamisen yhdeksi koostepalveluksi. Koostepalvelun pitää täten kyetä muuntamaan viestit eri ontologioiden välillä. Jotta tämä olisi mahdollista, koostepalvelun täytyy toteuttaa seuraavat asiat:

- Koostepalvelun täytyy ymmärtää koostamiensa web-palvelujen viestien rakenne ja merkitys.
- Koostepalvelun täytyy osata muuntaa koostamiensa web-palvelujen viestit ja rakenteet.

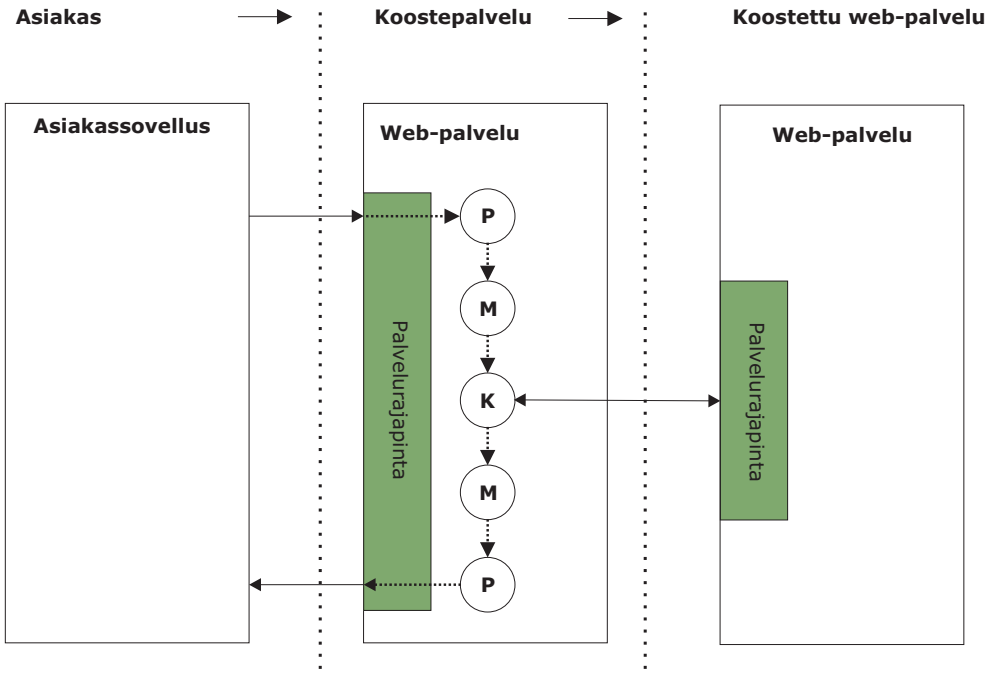
Viestien rakenne ja tietotyypit saadaan selvitettyä web-palvelujen WSDL-dokumenttien XML-skeemoista. Viestien merkitys perustuu ontologiaan, joka on web-palvelun toteutuksen perustana. Web-palvelun toteutus tulkitsee viestien sisältöä tietomallinsa mukaisesti. Esimerkiksi web-palvelun ontologia määrittää suomalaisen ”pp.kk.vvvv” -muotoisen päiväysmuodon ja tulkitsee saapuvat päivämäärämuodot sen mukaisesti, vaikka merkkijonotyyppi mahdollistaisikin myös muunlaiset päiväysmuodot.

Viestien muuntamisessa luodaan uusi kohdeviesti, joka pohjautuu saatuun lähdeviestiin. Tämä mahdollistaa lähdeviestin rakenteen ja tarkoituksen muuntamisen kohdeviestissä käytettyyn muotoon. Koska lähdeviestin lähettänyt järjestelmä ja kohdeviestin vastaanottava tunnistavat omat viestinsä, eriävien tietomallien mukanaan tuoma ongelma saadaan ratkaistua muunnoksella. Muunnokset voidaan suorittaa usealla toteutustavalla. Joku voi tehdä muunnoksen tekemällään ohjelmalla ja toinen voi käyttää *XSLT* (Extensible Stylesheet Language Transformations) -sääntöjä. Valitulla toteutustekniikalla ei ole merkitystä, mikäli itse tiedon merkitys ei muutu muunnoksessa. Mikäli lähde- ja kohdeviestit jakavat saman ontologian muunnosta ei tarvita lainkaan. Bussler (2006) esittää kaksi lähestymistapaa viestimuunnosten tekemiseksi:

- *avoin* eli välitön muunnos (inline step)
- *rajapinta-abstraktio* (interface abstraction)

Avoimen muunnoksen lähestymistavassa koostepalvelun tekemä viestimuunnos tehdään erillisenä vaiheena ennen web-palvelukutsua. Tällöin ennen web-palvelukutsua tulisi suorittaa erillinen muunnosvaihe, joka muuntaa koostepalvelun viestin kohdepalvelun ymmärtämään muotoon. Vastaavasti kaksisuuntaisessa kutsussa tulee tehdä muunnos myös paluuviestin vastaanottamisen jälkeen (kuva 33). Bussler esittää seuraavat havainnot liittyen avoimeen muunnokseen:

- *Muunnosvaiheiden lukumäärä*. Jokainen web-palvelukutsu vaatii yhden tai kaksi muunnosvaihetta, jotta viestimuunnos saadaan suoritettua.
- *If-then-else-rakenteet*. Jokaiselle tietotyypille ja tulkinnalle täytyy tehdä omat muunnoksensa. Esimerkiksi päivämäärämuotoja saattaa olla useita erilaisia. Tällöin jokainen tulkinta vaatii oman tarkistuksensa, jolla ohjataan käyttämään oikeantyyppistä muunnosta.
- *Koostepalvelun viestiskeeman kasvaminen*. Koostepalvelun täytyy hallita omien viestiskeemojensa lisäksi myös kaikkien koostettujen web-palvelujen viestiskeemat. Tämä johtuu siitä, että web-palvelukutsussa viitataan koostetun palvelun viestiskeemaan.



Kuva 33: Viestimuunnos avoimella muunnoksella, missä P = web-palvelun päätepiste, M = muunnos ja K = web-palvelukutsu.

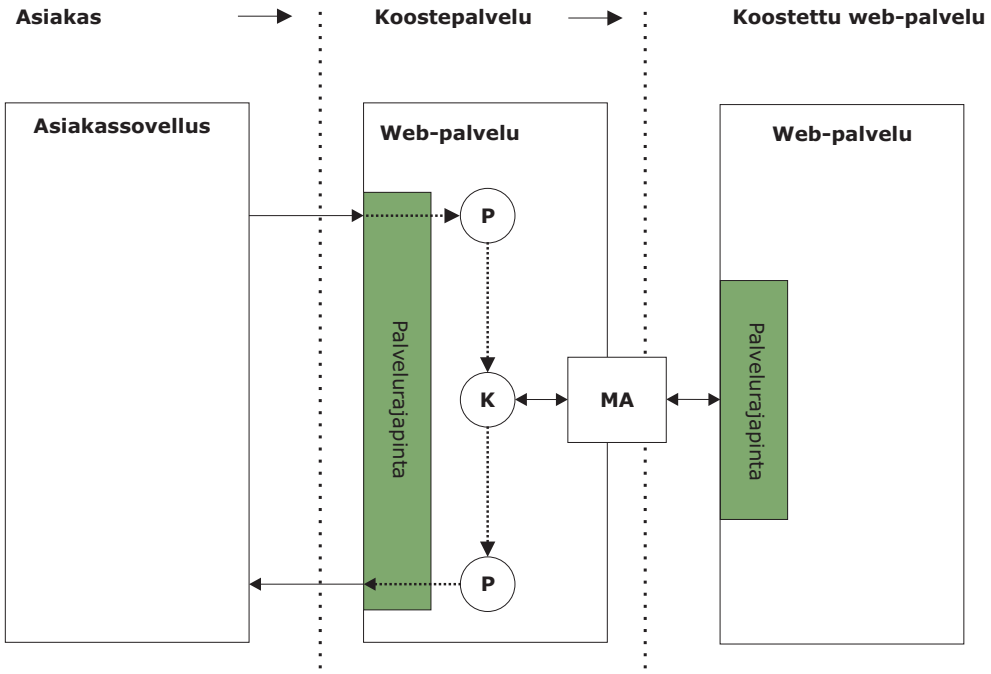
Bussler on johtanut edellisten havaintojen perusteella seuraavat huomiot:

- Mikäli koostetun palvelun viestiskeema muuttuu, se vaikuttaa myös koostepalveluun. Tällöin koostepalvelun viestimuunnos täytyy muokata vastaamaan muuttunutta viestiskeemaa.
- Jokaisen uuden web-palvelun lisääminen koostepalveluun vaatii koostepalvelun päivittämistä. Web-palvelun kutsumetodien lisäksi koostepalvelun täytyy huomioida myös lisättävän web-palvelun viestiskeeman aiheuttamat muutostarpeet.

Edellisten huomioiden pohjalta Bussler esittää vaihtoehtoista lähestymistapaa, missä viestimuunnos perustuu rajapinnan abstrahointiin. Lähestymistapa perustuu ajatukseen, missä heterogeenisuuden käsittely erotetaan koostepalvelusta omaksi kokonaisuudekseen. Lähestymistavassa koostepalvelun ja muunnoksen logiikka erotetaan toisistaan esittelemällä sopivat abstraktiot. Täten lähestymistapaa kutsutaan *muunnosabstraktioksi* (transformation abstraktion). Muunnosabstraktiota käyttävässä koostepalvelussa koostepalvelu ei kiinnitä huomiota viestimuunnoksiin, jolloin koostettujen palvelujen oletetaan olevan homogeenisia koostepalveluun nähden. Muunnos toteutetaan erillisenä ja eristettynä toimenpiteenä koostepalvelulogiikan ulkopuolella. Muunnosabstraktion perusajatuksena on siepata web-palvelukutsu, jolloin sieppaus itsessään suorittaa muunnoksen ennen varsinaista web-palvelukutsua (kuva 34).

Muunnosabstraktiossa muunnos tehdään web-palvelumetodin kutsun yhteydessä, missä muunnos kohdistuu suoraan syöte- ja tulosteparametreihin. Tällöin koostepalvelusta tehtävä web-palvelukutsu ei kutsu suoraan koostettavaa web-palvelua. Sen sijaan kutsu kohdistuu muunnosabstraktion komponenttiin, joka muuntaa viestin kohdepalvelun ymmärtämään muotoon ja lähettää viestin kohdepalvelulle. Täten ainoastaan abstraktiokomponentin täytyy olla tietoinen koostepalvelun ja kohdepalvelun ontologioista. Vastaavasti myös koostetusta palvelusta lähetetty vastausviesti muunnetaan abstraktiokomponentissa, joka palauttaa viestin koostepalvelulle sen ymmärtämässä muodossa. Bussler esittää muunnosabstraktion eduiksi seuraavat kohdat:

1. Koostepalvelun ei tarvitse toteuttaa ylimääräisiä muunnosvaiheita.
2. Koostepalvelu ainoastaan välittää viestit web-palvelun kutsuvaiheelle oman ontologiansa mukaisesti. Sen ei tarvitse käsitellä koostettavan web-palvelun rakenteita ja niiden merkityksiä.



Kuva 34: Viestimuunnos muunnosbastraktiolla, missä P = web-palvelun päätepiste, K = web-palvelukutsu ja MA = muunnosabstraktio.

3. Olemassaolevaa toiminnallisuutta toteuttava uusi web-palvelu linkitetään muunnosabstrakiokomponentin läpi, jolloin koostepalvelun ei tarvitse välittää koostettavan web-palvelun toimintalogiikasta.

4 WEB-API -välityspalvelurajapinta

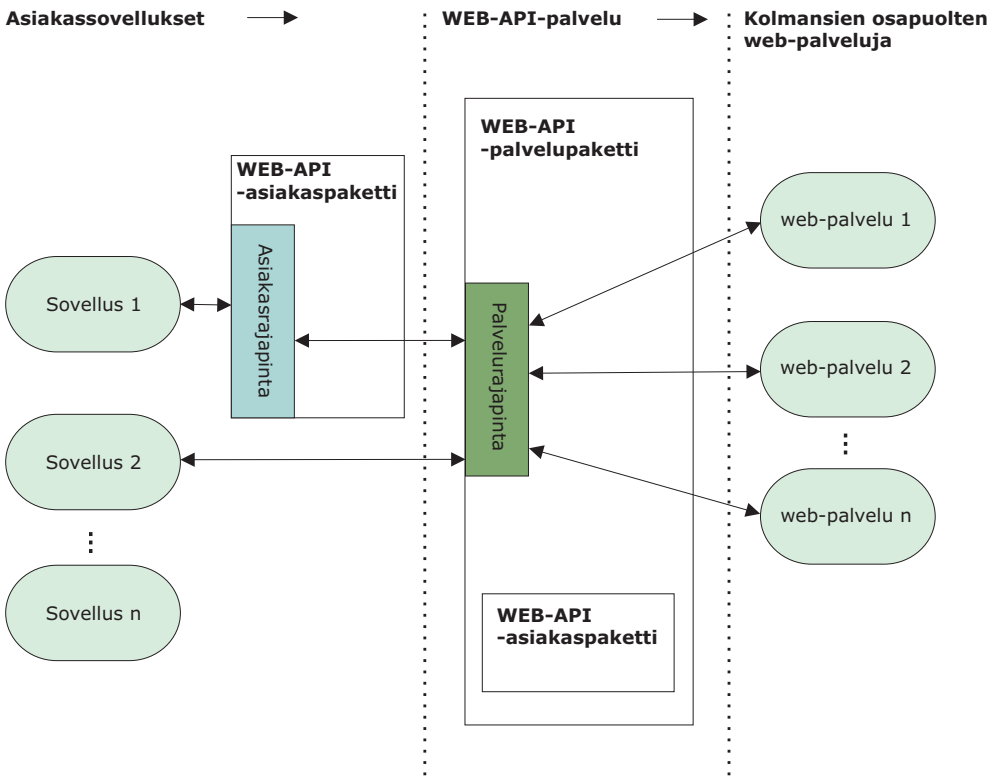
Tässä luvussa esitellään WEB-API-koostepalvelu (Kinnunen, 2008), jossa koostaminen tehdään leveyssuuntaisesti. Tällöin voimme puhua WEB-API:n kohdalla palvelujen koostamisesta. WEB-API tarjoaa yhtenäisen web-palvelurajapinnan ulkoisten web-palvelujen kutsumiseksi. WEB-API 1.0 sisältää tuen Google Picasa -web-palvelurajapinnalle, mutta tukea voidaan laajentaa myöhemmin myös muille web-palveluille. Picasa-palvelun lisäksi WEB-API tukee Googlen tunnistuspalvelua, jota voidaan hyödyntää Picasan lisäksi myös muissa Googlen tarjoamissa web-palvelurajapinnoissa.

Ensiksi kuvataan WEB-API-palvelun toimintaa ja rakennetta yleisellä tasolla. Tämän jälkeen esitellään WEB-API:n tarjoamat palvelut ja niiden käyttämiseksi tarjotut web-palvelurajapintaratkaisut. Palveluista esitellään ensin Googlen tunnistuspalvelu ja siihen liittyvä WEB-API:n rajapinta. Tunnistuspalvelun jälkeen esitellään Googlen Picasa-palvelu ja siihen liittyvä WEB-API:n rajapinta.

4.1 Palvelun rakenne

Tässä kuvataan WEB-API-palvelun rakenne käsitteellisellä tasolla, joka konkretisoidaan toiminnan kannalta olennaisilla luokilla. Palvelun rakenne on jaettu *palvelu-* ja *asiakaspaketeiksi* (kuva 35). Asiakaspaketin ensisijaisena tarkoituksena on helpottaa WEB-API-palvelumetodien kutsua Java-web-sovelluksissa, eikä sen käyttö ole täten pakollista. Asiakaspaketti sisältää kaikki asiakassovelluksen tarvitsemat luokat, kun taas palvelupaketti sisältää WEB-API-palvelun kaikki luokat mukaan lukien myös asiakasluokat. Molemmat paketit sisältävät yhteisesti jaetut luokkapaketit. Lisäksi molemmat paketit sisältävät myös web-palvelujen WSDL-kuvaukset sekä niihin liittyvät skeemat ja kääreluokat, joita tarvitaan SOAP-palveluviestien välityksessä web-palvelun ja asiakaspaketin välillä.

Asiakaspaketti muodostuu fyysisesti jar-paketista, joka sisältää kaikki asiakassovelluksen tarvitsemat luokat. Paketin sisältö koostuu ensisijaisesti web-palvelujen asiakasluokista, joilla voidaan suorittaa WEB-API-palvelun tarjoamia palvelukutsuja. Web-palvelun asiakasluokat muodostavat näin ollen WEB-API:n *Java-asiakasrajapinta API:n,* jota kutsutaan jatkossa WEB-API -palvelun *asiakasrajapinnaksi*. WEB-API 1.0



Kuva 35: WEB-API-palvelun suhteet asiakassovelluksiin ja kolmansien osapuolten koostettuihin web-palveluihin. Ylhäällä olevat nuolet kuvaavat etäpyyntöjen kutsusuunnan.

sisältää asiakasluokat Googlen tunnistus- ja Picasa-palvelujen käyttämiseksi. Asiakasrajapinnassa käytetään WEB-API:n omia luokkakirjastoja, jolloin asiakassovellus ei tarvitse Google Data API:a tai muita kolmansien osapuolten luokkakirjastoja.

WEB-API:a voidaan kutsua myös ilman asiakaspakettia hyödyntäen palvelun WSDL-rajapintaa. Tällöin lähetetyt SOAP-viestit täytyy olla WEB-API:n skeemojen mukaisia. Myös asiakaspaketin lähettämät viestit noudattavat palvelun skeemamäärityksiä. Esimerkki asiakaspaketin lähettämästä SOAP-viestistä on nähtävissä kuvassa 36.

Palvelupaketti muodostuu fyysisesti jar-paketista, WSDL-dokumenteista ja niihin liittyvistä skeemoista, jotka muodostavat yhdessä WEB-API-web-palvelun. Palvelupaketti sisältää WEB-API:n *palvelurajapinnan*, johon voidaan lähettää etäpyyntöjä SOAP-muotoisten XML-viestien avulla. Palvelurajapinta sisältää rajapinnat Googlen tunnistus- ja Picasa-palvelujen käyttämiseksi. Näiden lisäksi palvelupaketti sisältää asiakaspaketin kokonaisuudessaan.

Palvelupaketti muodostaa WEB-API-web-palvelun, kun asiakaspaketti toimii lähinnä

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/
   soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:getRandomPublicPhoto xmlns:ns2="http://
         picasa.google.service.webapi.mkinnu.com">
      <arg0>
        <imagesize>SIZE_288</imagesize>
        <query>
          <alt>ALT_ATOM</alt>
          <fullTextQuery>
            <queryStr>nature</queryStr>
          </fullTextQuery>
          <maxResults>100</maxResults>
        </query>
        <thumbsize>SIZE_64</thumbsize>
      </arg0>
    </ns2:getRandomPublicPhoto>
  </S:Body>
</S:Envelope>
```

Kuva 36: Esimerkki asiakaspaketin WEB-API-palveluun lähettämästä SOAP-pyynnöstä, jolla pyydetään satunnaista julkista kuvaa Googlen Picasa-palvelusta.

käyttöä helpottavana tukipakettina. Kaikki WEB-API-palvelulle lähetetyt etäpyynnöt välitetään palvelupaketin palvelurajapinnalle, joka käsittelee ja välittää pyynnöt tarvittaessa eteenpäin. Myös kaikki asiakaspaketista tulevat pyynnöt ohjautuvat palvelurajapinnalle. Palvelurajapinta lähettää vastaukset skeemojen mukaisessa muodossa. Esimerkki asiakaspaketille lähetetystä vastauksesta on nähtävissä kuvassa 37.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/
   soap/envelope/">
  <S:Body>
    <ns2:getRandomPublicPhotoResponse
         xmlns:ns2="http://picasa.google.service.
         webapi.mkinnu.com">
      <return>
        <serviceResponse>RESPONSE_SUCCESS
        </serviceResponse>
        <photoEntry>
         ...
        </photoEntry>
      </return>
    </ns2:getRandomPublicPhotoResponse>
  </S:Body>
</S:Envelope>
```

Kuva 37: Esimerkki asiakaspaketille lähetetystä SOAP-vastauksesta, joka sisältää Picasa-palvelusta saatuja kuvatietoja (muokattu).

4.2 WEB-API:n palvelut

WEB-API toimii kokoavana välityspalvelurajapintana muille kolmansien osapuolten web-palveluille. Välitysrajapinnalla tarkoitetaan kaikkia WEB-API-palvelun sisältämiä palvelurajapintoja. WEB-API toimii siten, että sille lähetetyt etäpyynnöt lähetetään edelleen kolmansien osapuolten web-palveluille. Näiden palauttamat vastaukset kulkevat WEB-API:n kautta edelleen asiakkaalle. WEB-API 1.0 ei sisällä lainkaan omia palveluja, vaan se välittää kaikki etäpyynnöt edelleen muille palveluille. Asiakkaan lähettämät etäkutsut muunnetaan WEB-API:ssa kunkin palvelun ymmärtämään muotoon. Viestivälityksen toteutuksessa on noudatettu avoimen muunnoksen lähestymistapaa. WEB-API:n ja asiakkaan välillä välitettävät tiedot kääritään WEB-API:n omien skeemojen mukaisiksi SOAP-viesteiksi, jolloin asiakassovelluksen ei tarvitse olla tietoinen koostettujen web-palvelujen toteutusteknisistä piirteistä. Kaikki asiakkaan ja WEB-API -palvelun välinen liikenne tapahtuu WEB-API-palvelurajapinnan

sisältämien XML-viestiskeemojen mukaisesti.

WEB-API:n sisältämät palvelurajapinnat koostuvat *rajapintaluokista* (interface classes), rajapintaluokkien metodikutsut toteuttavista *konkreettisista luokista* (concrete classes) ja web-palvelun kuvaavasta WSDL-dokumentista. Näiden lisäksi palvelut sisältävät tietojen välityksestä vastaavat tietoluokat ja erilaisia apuluokkia. Web-palvelujen rajapintaluokissa käytetään *Java-annotaatioita* (Java-annotations), joiden avulla kuvataan web-palvelu ja palvelun metodit (kuva 38). Näiden avulla web-palvelujen WSDL- ja skeemadokumentit voidaan generoida automaattisesti. Lisäksi rajapintaluokissa määritetään rajapinnan toteuttavan luokan luokkapolku, jota käytetään myös web-palvelun nimiavaruusmäärittelyissä.

WEB-API:n asiakasrajapinnat muodostavat etäkutsuja vastaaviin palvelurajapintaluokkiin. Asiakasrajapintaluokkien metodikutsut noudattavat vastaavien palvelujen rajapintaluokkien kutsumuotoja. Asiakasluokat eivät kuitenkaan toteuta näitä rajapintoja. Sen sijaan niissä muodostetaan palvelurajapintaluokkien mukaisia etäkutsuja, jotka lähetetään vastaaville web-palveluille.

Palvelurajapintaluokkien metodikutsut muodostavat WEB-API-web-palvelun ytimen. Metodikutsujen syöte- ja tulosteparametrit välitetään SOAP-viesteinä. Jokaisen kutsumetodin syöte- ja tuloste-olioita vastaa skeeman mukainen SOAP-viesti. Täten web-palvelun ja asiakkaan välisiä syöte- ja tuloste- Java-olioita kutsutaan jatkossa *viestioliioiksi*. Java-viestioliot voidaan jäsentää SOAP-viesteiksi skeemadokumenttien avulla. Samoin kaikki SOAP-viestit voidaan jäsentää Java-viestiolioiksi. Tämä tapahtuu automaattisesti web-palvelun taustajärjestelmien toimesta. WEB-API:n metodikutsut käsittelevät näin ollen aina Java-olioita, vaikka olioiden data välitetäänkin asiakkaan ja palvelun välillä SOAP-viesteinä. Esimerkki XSD-skeemadokumentin sisältämästä XML-skeematyypistä on nähtävissä kuvassa 39.

WEB-API:n ja koostettujen web-palvelujen välinen viestiliikenne tapahtuu kunkin palvelun mahdollistamalla tavalla. WEB-API:n käyttämät sisäiset toteutustekniikat voivat vaihdella eri web-palvelujen välillä, riippuen käytettävän web-palvelun rajapinnasta. WEB-API 1.0 käyttää Googlen palveluihin tehtäviin etäkutsuihin Google Data API:a. Tämä API-rajapinta sisältää tarvittavat Java-luokkakirjastot, jotka osaavat muuntaa Java-oliot Google-palvelujen ymmärtämiksi viesteiksi ja vastaanotetut viestit Java-olioiksi. WEB-API muuntaa käyttämänsä Google Data API:n Java-tietoluokat omiksi luokkatyypeikseen, joiden attribuutit välitetään asiakkaalle. Täten jokaisessa

```
@WebService(
  name="GoogleAuth",
  targetNamespace="http://auth.google.service.
  webapi.mkinnu.com" )
public interface IGoogleAuthService {

  @WebMethod(action="initRemoteAuth")
  public GoogleAuthInitTokens initRemoteAuth(
         GoogleAuthInitParameters remoteParams );

  @WebMethod(action="requestPersistentAuth")
  public GoogleAuthTokens
                 requestPersistentRemoteAuth(
                 GoogleAuthTokens authToken );

  @WebMethod(action="revokePersistentAuth")
  public GoogleAuthTokens
                   revokePersistentRemoteAuth(
                   GoogleAuthTokens authToken );
}
```

Kuva 38: Google-tunnistuspalvelun määrittelevä rajapintaluokka. WebService-annotaatiolla kuvataan itse palvelu, WebMethod-annotaatiolla puolestaan palvelun julkiset metodikutsut.

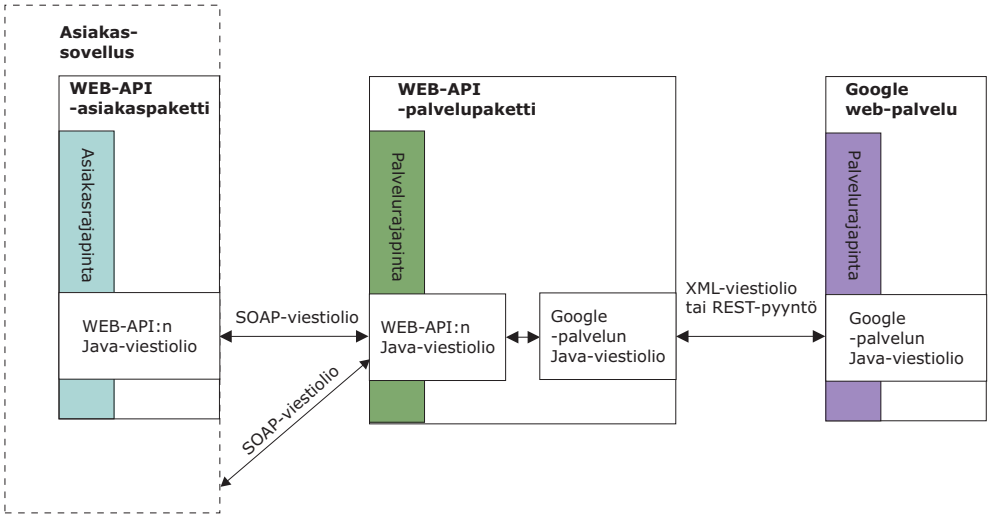
metodikutsussa WEB-API:n syöteviestiolio puretaan ensin Google Data API:n ymmärtämään muotoon, ennen kuin suoritetaan etäkutsu Googlen palveluun. Vastaavasti jokainen Googlen palvelusta saatava tulosteolio muunnetaan WEB-API:n luokkatyypin mukaiseksi viestiolioksi. Tieto-olioiden välitys asiakkaan, WEB-API:n ja Google-palvelujen välillä on nähtävissä kuvassa 40.

WEB-API:n viestiolioluokat on toteutettu siten, että niistä saadaan generoitua WSDL:n avulla XML-viestit automaattisesti. Tämä tarkoittaa sitä, että luokissa käytetään ainoastaan JAX-WS 2.0 -määritysten tukemia tietotyyppejä ja rakenteita. Primitiivi-

tyyppien lisäksi käytetään taulukoita ja alistettuja tieto-olioita, jotka voidaan muuntaa SOAP-muotoisiksi XML-viesteiksi. Jokaista *yksityistä* (private) attribuuttia vastaa *julkiset* (public) get- ja set-metodit, joilla attribuutteja voidaan käyttää luokan ulkopuolelta. Lisäksi jokainen viestiolioluokka sisältää parametrittoman *konstruktorin* (constructor), jota tarvitaan muunnettaessa XML-viestioliot Java-olioiksi. Kaikki WEB-API:n viestioliot perivät com.mkinnu.webapi.common-paketista löytyvän ServiceMetaData-luokan. ServiceMetaData sisältää metodikutsuihin liittyviä metatietoja. Näitä ovat metodikutsun status ja statusviesti. Statuksella viestitetään kutsun onnistuminen. Statusviesti on puolestaan statukseen liittyvä tarkentava viesti, jolla ilmaistaan virhetilanteen syy. Statusviestiä ei välitetä onnistuneissa kutsuissa. Palvelumetodin kutsu voi sisältää seuraavat vakioidut statuskoodit:

```
<xs:complexType name="googleAuthTokens">
    <xs:complexContent>
      <xs:extension base="tns:googleAuthInitTokens">
        <xs:sequence>
          <xs:element name="authToken"
                      type="xs:string"
                      minOccurs="0"/>
          <xs:element name="privateKey"
                      type="tns:googlePrivateKey"
                      minOccurs="0"/>
          <xs:element name="userNameToken"
                      type="xs:string"
                      minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

Kuva 39: Esimerkki XML-skeematyypistä. Esimerkissä GoogleAuthTokens-luokan skeematyyppi, jonka pohjalta luodaan GoogleAuthTokens-olioita vastaavat SOAP-muotoiset XML-viestit.



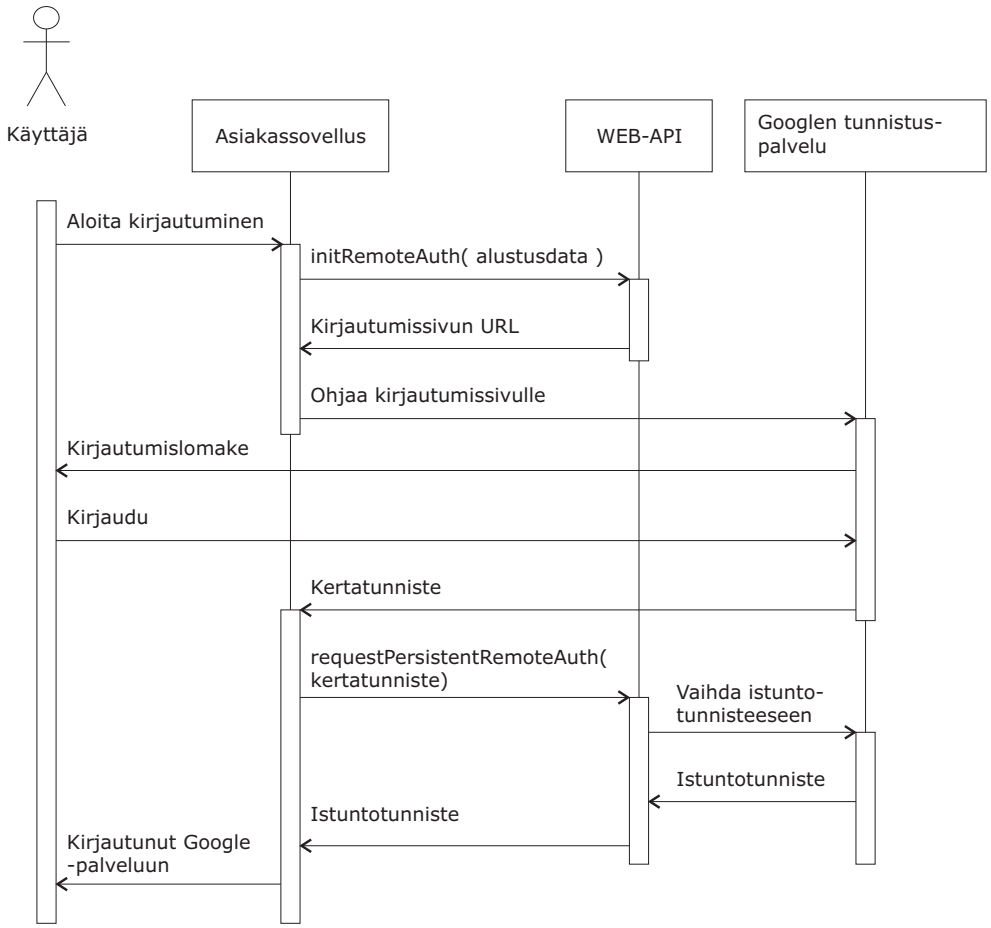
Kuva 40: WEB-API-palvelun välittämät viestit asiakassovelluksen ja Google-palvelujen välillä.

- `RESPONSE_SUCCESS(200)`. Koodilla ilmaistaan metodikutsun onnistuminen.
- `RESPONSE_AUTH_FAILED(403)`. Koodilla ilmaistaan tunnistautumisen epäonnistuminen.
- `RESPONSE_FAILURE(501)`. Koodilla ilmaistaan metodikutsun keskeytyminen virheeseen.

4.3 Googlen tunnistuspalvelu

Googlen tunnistuspalvelu (Google Account Authentication API, 2008) toimii tukipalveluna tunnistusta vaativille Google-palveluille. Palvelua voidaan hyödyntää tilanteissa, joissa tunnistus halutaan suorittaa suoraan Googlen palvelussa. Tässä asiakassovelluksen käyttäjä suorittaa palveluun sisäänkirjautumisen *etätunnistautumisella* Googlen kirjautumispalvelun kautta (kuva 41). Tällöin myöskään käyttäjän Google-salasana ei tule asiakassovelluksen ja WEB-API-palvelun tietoon. Käytännössä kirjautuminen tapahtuu siten, että käyttäjä ohjataan Googlen tunnistuspalvelun kirjautumissivulle. Mikäli asiakassovellusta ei ole rekisteröity Googlen tunnistuspalveluun, näytetään käyttäjälle kirjautumissivun jälkeen varmistussivu ennen kirjautumisen lopullista hyväksymistä. Rekisteröityneille sovelluksille varmistussivua ei näytetä. Rekisteröityjen sovellusten tulee myös käyttää SSL-varmennettua yhteyttä. Onnistuneen kirjautumisen jäl-

keen tunnistuspalvelu palauttaa kertakäyttöisen tunnistemerkkijonon. Tämän jälkeen tunniste voidaan vaihtaa pysyvään istunnon kestävään *istuntotunnisteeseen*. Mikäli tunnistetta ei vaihdeta istuntotunnisteeseen, tunniste täytyy uusia jokaisen Google-palveluihin tehdyn etäkutsun jälkeen. Vastaavasti istuntotunniste voidaan myös lakkauttaa, jolloin vaaditaan uusi kirjautuminen Google-palveluihin. Istuntotunniste toimii kertakirjautumisena pyydettyyn palveluun.



Kuva 41: Käyttäjän kirjautumisprosessi sekvenssikaaviona käytettäessä istunnon kestoista tunnistusta.

Etätunnistus on jaettu kaikkien Google-palvelujen kesken, jolloin kaikkiin palveluihin tunnistaudutaan saman palvelun kautta. Etätunnistautumisen *vaikutusalue* (scope) määritellään alustusparametrilla kirjautumispalveluun ohjauksen yhteydessä.

Google-palvelut mahdollistavat myös kutsukohtaisen tunnistautumisen, jolloin kirjautuminen voidaan suorittaa suoraan asiakassovelluksessa. Tätä kutsutaan WEB-API:n yhteydessä *lähitunnistautumiseksi*. Tässä käyttäjän Google-käyttäjätunnus ja -salasana välitetään jokaisen tunnistusta vaativan etäkutsun mukana. Tällöin asiakassovellus vas-

taa istunnon kestosta kuin myös käyttäjän Google-tunnuksen ja -salasanan säilönnästä.

4.4 Google-tunnistusrajapinta

WEB-API:n *Google-tunnistusrajapinta* tarjoaa välitysrajapinnan Googlen tunnistuspalvelua varten. Välitysrajapinnan web-palvelu määräytyy com.mkinnu.webapi.common.google.auth-paketissa sijaitsevan IGoogleAuthService-rajapintaluokan mukaisesti, joka toimii web-palvelun *päätepisterajapintana* (endpoint interface). Rajapinta sisältää tarvittavat kutsumetodimäärittelyt Googlen tunnistuspalvelun käyttämiseksi. Rajapinta toteutetaan com.mkinnu.webapi.service.google.auth-paketin GoogleAuth-luokassa, joka toimii web-palvelun toteuttavana luokkana. IGoogleAuthService-rajapintaluokka sisältää seuraavat kutsumetodit:

- `public GoogleAuthInitTokens initRemoteAuth( GoogleAuthInitParameters remoteParams )`
- `public GoogleAuthTokens requestPersistentRemoteAuth( GoogleAuthTokens authToken )`
- `public GoogleAuthTokens revokePersistentRemoteAuth( GoogleAuthTokens authToken )`

InitRemoteAuth-metodia käytetään etätunnistuspyynnön alustukseen. Tätä tulee kutsua ensimmäisenä, jotta asiakassovellus pystyy välittämään Googlen tunnistuspalveluun oikeat alustustiedot. Tämä sisältää myös itse tunnistuspalvelun URL-osoitteen. Metodi saa syöteparametriksi GoogleAuthInitParameters-luokan ilmentymäolion. Syöteparametriolio sisältää pakollisina attribuutteina paluu- ja vaikutusalue-URL-osoitteet. Käyttäjä ohjataan paluusoitteeseen, mikäli Googlessa suoritettu tunnistus onnistui. Vaikutusalueosoitteella ilmaistaan, mihin palveluun käyttäjä tulee tunnistaa. Syöteparametriolion avulla voidaan myös pyytää varmistettua yhteyttä ja istunnon kestoista tunnistusta. Mikäli näihin attribuutteihin ei aseteta arvoa tosi, jätetään kyseiset ominaisuudet oletusarvoisesti pois käytöstä. WEB-API 1.0 ei sisällä tukea varmistetuille yhteyksille, joten varmistetun yhteyden pyytämisellä ei ole vaikutusta metodin toimintaan. Metodi hyödyntää alustuksessa Google Data API:n AuthSubUtil-luokan *getRequestUrl*-metodia, joka palauttaa alustetun URL-osoitteen parametreineen. Tämän jälkeen tulosteet kapseloidaan GoogleAuthInitTokens-luokan olioon, joka sisältää

metodikutsun statuksen, Google-tunnistuspalvelun URL-osoitteen alustettuna oikeilla parametreilla sekä muita alustukseen liittyviä metatietoja.

RequestPersistentRemoteAuth-metodilla voidaan pyytää istunnon kestoista tunnistusta. Metodia voidaan kutsua, kun Googlen palvelusta on saatu kertakäyttöinen tunnistemerkkijono. Metodi saa syöteparametrina GoogleAuthTokens-luokan ilmentymäolion. Syöteparametriolio sisältää pakollisena attribuuttina kertakäyttöisen tunnistemerkkijonon. Metodi kutsuu Google Data API:n AuthSubUtil-luokan *exchangeForSessionToken*-metodia, joka tekee etäkutsun Googlen tunnistuspalveluun. Tämä palauttaa istuntokestoisen tunnistemerkkijonon, joka kapseloidaan GoogleAuthTokens-luokan tulosteolioksi. Mikäli tunnistemerkkijonon vaihto epäonnistuu virheelliseen kertatunnisteeseen, tulosteolion sisältämän tunnisteen arvoksi tulee null ja statuskoodiksi RESPONSE_AUTH_FAILED.

RevokePersistentRemoteAuth-metodi lakkauttaa istuntokestoisen tunnisteen voimassaolon. Metodia kutsutaan, kun halutaan lakkauttaa voimassaoleva istunnon kestoinen tunnistautuminen. Metodi saa syöteparametriksi GoogleAuthTokens-luokan ilmentymäolion. Tämä sisältää pakollisena attribuuttina voimassaolevan istuntotunnistemerkkijonon. Metodi kutsuu Google Data API:n AuthSubUtil-luokan void-tyyppistä *revokeToken*-metodia, joka tekee etäkutsun Googlen tunnistuspalveluun. RevokePersistentRemoteAuth-metodi muodostaa paluuolioksi tyhjän GoogleAuthTokens-luokan tulosteolion. Metodikutsun onnistuminen ilmaistaan tulosteolion statuskoodilla.

4.5 Picasa-web-palvelu

Googlen Picasa-sovellus mahdollistaa valokuvien hallinnoinnin omalla tietokoneella. Tämän lisäksi Picasa sisältää Internet-palvelun, jonne käyttäjät voivat varastoida omia valokuviaan. Tässä yhteydessä käsitellään Picasa Web Albums -Internet-palvelua ja siihen liittyvää web-palvelua, joilla voidaan hallinnoida valokuvia joko omien yksityisten tai julkisten valokuvakansioiden avulla.

WEB-API hyödyntää omassa Picasa-palvelussaan Picasa Web Albums Data API -web-palvelurajapintaa. Picasan web-palvelurajapinta mahdollistaa kuvien haut ja lataukset Picasa-palvelun ja asiakassovelluksen välillä. Lisäksi web-palvelun avulla voidaan hakea ja muokata valokuvien metatietoja. Valokuviin voidaan myös liittää sijaintitietoja,

joissa hyödynnetään Google Maps -palvelua. Kuvia voidaan hakea sekä omista yksityisistä kansioista että yhteisön julkisista kansioista, jolloin kuvia voidaan hakea myös muiden Picasa-käyttäjien julkisista kansioista. Yksityisiin kansioihin kohdistuvat toimenpiteet vaativat aina tunnistautumisen. Myös julkisiin kansioihin ja kuviin tehtävät muokkaustoimet vaativat tunnistautumisen. Sen sijaan julkisista kansioista ja kuvista voidaan tehdä hakuja ilman tunnistusta.

Picasa-web-palvelu lähettää ja vastaanottaa viestit XML- tai REST-muotoisina. Asiakassovelluksen tekemillä REST-tyyppisillä etäpyynnöillä voidaan määrittää esimerkiksi hakuparametreja tai päivitettäviä tietoja. Vastausviestit sisältävät kuvien URL-linkkejä ja kuviin liittyviä metatietoja. Vastausviestit eivät sisällä varsinaista kuvadataa, mutta asiakassovellukset voivat linkittää Picasan pavelimella olevia kuvia. Vaihtoehtoisesti kuvat voidaan myös ladata tai lähettää *tietovirtoina* (datastream), jotka sisältävät kuvatiedoston sisältämän datan.

Picasa-web-palvelussa kuvahaut perustuvat syötteiden käyttöön. Syöte-URL-osoitteisiin tehdyt REST-pyynnöt palauttavat vastaavat XML-tulosviestit, jotka sisältävät pyynnöllä haetut tiedot. Syöte-URL-osoitteilla määritetään miten ja millaista tietoa halutaan hakea. Syöte-URL rakentuu eri osista, jotka tarkentavat syötteen määrityksiä. Picasa-syötteen perusosa muodostuu osoitteesta: ”http://picasaweb.google.com/data/feed/”. Tätä tarkennetaan *kohdistus*-tyypillä (projection), vakioidulla *laji*-tyypillä (kind) ja syötetyyppien mukaisilla tarkentavilla kohdistustiedoilla. Tällöin syöte-URL noudattaa muotoa: ”http://picasaweb.google.com/data/feed/*kohdistus*/*syötetyyppien tarkentavat kohdistustiedot*?*lajityyppi-parametri*&*muita parametreja*”.

Vakioituja kohdistustyyppejä ovat base- ja api-tyypit. Base-kohdistuksella vastausviesteihin ei sisällytetä päivitysoperaatioissa vaadittuja tietoja, jolloin Picasa Web Albums -palvelussa olevia tietoja ei voida päivittää. Base-kohdistus sopii silloin, kun palvelussa sijaitsevia tietoja ei tarvitse päivittää ja vastausviestien ei tarvitse sisältää kuvista yksityiskohtaisia tietoja. Api-kohdistuksella vastausviesteihin sisällytetään Picasan gphoto-nimiavaruuden laajennuselementtien sisältämät tiedot. Laajennuselementit sisältävät kuvien ja tietojen päivityksessä tarvittavia tietoja. Esimerkkinä kuvien tunnisteet, joilla yksittäinen kuva voidaan yksilöidä. Lajityyppiparametrilla ilmaistaan, millaista tietoa syötteellä halutaan hakea. Vakioituja lajeja ovat albumi-, *merkintä* (tag)-, valokuva- ja kommenttilajit. Haluttu laji ilmaistaan syötteessä lajityyppiparametrilla. Syötteille määriteltävä syötetyyppi täydentää syötteen URL-osoitteen kohdistustiedot.

Tällöin jokaiselle syötetyypille muodostuu yksilöllinen URL-osoite (kuva 42). Picasa sisältää neljä syötetyyppiä:

- *Käyttäjäpohjainen* (user-based) syöte. Syötetyyppi sisältää tiettyyn käyttäjään sidoksissa olevia tietoja. Käyttäjäpohjaisella syötteellä voidaan hakea valitun käyttäjän albumi-, merkintä-, valokuva- ja kommenttilajisia tietoja. Käyttäjään viitataan syöteosoitteen käyttäjätunnuksella, joka on käyttäjän Google-käyttäjätunnus. Mikäli käyttäjä on tunnistautunut Googlen tunnistuspalvelun kautta, käyttäjätunnuksen sijaan käytetään vakioitua ”default”-arvoa, jolloin Picasa selvittää todellisen käyttäjätunnuksen tunnistemerkkijonon perusteella.
- *Albumipohjainen* (album-based) syöte. Syötetyyppi sisältää tiettyyn käyttäjään ja valokuva-albumiin sidoksissa olevia tietoja. Albumipohjaisella syötteellä voidaan hakea valitun albumin valokuva- ja merkintälajisia tietoja. Käyttäjän albumiin viitataan käyttäjätunnuksella ja albumin nimellä tai albumin tunnuksella. Albumi voidaan siis hakea sekä yksilöivällä tunnuksella että albumin nimellä.
- *Valokuvapohjainen* (photo-based) syöte. Syötetyyppi sisältää tiettyyn valokuvaan sidoksissa olevia tietoja. Valokuvapohjaisella syötteellä voidaan hakea valitun valokuvan merkintä- ja kommenttilajisia tietoja. Valokuvaan viitataan valokuvan tunnuksella. Lisäksi tarvitaan käyttäjän Google-käyttäjätunnus ja albumin tunnus tai nimi.
- *Yhteisöpohjainen* (community search) syöte. Syötetyypillä etsitään julkisia valokuvia ja niihin liittyviä metatietoja. Toiminnallisesti tämä vastaa albumipohjaista syötettä. Yhteisöpohjaisessa syötteessä pitää myös määrittää hakusanaparametri ”q”.

Syötteille voidaan antaa edellä mainittujen laji- ja hakusana-parametrien lisäksi myös muita parametreja, joilla voidaan säädellä Picasan palauttamia tietoja. Näitä ovat esimerkiksi kuvan kokoon, näkyvyyteen ja tulosten maksimilukumäärään liittyvät parametrit. Syöte-URL-parametrien järjestyksellä ja lukumäärällä ei ole merkitystä, joten niitä voidaan käyttää mielivaltaisessa järjestyksessä. Picasa tukee seuraavia syöteparametreja:

- *access*-parametrilla suodatetaan hakutuloksia näkyvyyden mukaan. Mahdollisia arvoja ovat: all, private ja public. Arvolla ”all” haetaan sekä yksityisiä että jul-

Käyttäjäpohjainen syöte:

http://picasaweb.google.com/data/feed/*projection*/user/*userID*/?kind=kinds

Albumipohjainen syöte:

Viittaus albumin tunnuksella:

http://picasaweb.google.com/data/feed/*projection*/
user/*userID*/albumid/*albumID*?kind=kinds

Viittaus albumin nimellä:

http://picasaweb.google.com/data/feed/*projection*/
user/*userID*/album/*albumName*?kind=kinds

Valokuvapohjainen syöte:

Viittaus albumin tunnuksella:

http://picasaweb.google.com/data/feed/*projection*/user/userID/
albumid/*albumID*/photoid/*photoID*?kind=kinds

Viittaus albumin nimellä:

http://picasaweb.google.com/data/feed/*projection*/user/userID/album
/*albumName*/photoid/*photoID*?kind=kinds

Yhteisöpohjainen syöte:

http://picasaweb.google.com/data/feed/*projection*/all?kind=photo&q=searchTerm

Kuva 42: Syötetyyppien URL-osoitteet Picasa-palvelussa. Kursiivilla olevat arvot korvataan kohdistus-, laji- ja syötetyyppikohtaisilla yksilöivillä tiedoilla.

kisia tietoja, kun taas ”private”-arvolla pelkästään yksityisiä ja ”public”-arvolla pelkästään julkisia tietoja. Arvot ”all” ja ”private” vaativat tunnistautumisen, jotta arvoja voisi käyttää. Kirjautuneella käyttäjällä käytetään oletusarvoa ”all” ja kirjautumattomalla arvoa ”public”.

- *alt*-parametrilla määritetään Google-palvelun palauttaman XML-dokumentin muoto.

- *imgmax*-parametrilla määritetään valokuva- ja albumilajeilla palautettavien valokuvien vaakaresoluutio. Resoluutiot on vakioitu taulukon 3 mukaisesti. Picasa rajoittaa sivulle upotettavilla kuvilla maksimiresoluutioksi 800×600-tarkkuuden.

- *kind*-parametrilla määritetään hakuun liittyvä lajityyppi.
- *max-results*-parametrilla asetetaan hakutulosten maksimilukumäärä.
- *q*-parametrilla asetetaan hakusanat.
- *start-index*-parametrilla asetetaan ensimmäisen palautettavan tietueen indeksi.
- *tag*-parametrilla voidaan suodattaa hakutuloksia Picasa-käyttäjien määrittelemien merkintöjen avulla.
- *thumbsize*-parametrilla määritetään *miniatyyrikuvien* (thumbnail) vaakaresoluutio. Tähän pätee samat säännöt kuin imgmax-parametrilla.

Taulukko 3: Thumbsize- ja imgmax-parametreissa sallitut resoluutiot, missä resoluutio kuvaa kuvan vaakaresoluutiota. Symbolilla ”d” kuva palautetaan alkuperäisellä resoluutiolla.

Resoluutio	**Rajattu**	**Upotettavissa**	**Resoluutio**	**Rajattu**	**Upotettavissa**
32	Kyllä	Kyllä	*576*	Ei	Kyllä
48	Kyllä	Kyllä	*640*	Ei	Kyllä
64	Kyllä	Kyllä	*720*	Ei	Kyllä
72	Ei	Kyllä	*800*	Ei	Kyllä
144	Ei	Kyllä	*912*	Ei	Ei
160	Kyllä	Kyllä	*1024*	Ei	Ei
200	Ei	Kyllä	*1152*	Ei	Ei
288	Ei	Kyllä	*1280*	Ei	Ei
320	Ei	Kyllä	*1440*	Ei	Ei
400	Ei	Kyllä	*1600*	Ei	Ei
512	Ei	Kyllä	*d*	Ei	Ei

4.6 Picasa-välitysrajapinta

WEB-API 1.0:n *Picasa-välitysrajapinta* tarjoaa välitysrajapinnan Googlen Picasa Web Albums -palvelun hyödyntämiseksi. Välitysrajapinta sisältää suurimman osan Picasan web-palvelun sisältämistä toiminnoista. Välitysrajapinnasta on jätetty pois kaikki kuvatiedostojen lataukseen liittyvä toiminnallisuus, koska välityspalvelun käyttäminen

hidastaisi kuvatiedostojen latauksia turhaan. Lisäksi myös käsiteltävien tietojen määrää on rajattu alkuperäistä pienemmäksi.

Picasa-välitysrajapinnan tarjoama web-palvelu määräytyy com.mkinnu.webapi.common.google.picasa-paketissa sijaitsevan IGooglePicasaService-rajapintaluokan mukaisesti, joka toimii myös web-palvelun päätepisterajapintana. Rajapinta sisältää tarvittavat kutsumetodimäärittelyt Googlen Picasa web-palvelun käyttämiseksi. Rajapinta toteutetaan com.mkinnu.webapi.service.google.picasa-paketin GooglePicasa-luokassa, joka toimii web-palvelun toteuttavana luokkana. IGooglePicasaService-rajapintaluokka sisältää kaikista tunnistusta vaativista kutsumetodeista kaksi versiota. Toinen versio on tarkoitettu käytettäväksi etätunnistuksella hyödyntäen Googlen tunnistuspalvelua ja toinen lähitunnistuksella, missä tunnistustiedot välitetään jokaisen kutsun mukana erikseen. Lähitunnistusta käyttävät metodit eroavat etätunnistusmetodeista ainoastaan nimen ja syöteparametrien osalta. Etätunnistusta käyttävien metodien nimen loppu muodostuu ByAuth-päätteestä, kun taas lähitunnistusta käyttävien metodiversioiden nimen loppu muodostuu ByUser-päätteestä. Yhteisöhakua käyttävistä metodeista on vain yksi versio, koska metodien käyttö ei vaadi tunnistusta Picasa-palveluun.

IGooglePicasaService-rajapintaluokka sisältää palvelumetodit syötetyyppien mukaisten metatietojen, albumien ja valokuvien hakemiseksi. Albumien ylläpitoon rajapinta tarjoaa albumin luonti-, muokkaus- ja poistometodit. Lisäksi myös valokuvia voidaan ylläpitää valokuvan muokkaus- ja poistometodeilla. Valokuvien lisääminen ei ole kuitenkaan mahdollista välitysrajapinnan kautta. Näiden lisäksi Picasan rajapintaa on laajennettu satunnaisten valokuvien haulla, missä haetaan satunnaista valokuvaa tietyltä käyttäjältä tai Picasa-yhteisöstä. Kaikkien metodien suoritus noudattaa pääpiirteittäin seuraavaa muotoa:

1. Muodostetaan vakioitujen tietojen ja syöteparametrien avulla Picasan syötetyypin mukainen syöte-URL-osoite ja siihen liittyvät parametrit.
2. Tehdään etäkutsu Googlen Picasa-web-palveluun hyödyntäen Googlen Data API -kirjaston sisältämiä Picasa-palvelun asiakasluokkia.
3. Luetaan Picasa-palvelun palauttamat tulosteoliot ja muunnetaan ne WEB-API:n omiksi tulosteolioiksi.
4. Palautetaan tulosteoliot asiakkaalle.

Syöteparametrien osalta kaikki tunnistusta vaativat metodit saavat syöteparametriksi joko GoogleAuthToken- tai GoogleUserCredentials-luokan ilmentymäolion. Etätunnistusta vaativat metodit saavat syöteparametrina GoogleAuthToken-luokan ilmentymäolion, missä tunnistemerkkijono on olion ainoa pakollinen attribuutti. Tätä merkkijonoa käytetään Google-palveluihin tunnistauduttaessa. Lähitunnistusta vaativat metodit saavat syöteparametrina GoogleUserCredentials-luokan ilmentymäolion, joka sisältää pakollisina attribuutteina käyttäjätunnuksen ja salasanan. Nämä välitetään asiakassovelluksen ja WEB-API:n 1.0 välillä oletusarvoisesti salaamattomina, joten näiden metodiversioiden käyttöä tulisi välttää suojaamattomissa ympäristöissä.

Metatiedot voidaan hakea käyttäjä-, albumi- ja valokuva-pohjaisilta syötetyypeiltä. Tunnistusparametrin lisäksi jotkut metodit saavat parametriksi *suodatin* (filter) -parametriolion, jolla voidaan suodattaa hakutuloksia. Metatietoja palauttavat metodit voivat palauttaa syötetyyppien mukaisesti GoogleUserFeedMetaData-, GoogleAlbumFeedMetaData- ja GooglePhotoFeedMetaData-luokkien olioilmentymiä. Metatietojen haku onnistuu seuraavilla metodeilla, kun käytetään Googlen tunnistuspalvelua:

- `public GoogleUserFeedMetaData getUserMetaDataByAuth( GoogleAuthTokens authTokens );`
- `public GoogleAlbumFeedMetaData getAlbumMetaDataByAuth( GoogleAuthTokens authTokens, PicasaPhotoFilter filter );`
- `public GooglePhotoFeedMetaData getPhotoMetaDataByAuth( GoogleAuthTokens authTokens, PicasaPhotoFilter filter );`

GetUserMetaDataByAuth-, *getAlbumMetaDataByAuth*- ja *getPhotoMetaDataByAuth*-metodit palauttavat eri syötetyyppeihin liittyviä metatietoja. Metodit käyttävät etäkutsuihin PicasawebService-luokan *getFeed*-metodia. GetUserMetaDataByAuth palauttaa käyttäjäpohjaiseen syötteeseen liittyviä metatietoja. Näitä ovat muun muassa käyttäjän nimi, Google-tilikohtainen kuvake ja Picasa-albumissa olevan vapaan tilan määrä. GetAlbumMetaDataByAuth-metodi palauttaa albumipohjaiseen syötteeseen liittyviä metatitietoja, joita ovat muun muassa albumin näkyvyys ja julkiseen albumiin jätettyjen kommenttien lukumäärä. GetPhotoMetaDataByAuth palauttaa puolestaan va-

lokuvapohjaiseen syötteeseen liittyviä metatietoja, joita ovat muun muassa kuvan koko tavuina, kommenttien lukumäärä ja tekijätiedot. Myös hakumetodit voivat palauttaa vastaavia metatietoja, koska ne perustuvat samoihin syötetyyppeihin. Metatietojen haulla voidaan suodattaa pois hakumetodien palauttamat kuvalinkit, jolloin tulostevies-tit pysyvät pienempinä.

Hakumetodeilla voidaan hakea käyttäjä-, albumi-, valokuva- ja yhteisöpohjaisten syötetyyppien mukaisia tietoja. Hakumetodit saavat mahdollisen tunnistusparametrin lisäksi suodatinparametriolion. Hakumetodien suodatinparametrioliot ovat PicasaPhotoQueryFilter-, PicasaPhotoFilter-, PicasaUserPhotoQueryFilter- ja PicasaAlbumQueryFilter-luokkien ilmentymäolioita. Suodatinolioiden attribuutit ovat yleensä vakioituja *enum*-tyyppejä, joiden mahdolliset arvot ovat ennalta määrättyjä. Nämä noudattavat Picasa-palvelun vakioituja osoite- ja parametriarvoja. Poikkeuksena on PicasaQuery-luokka, jonka sisältämällä PicasaFullTextQuery-oliolla voidaan luoda tekstihakulausekkeita. Suodatinolion muodostaminen on nähtävissä kuvassa 43. Hakumetodit palauttavat GoogleAlbums-, GoogleAlbum-, GooglePhotos- ja GooglePhoto-luokkien ilmentymäolioita. Nämä ovat kääreolioita, jotka sisältävät palautettavien tulosteolioiden lisäksi myös metatietoa metodikutsun suoritukseen onnistumisesta. Välitysrajapinta tarjoaa seuraavat hakumetodit, kun käytetään Googlen tunnistuspalvelua:

- `public GoogleAlbums getUserAlbumsByAuth( GoogleAuthTokens authTokens, PicasaAlbumQueryFilter filter );`
- `public GooglePhotos getUserPhotosByAuth( GoogleAuthTokens authTokens, PicasaUserPhotoQueryFilter filter );`
- `public GooglePhoto getPhotoByAuth( GoogleAuthTokens authTokens, PicasaPhotoFilter filter );`
- `public GooglePhoto getRandomPhotoByAuth( GoogleAuthTokens authTokens, PicasaUserPhotoQueryFilter filter );`
- `public GooglePhotos getPublicPhotos( PicasaPhotoQueryFilter filter );`

- `public GooglePhoto getRandomPublicPhoto( PicasaPhotoQueryFilter filter );`

GetUserAlbumsByAuth-metodi palauttaa tunnistautuneen käyttäjän valokuva-albumit suodatinparametrin mukaisesti suodatettuna. Metodi käyttää albumi-pohjaista syötetyyppiä. Etäkutsu suoritetaan PicasawebService-luokan getFeed-metodilla. Metodin palauttama GoogleAlbums-luokan olio sisältää tulosteet taulukossa GoogleAlbumEntry-luokan olioilmentyminä. GoogleAlbumEntry-luokalta saadaan muun muassa albumin tunnus, nimi ja miniatyyrikuvakkeiden tiedot. Nämä saadaan luettua *getId*-, *getTitle*- ja *getThumbnails*-metodeilla, missä getThumbnails palauttaa GoogleMediaContentBase-luokan olioilmentymän, joka sisältää eri

```
...
PicasaUserPhotoQueryFilter filter =
            new PicasaUserPhotoQueryFilter();
filter.setProjection(
            PicasaProjectionType.PROJECTION_API );
filter.setThumbsize(
            PicasaImagesizeType.SIZE_64 );
filter.setImagesize(
            PicasaImagesizeType.SIZE_1280 );
filter.setAlbumId(
            this.getAlbumId() );
...
PicasaFullTextQuery ftq =
           new PicasaFullTextQuery();
ftq.appendTerm( "hakusana", false, false );
PicasaQuery query = new PicasaQuery();
query.setFullTextQuery( ftq );
filter.setQuery( query );
...
```

Kuva 43: Esimerkki suodattimen luonnista ja konfiguroinnista. Esimerkissä luodaan ensin suodatin, jonka jälkeen siihen lisätään erilaisia suodatinparametreja.

kokoisten miniatyyrikuvakkeiden URL-osoitteet ja kokotiedot.

GetUserPhotosByAuth-, *getPhotoByAuth*- ja *getRandomPhotoByAuth*-metodit palauttavat tunnistautuneen käyttäjän valokuvia suodatinparametrin mukaisesti suodatettuna. GetUserPhotosByAuth- ja getRandomPhotoByAuth-metodeissa käytetään käyttäjäpohjaista syötetyyppiä, kun taas getPhotoByAuth käyttää valokuvapohjaista syötetyyppiä. Metodien etäkutsut suoritetaan PicasawebService-luokan getFeed-metodilla. GetUserPhotosByAuth-metodi palauttaa useita valokuvia, jotka voidaan suodattaa esimerkiksi albumin tunnuksen tai nimen perusteella. Metodin palauttama GooglePhotos-luokan olio sisältää tulosteet taulukossa GooglePhotoEntry-luokan olioilmentyminä. GetPhotoByAuth-metodi palauttaa valitusta albumista valitun kuvan. Metodi vaatii syöteparametreiksi joko albumin tunnuksen tai nimen sekä kuvan tunnuksen. Metodi palauttaa GooglePhoto-luokan olioilmentymän, joka sisältää puolestaan yhden GooglePhotoEntry-luokan olioilmentymän. GetRandomPhotoByAuth-metodi palauttaa syöteparametrin mukaisesti suodatetusta kuvajoukosta satunnaisen valokuvan. Suodatettu kuvajoukko muodostuu samoin kuin GetUserPhotosByAuth-metodilla. GetRandomPhotoByAuth palauttaa GooglePhoto-luokan olioilmentymän. GooglePhotoEntry-luokka sisältää kaikki valokuvaan liittyvät tiedot. Luokalta saadaan muun muassa valokuvan tunnus, nimi sekä valokuvan ja sitä vastaavien miniatyyrikuvakkeiden tiedot. Nämä saadaan luettua getId-, getTitle-, *getMediaContents*- ja getThumbnails-metodeilla, missä valokuvan tiedot palauttava getMediaContents-metodi voi palauttaa usean kokoisia versioita, samaan tapaan kuin miniatyyrikuvienkin kohdalla. GetMediaContents-metodi palauttaa taulukossa 1-n kappaletta GoogleMediaContent-luokan ilmentymäolioita. Palautetut GoogleMediaContent-oliot sisältävät kuvien URL-osoitteet ja kokotiedot. Näillä tiedoilla valokuvat voidaan upottaa HTML-sivulle kuvan 44 mukaisesti.

GetPublicPhotos- ja *getRandomPublicPhoto*-metodeilla saa haettua Picasa-käyttäjien julkisia kuvia. Molemmat metodit käyttävät yhteisöpohjaista syötetyyppiä. Metodit eivät myöskään vaadi tunnistusta. Metodien etäkutsut suoritetaan PicasawebService-luokan getFeed-metodilla. Koska metodien haku ulottuu koko Picasa-yhteisöön, Picasa rajoittaa tulosten lukumäärää oletusarvolla, joka voidaan halutessa yliajaa omalla arvolla. GetPublicPhotos palauttaa valokuvia syöteparametrin mukaisesti suodatettuna. Metodi palauttaa GooglePhotos-luokan olioilmentymän. GetRandomPublicPhoto palauttaa syöteparametrin mukaisesti suodatetusta kuvajoukosta satunnaisen valokuvan. Suodatettu kuvajoukko muodostuu samoin kuin GetPublicPhotos-metodilla.

GetRandomPublicPhoto-metodi palauttaa GooglePhoto-luokan olioilmentymän.

Lisäys-, muokkaus- ja poistometodit saavat tunnistusparametrien lisäksi joko GoogleAlbumEntry- tai GooglePhotoEntry-luokan ilmentymäolion. Muokkaus- ja poisto-operaatioiden suoritus vaatii, että syöteoliolla on tiedossa Googlen web-palvelusta saatu muokkausosoite. Tämän voi tarkastaa erityisellä *isEditable*-metodikutsulla. Muokkauksessa ja poistossa käytettävät luokat perivät tämän metodin GoogleMediaEntry-luokalta, joka on kantaluokkana kaikille WEB-API:n medialuokille. Muokkaus-URL-osoite ei tule automaattisesti kaikkien hakujen mukana. Sen saa

```
<%
...
GooglePhoto photoMsgObj = null;
...
//haetaan valokuvat palvelusta...
...
//haetaan valokuvan entry-olio
GooglePhotoEntry entry =
                photoMsgObj.getPhotoEntry();
//haetaan entry-olion ensimmäinen kuva-alkio...
GoogleMediaContent photo =
                entry.getMediaContents()[0];
//upotetaan kuva HTML-merkintään.
%>
<img src="<%= photo.getUrl() %>"
     width="<%= photo.getWidth() %>"
     height="<%= photo.getHeight() %>"
     alt="<%= photo.getTitle()%>"
     title="<%= photo.getTitle()%>" />
...
```

Kuva 44: Esimerkki kuvan upottamisesta *JSP* (Java Server Pages) -sivulla. Esimerkissä kuva upotetaan HTML:n img-merkinnän sisään, missä kuvan URL viittaa Picasan Web Albums -palvelimella sijaitsevaan kuvatiedostoon.

ainoastaan silloin, kun kohdistustyyppinä käytetään api-tyyppiä. Välitysrajapinta tarjoaa seuraavat metodit lisäykseen, muokkaukseen ja poistoon, kun käytetään Googlen tunnistuspalvelua:

- `public GoogleAlbum addAlbumByAuth( GoogleAuthTokens authTokens, GoogleAlbumEntry album );`
- `public GoogleAlbum updateAlbumByAuth( GoogleAuthTokens authTokens, GoogleAlbumEntry album );`
- `public ServiceMetaData deleteAlbumByAuth( GoogleAuthTokens authTokens, GoogleAlbumEntry album );`
- `public GooglePhoto updatePhotoMetadataByAuth( GoogleAuthTokens authTokens, GooglePhotoEntry photo );`
- `public ServiceMetaData deletePhotoByAuth( GoogleAuthTokens authTokens, GooglePhotoEntry photo );`

AddAlbumByAuth-, *updateAlbumByAuth*- ja *deleteAlbumByAuth*-metodeilla voidaan lisätä, muokata ja poistaa albumeja. AddAlbumByAuth-metodi käyttää käyttäjäpohjaista syöte-URL-osoitetta, kun taas updateAlbumByAuth- ja deleteAlbumByAuth-metodit käyttävät Picasa-web-palvelusta saatua muokkaus-URL-osoitetta. Lisäysmetodi saa parametriksi GoogleAlbum-luokan ilmentymäolion, jolla voidaan antaa luotavalle albumille otsikko, kuvaus, yhteenveto, versiotunnus ja tekijätiedot. Nämä saadaan metodeilla getTitle, *getDescription*, *getSummary*, *getVersionId* ja *getAuthors*, missä getAuthors palauttaa tekijätiedot taulukossa. Lisäyksen etäkutsu tehdään PicasawebService-luokan insert-metodilla. Lisäysmetodi palauttaa Picasa-palveluun luodun albumin tiedot, sisältäen albumin Picasa-tunnistetiedot. Muokkaus- ja poistometodit saavat parametrina GoogleAlbumEntry-luokan olioilmentymän, missä muokkausmetodilla tehtävä päivitys voi kohdistua samoihin attribuutteihin kuin

lisäysmetodilla. Molempien metodien syöteparametriolio sisältää pakollisena attribuuttina syöte-URL-osoitteen, jota ilman muokkausta tai poistoa ei voida suorittaa. Muokkausmetodi käyttää etäkutsuihin PicasawebService-luokan getEntry-metodia ja sen palauttaman entry-olion update-metodia. Poistometodi käyttää puolestaan PicasawebService-luokan delete-metodia. Muokkausmetodi palauttaa päivitetyn GoogleAlbum-ilmentymän, missä päivitys voi kohdistua samoihin attribuutteihin kuin lisäysmetodilla. Poistometodi palauttaa ServiceMetaData-luokan ilmentymäolion, jolla ilmaistaan poiston onnistuminen.

UpdatePhotoMetadataByAuth- ja *deletePhotoByAuth*-metodeilla voidaan muokata valokuvan tietoja tai poistaa valokuvia Picasa Web Albums -palvelusta. Metodit suorittavat etäkutsut samaan tapaan kuin valokuva-albumien muokkaus- ja poistometodit. Molemmat metodit käyttävät Picasa-web-palvelusta saatua muokkausosoitetta, joka toimii syöteparametriolioiden pakollisena attribuuttina. Muokkausmetodilla voidaan päivittää valokuvasta samat metatiedot kuin albumin päivitysmetodilla albumista. Muokkausmetodi palauttaa muokatun valokuvan tiedot, kun taas poistometodi pelkän statustiedon poiston onnistumisesta.

5 Yhteenveto

Internet sisältää monia web-palveluja, joiden tarjoamia palveluja voidaan hyödyntää web-sovelluksissa. Internetistä löytyvät web-palvelut myös kehittyvät nopeasti. Web-palvelujen hyödyntämiskynnystä on saatu madallettua tarjoamalla erilaisia Web API -rajapintoja. Kohdassa 3.1 kuvatut Googlen, Amazonin ja eBayn API-rajapinnat tarjoavat monipuoliset ja moniulotteiset työvälineet web-palvelujensa käyttämiseksi. Kaikki edellä mainitut API-rajapinnat sisältävät asiakaskirjastoja useille ohjelmointikielille, esimerkikkilähdekoodeja, dokumentaatiota kuin myös tukipalvelujakin, joista voidaan ottaa esille esimerkiksi vertaistukea tarjoavat yhteisöpalvelut. Nämä seikat osaltansa madaltavat kynnystä käyttää API-rajapintojen avulla hyödynnettäviä web-palveluja.

Google Data API -rajapinta osoittaa sen, että API-rajapinnoilla voidaan saavuttaa konkreettista hyötyä web-palvelujen hyödyntämistä ajatellen. Kohdassa 3.3 esitetyistä Java-esimerkeistä voi havaita, että asiakassovelluksen ei tarvitse tietää Googlen web-palveluista yksityiskohtaisia palveluspesifisiä tietoja. Lähes ainoa web-palveluihin viittaava asia on palvelun URL-osoite ja parametrit, joiden muodostaminen on kuvattu Google Data API:n dokumentaatiossa ja esimerkeissä. Muuten web-palvelukutsut tapahtuvat asiakassovelluksen kannalta läpinäkyvästi, jolloin asiakassovelluksen ei tarvitse tiedostaa tehtyjä web-palvelukutsuja.

Web API -rajapintojen lisäksi tutkielmassa tarkasteltiin koostepalvelujen rakentamista välityspalvelurajapintamallilla. Välityspalvelurajapinnat tarjoavat asiakassovelluksille laajemman kirjon web-palveluja samalla rajapinnalla ja ontologialla. Välityspalvelurajapinnat vähentävät näin ollen asiakassovelluksen ohjelmointiin tarvittavaa työtä ja nopeuttavat siten uusien web-palvelujen käyttöönottoa. Itse välityspalvelurajapinnan rakentaminen muodostuu kuitenkin jossain määrin työlääksi. Luvussa 4 esitellyn WEB-API-välityspalvelurajapinnan toteutus osoittautui verrattaen työlääksi, koska viestimuodot ja tietotyypit muunnettiin WEB-API:n omiin tietomuotoihin. Täten uusien web-palvelujen lisääminen tulee vaatimaan aina lisätyötä, koska välityspalvelun täytyy kyetä käsittelemään koostamiensa web-palvelujensa tietotyypit ja ontologiat. Tästä syystä olisi hyvä, jos tietotyyppimuunnokset saataisiin automatisoitua koneellisesti tehtäviksi. Tämä voitaisiin yhdistää kohdassa 3.4 esitettyyn muunnosabstraktioon, jolloin välityspalvelurajapinnan toteutusta saataisiin automatisoitua riittävästi, jotta voitaisiin tarjota uusia web-palveluja ilman merkittävää lisätyötä ja kehitysaikaa.

Viitteet

Amazon Web Services API (2008) *Amazon Web Services*. Internet web-sivu, Amazon.com, http://www.amazon.com/gp/browse.html?node=3435361 (27.5.2008).

Bih J. (2006) Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions. *Ubiquity*, **7**(30), 1-1.

Bussler C. (2006) Message mediation in composite Web Services. *Int. J. Web Engineering and Technology*, **2**(4), 373-395.

Chinnici R., Hadley M., Mordani R. (2005) *The Java API for XML Web Services (JAX-WS) 2.0*. Sun Microsystems, Inc., Santa Clara, CA, USA.

Ebay API (2008) *EBay developers program*., Internet web-sivu, Ebay, http://developer.ebay.com/common/api/ (27.5.2008).

Fitzgerald M. (2008) *Cloud Computing's Promise: A Power Grid for the Net*. Internet web-sivu, Top Tech News, http://www.toptechnews.com/story.xhtml?story_id=022001FOQFWK (10.6.2008).

Google Account Authentication API (2008) *Account Authentication API.*, Internet web-sivu, Google, http://code.google.com/apis/accounts/ (24.2.2008).

Google Data API (2008) *Google Data APIs*. Internet web-sivu, Google, http://code.google.com/apis/gdata/ (28.5.2008).

Gosnell, Denise M. (2005) *Professional Web APIs: Google, eBay, Amazon.com, MapPoint, FedEx*. John Wiley & Sons, Incorporated, Hoboken, NJ, USA.

Kinnunen M. (2008) *WEB-API-välityspalvelurajapinta*. Systemoinnin menetelmien harjoitustyö. Joensuun yliopisto, tietojenkäsittelytiede.

Kung-Kiu L., Cuong T. (2007) Composite Web Services. *2nd Workshop on Emerging Web Services Technology (WEWST07)* (toim. Pautasso C., Gshwind T.), CEUR-WS.org, Vol-313.

Nottingham R., Sayre R. (2005) *The Atom Syndication Format*. Internet web-sivu, Network Working Group, http://atompub.org/rfc4287.html (10.6.2008).

Picasa Web Albums Data API (2008) *Web Albums Data API*. Internet web-sivu, Google, http://code.google.com/apis/picasaweb/reference.html (24.2.2008).

Prescord P. (2002) *REST and the Real World*. Internet web-sivu, O'reilly web-services.xml.com, http://webservices.xml.com/pub/a/ws/2002/02/20/rest.html (10.6.2008).

Tyagi S. (2006) *RESTful Web Services*. Internet web-sivu, Sun Developer Network, http://java.sun.com/developer/technicalArticles/WebServices/restful/ (25.5.2008).

W3C (2000) *Simple Object Access Protocol (SOAP) 1.1*. Internet web-sivu, World Wide Web Consortium, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (24.5.2008).

W3C (2001) *Web Services Description Language (WSDL) 1.1*. Internet web-sivu, World Wide Web Consortium, http://www.w3.org/TR/wsdl/ (24.5.2008).

W3C (2004a) *Web Services Architecture*. Internet web-sivu, World Wide Web Consortium, http://www.w3.org/TR/ws-arch/ (24.5.2008).

W3C (2004b) *Using Qualified Names (QNames) as Identifiers in XML Content*. Internet web-sivu, World Wide Web Consortium, http://www.w3.org/2001/tag/doc/qnameids.html (11.6.2008).

W3C (2006) *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. Internet web-sivu, World Wide Web Consortium, http://www.w3.org/TR/REC-xml/ (11.6.2008).

Winer D. (2003) *RSS 2.0 Specification*. Internet web-sivu, Berkman Center, http://cyber.law.harvard.edu/rss/rss.html (10.6.2008).

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, Crawfordsville, Indiana, USA.