Web-palvelu yritysten välisessä viestinnässä

Riina Pakarinen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Erkki Mäkinen
Toukokuu 2007

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pakarinen Riina: Web-palvelu yritysten välisessä viestinnässä
Pro gradu -tutkielma, 48 sivua
Toukokuu 2007

Tässä tutkielmassa käsitellään web-palveluja tehokkuuden ja turvallisuuden näkökulmasta. Aluksi käsitellään lyhyesti erilaisia arkkitehtuuriratkaisuja viestin välitykseen sekä tärkeimpiä väliohjelmistojen piirteitä. Väliohjelmisto on yksi ratkaisuista teknologialtaan erilaisten osapuolten väliseen viestin välitykseen. Tämän jälkeen käsitellään lyhyesti erilaisia ratkaisuvaihtoehtoja yritysten väliseen viestintään ja vertaillaan niitä web-palvelun ominaisuuksiin. Ratkaisuvaihtoehdoissa käydään läpi esimerkiksi etäproseduurikutsu, joka on vanhin esiteltävistä ratkaisuista. Lisäksi käsitellään kokonaisratkaisuja kuten RosettaNet sekä EbXml. Näiden vastapainona esitellään myös Java-RMI ja CORBA, jotka mahdollistavat liiketoimintaprosessin toteuttajalle enemmän vastuuta ja vapautta palvelun toteutuksessa. Tutkielman loppuosassa käsitellään tarkemmin erilaisia ratkaisuja web-palvelun tietoturvan sekä tehokkuuden parantamiseen. Tietoturvan arvioinnissa keskitytään tiedon luottamuksellisuuden takaamiseen sekä lähettäjän autentikointiin. Tehokkuuden käsittelyssä esitellään erilaisia ratkaisuvaihtoehtoja jäsentämisprosessin optimointiin.

Avainsanat ja -sanonnat: Web-palvelu, XML, SOAP, väliohjelmistot, tehokkuus, turvallisuus

Sisällys

1. Johdanto

Liiketoimintaprosessilla tarkoitetaan toimintaa, joka tukee yrityksen missiota [Bussler, 2001]. Esimerkki tällaisesta prosessista voi olla valituksen käsittely tai tilauksen vastaanotto. Yritykset ovat jo vuosia automatisoineet sisäisiä liiketoimintaprosesseja. Tällöin esimerkiksi syötetty tilaus näkyy välittömästi myös varaston puolella katevarauksena. Parantuneet verkkoyhteydet tarjosivat tilaisuuden myös yritysten väliseen sähköiseen viestintään [Brodie, 2000]. Silloin esimerkiksi tilausten siirto asiakkaalta tuotteiden toimittajalle nopeutui. Viime vuosikymmenten aikana on syntynyt monia uusia teknologioita, joiden avulla yritykset voivat tehostaa liiketoimintaprosesseja, joiden toisena osapuolena on toinen yritys. Näitä teknologioita kutsutaan väliohjelmistoiksi. Niitä tarvitaan, sillä prosessin osapuolilla voi olla käytössään erilaisia ohjelmistoalustoja sekä päätelaitteita. Väliohjelmistojen tarkoituksena on selvittää erilaisista tietojärjestelmistä sekä päätelaitteista johtuvat ongelmat. Haasteita riittää, sillä esimerkiksi sulautetun tietotekniikan visiossa ounastellaan, että tulevaisuudessa erilaisia päätelaitteita on nykyistä enemmän [Satyanarayanan, 2001; Weiser, 1993].

Vanhimmaksi väliohjelmistoksi mielletään EDI-dokumenttien standardi, organisaatioiden välinen tiedonsiirto [ASC X12, 2006]. Se on saanut viime vuosina kilpailijakseen kevyemmän version, web-palvelun. Web-palvelu koostuu yleensä XML-kieleen pohjautuvasta SOAP-formaatista [W3C, 2003] sekä WSDL-kuvailukielestä [Christensen et al,. 2001]. Web-palvelun vahvuudeksi nimetään usein sen alusta-riippumattomuus [Jun et al., 2006]. Sillä on merkitystä, jos palveluntarjoajalla on asiakkaita, joilla on erilainen tietojärjestelmä. Alustariippumattomuuden lisäksi yritysten välisessä viestinnässä on tärkeää huolehtia tietojen ja tietojärjestelmien turvallisuudesta. Lisäksi erilaiset päätelaitteet ja niiden käyttämät erilaiset tietoliikenne-yhteydet, kuten langaton yhteys, asettavat tehokkuusvaatimuksia väliohjelmistoille [Kangasharju et al., 2005].

Tutkielma esittelee aluksi yritysten välisen viestinnän konseptia sekä väliohjelmistojen ominaisuuksia. Tämän jälkeen käsitellään erilaisia viestin välityksessä käytettyjä teknologioita. Työn loppuosa keskittyy yhteen ratkaisuvaihtoehtoon, web-palveluun. Tätä ratkaisua arvioidaan tarkemmin turvallisuuden ja tehokkuuden osalta.

2. Yritysten sähköisestä viestinnästä

2.1. Yritysten välisen viestinnän arkkitehtuureista

Tyypillinen yritysten välisen viestinnän arkkitehtuurimalli koostuu kahdesta yrityksestä, jotka ovat yhteydessä toisiinsa tietoverkon kautta. Medjahed ja muut [2005] kuvaavat arkkitehtuuria, jossa molemmilla yrityksillä on oma sisäinen tietojärjestelmänsä sekä rajapinta ulkoista viestintää varten. Ulkoisen viestinnän rajapinta huolehtii viestin lähetyksestä ja vastaanotosta sekä viestin välityksen turvallisuudesta. Yritykset voivat viestiä keskenään itse sopimiensa viestintäsääntöjen mukaan tai ne voivat käyttää olemassa olevaa standardia kuten EDI [ASC X12, 2006] tai standardin ympärille rakennettua ohjelmistoa kuten ebXML [Oasis, 2006a; ebXml, 2006]. Yritysten välistä viestintää tehostavia teknologioita kutsutaan väliohjelmistoiksi. Termillä on monta määritelmää. Ciancarini [2000] kuvailee väliohjelmistoa työkaluksi, joka abstrahoi erilaisten tietojärjestelmien erityiset piirteet, kuten käytetyn ohjelmointikielen tai verkkoyhteyden. Työkalu muodostaa yhtenäisen standardin ja alustan palvelujen tarjoamiselle ja pyytämiselle, jota palvelun osapuolet voivat kutsua riippumatta omasta tietojärjestelmästään.

Väliohjelmistolla on monia tehtäviä. Kerroksellisessa arkkitehtuurissa väliohjelmiston tehtävät voidaan jakaa kolmeen kerrokseen: kommunikaatiotaso, kontekstitaso ja liiketoimintaprosessitaso [Medjahed et al., 2005]. Kommunikaatiotasolla tarkoitetaan protokollaa, jolla tieto siirretään yritykseltä toiselle. Kontekstitasolla tarkoitetaan mallia tai standardia, jonka avulla yritykset ymmärtävät, mitä välitetty viesti sisältää. Malli kertoo, onko viesti esimerkiksi tilausvahvistus vai hintapyyntö ja mitä tietoja se sisältää. Liiketoimintatasolla tarkoitetaan yritysten sopimusta kaupankäynnistä, esimerkiksi minkälaisia viestejä loppuunsaatettuun tilausprosessiin kuuluu tai kuinka olemassa olevaa tilausta voi muuttaa. Yksi vaihtoehto väliohjelmistoksi on web-palvelu. Sen arkkitehtuuri koostuu liikenne-, paketointi-, informaatio-, palvelu- sekä kirjastokerroksesta [Engelen, 2004]. Liikennekerros huolehtii viestien välittämisestä yleensä HTTP-protokollan avulla. Paketointikerros välittää viestin SOAP-tegnologian määrittelemässä muodossa. Informaatiokerros kuvaa välitettävän viestin XML-kielellä. Palvelukerros kuvaa yhden yrityksen tarjoamia palveluita, esimerkiksi rajapinnassa olevia funktiokutsuja, kuten tilauksen lähettäminen. Kirjastokerros sisältää tietoa eri yrityksistä ja niiden tarjoamista palveluista.

2.2. Väliohjelmistojen arvioinnista

Väliohjelmistoa suunnitellessa, valitessa tai arvioidessa tulee ottaa huomioon monia seikkoja. Medjahed ja muut [2005] ovat koonneet kattavan listan:

- yritysten kytkentä,
- heterogeenisyys,

- turvallisuus,
- autonomia,
- ohjattavuus
- mukautuvuus.

Yritysten kytkennällä tarkoitetaan sitä, täytyykö asiakasyritysten tehdä sopimus palveluntarjoajan kanssa palvelun käytöstä etukäteen vai onko yrityksen tarjoama palvelu vapaasti muiden yritysten käytettävissä. Esimerkiksi palveluntarjoajalla voisi olla verkkosivullaan ohjeet, kuinka tilauspalvelun avulla voi jättää tilauksia. Tämän jälkeen mikä tahansa yritys voisi käyttää palvelua. Toinen vaihtoehto on, että yritys avaa palvelunsa vain asiakasrekisterissään oleville yrityksille. Heterogeenisyys viittaa viestinnän osapuolten erilaisuuteen. Esimerkiksi laskun muoto ja sen sisältämät tietotyypit voivat poiketa toisistaan. Viestinnän osapuolten täytyy pystyä käyttämään väliohjelman määrittämää standardia. Turvallisuusnäkökulma liittyy asiakkaan tunnistukseen, käyttöoikeuden määrittämiseen sekä tiedon yhtenäisyyden varmistamiseen. Autonomia viittaa siihen, kuinka riippumaton viestinnän osapuolten valitsema tekniikka on väliohjelmistosta. Ääritapauksessa osapuoli tarjoaa vain käytetyn rajapinnan, mutta käytännössä osapuolet joutuvat tekemään esimerkiksi tietotyyppimuutoksia ennen rajapinnan toimintaa, joilloin ohjelmien välillä syntyy väistämättä riippuvuutta. Ohjattavuus liittyy myös tietojen näkyvyyteen sekä palvelun käytettävyyteen, esimerkiksi täytyykö tilauksen yhteydessä nähdä myös tavaran varastoarvo. Mukautuvuusnäkökulma viittaa välitettävän viestin muuttumiseen ja viestin muodostamisen joustavuuteen. [Baligand and Monfort, 2004].

Väliohjelmistoa valittaessa tärkeimmät kriteerit täytyy priorisoida, sillä yhden kriteerin hyvä toteutus voi hankaloittaa toisen kriteerin toteutumista. Esimerkiksi aiemmin mainittu kerroksellinen arkkitehtuuri helpottaa ohjelmiston kehitystä sekä mukautuvuutta [Demir et al., 2005]. Toisaalta se voi myös hankaloittaa esimerkiksi tehokkuuden optimointia ja ei-toiminnallisten vaatimusten, kuten turvallisuuden tai alustariippumattomuuden, yhtäaikaista toteuttamista [Demir et al., 2005; Baligand and Monfort, 2004]. Väliohjelmiston valintaan voi vaikuttaa myös yrityksessä käytössä oleva teknologia, jonka kanssa väliohjelmiston tulee toimia ilman suuria muutoksia.

Tehokkuuden ja turvallisuuden yhtäaikaista toteutusta voi hankaloittaa esimerkiksi yritysten käyttämät palomuurit. Yrityksen suojaavat tavallisesti sisäiset tieto-järjestelmänsä ulkoiselta liikenteeltä palomuurilla. Väliohjelmistoilla on erilaisia keinoja ylittää palomuuri. Yritysten välinen tietoliikenne voi kulkea esimerkiksi olemassa olevien protokollien, kuten HTTP, avulla, jolloin se käyttäisi porttia 80 ja viestit läpäisisivät palomuurin silloin, kun HTTP-liikenne olisi sallittu. Muita vaihtoehtoja ovat esimerkiksi portin avaaminen ja sulkeminen yhteyskohtaisesti tai

etukäteen yritysten välillä sovitut portit. Jos viestin välitykseen käytetään HTTP-protokollaa, viestin lähettämisen tehokkuus voi kärsiä, sillä HTTP:n käyttämällä TCP-protokollalla on suuri verkkolatenssi [Phan et al., 2006]. Jos puolestaan viestiliikennettä varten avataan erikseen portteja palomuuriin, ohjelmiston alustariippumattomuus sekä mukautuvuus voivat kärsiä [Damiani et al., 2001]. Langattomat päätelaitteet asettavat palveluille puolestaan korkeat tehokkuusvaatimukset, koska niiden virtalähde on rajallinen [Kangasharju et al., 2005].

Ohjelmiston luotettavuus, turvallisuus sekä alustariippumattomuus voivat olla myös hankalasti yhteensovitettavia ominaisuuksia Jos väliohjelmistossa on huolehdittu tietoliikenteen turvallisuudesta, se on yleensä toteutettu arkkitehtuurien matalammilla kerroksilla. Tällöin väliohjelmiston käyttäjä ei pääse käyttämään niitä suoraan, vaan väliohjelmisto on riippuvainen alustasta, johon se on kehitetty. Alustariippumattomuus paranee, jos väliohjelmisto ei ota kantaa turvallisuuteen, vaan siitä huolehtiminen jää ohjelmiston käyttäjälle. [Baligand and Monfort, 2004] Tällöin tosin turvallisuus-mekanismeja tarvinnee olla yhtä paljon kuin ohjelmistoa käyttäviä yrityksiä.

3. Väliohjelmistoja

Väliohjelmisto voi olla kevyt tai raskas riippuen siitä, mitä ominaisuuksia se toteuttaa. Kevyt väliohjelmisto tai siihen verrattava protokolla toteuttaa yleensä vain perustehtävät kuten viestistandardin määrittelyn. Raskas väliohjelmisto puolestaan voi toteuttaa tukiominaisuuksia kuten turvallisuus, viestien lähetys tai rekisterin ylläpito. [Gergic et al., 2000]

Seuraavassa on esitelty käytetyimpiä standardeja ja niiden ympärille rakennettuja ohjelmistoja. RPC eli etäproseduurikutsu on esitellyistä standardeista vanhin [IETF, 2006]. Se esittelee mallin, mutta ei määrittele sen toteutustapaa. Toista ääripäätä puolestaan edustavat EDI [ASC X12, 2006], RosettaNet [RosettaNet, 2006] ja ebXml [ebXml, 2006]. Ne ovat ohjelmistoja, jotka sisältävät viestien välityksen lisäksi muitakin liiketoiminnan tukitoimintoja. Java-RMI [Sun Microsystems, 2006b], DCOM [Microsoft, 2006a] ja CORBA [OMG, 2006] ovat objektien jakamiseen perustuvia teknologioita. Tällöin ennalta määrätyt tietotyypit eivät rajoita viestien sisältöä. [Damiani et al., 2002]

3.1. RPC

Etäproseduurikutsu (RPC) on yksinkertainen malli prosessille, joka tapahtuu kahdella eri päätteellä [IETF, 2006]. Prosessin käynnistäjä on asiakas, joka kutsuu palvelimen tarjoamaa palvelua. Palvelun suorituksen jälkeen palvelin lähettää vastauksen asiakkaalle. Kutsu voi sisältää funktion ja parametreja. RPC voidaan toteuttaa erilaisilla siirtoprotokollilla kuten HTTP. [Sun Microsystem, 2006]

Työkaluja, jotka toteuttavat etäproseduurikutsun, ovat mm. Sunin kehittämä Sun Microsystems RPC [Sun Microsystem, 2006] sekä UserLand Softwaren ja Microsoftin XML-RPC [Winer, 1999]. Sun Microsystems RPC on määritelty XDR-kielellä [Sun Microsystems, 1987]. XDR-kielen avulla voidaan kuvata dataa, ja siinä on määritelty tavallisimmat tietotyypit kuten kokonaisluvut [Sun Microsystems, 1987]. XML-RPC on yksinkertainen spesifikaatio, joka määrittelee etäproseduurikutsun XML-kielen avulla. Sen tavoitteena on olla helposti toteutettavissa ja ymmärrettävissä. XML-RPC käyttää HTTP-protokollaa, jolloin viestien kulkemista varten palomuuria ei tarvitse muuttaa. [Winer, 1999]. Tämä puolestaan vähentää tehokkuutta varsinkin langattomissa päätelaitteissa, koska HTTP käyttää TCPIP-protokollaa [Phan et al., 2006].

RPC-mallin hyväksi puoleksi on mainittu sen tehokkuus [Jun et al., 2006]. Huonoksi puoleksi on puolestaan mainittu se, että työkalut toteuttavat RPC-mallin mukaisen

kutsun eri tavoin. Tällöin työkalut ovat keskenään yhteensopimattomia [Jun et al., 2006]. Medjahedin ja muiden [2005] luonnehdinnan mukaan se aiheuttaa vahvaa kytkeytymistä eri yritysten välillä. Tämä puolestaan hankaloittaa yritysten verkostoitumista, koska palvelun tarjoajan täytyisi tukea erilaisia teknologioita saavuttaakseen palveluja käyttäviä asiakkaita.

3.2. EDI

EDI (Electronicic data interface) on standardi, jota käytetään organisaatioiden välisessä tiedonsiirrossa [ASC X12, 2006]. Standardi määrittelee liiketoimintaprosesseja, kuten ostotilauksen teko. EDI-standardin rakenne on jaettu kolmeen osaan: viittaus-, jäsennys- ja viestintäosa [Medjahed et al., 2005]. Ohjelmiston viittausosassa kerrotaan yrityksen tietojen sekä EDI-standardin osien suhde. Tällöin ohjelmistolle kerrotaan esimerkiksi, mikä yrityksen tiedoista vastaa standardissa määriteltyä tilausnumeroa. Ohjelmiston jäsennysosan vastuulla on muodostaa EDI-standardin mukaisia viestintädokumentteja. Dokumentin asiasisällön ohjelmisto saa viittauskerroksen muodostamista käsitesuhteista. Viestintäohjelma hoitaa standardimuotoisten viestien välityksen liiketoimintaprosessin osapuolien välillä. Viestien välitys tapahtuu suojatun yksityisen lisäarvoverkon avulla [Medjahed et al., 2005].

EDI-standardin tavoitteena on pienentää yrityksen välisen viestinnän kuluja. Tämä tapahtuu automatisoimalla viestintää, jolloin siihen käytetty aika vähenee. Kuluja voidaan vähentää myös kasvattamalla viestinnän tarkkuutta ennalta määrätyn standardin avulla [ASC X12, 2006]. Koska viestien välitys tapahtuu suojatussa verkossa, julkisen verkon turvallisuusuhat on rajattu viestiliikenteen ulkopuolelle. Yksityisen verkon käytöllä on myös heikkoutenaan sen hinta. Yksityisen verkon käyttö maksaa enemmän kuin julkisen. EDI-ohjelmiston hinnoittelu perustuu patenteille, joten sen käyttöönottokulut nousevat melko huomattaviksi. Korkeahkon hinnan lisäksi EDI-standardin huonona puolena on myös liiketoimintaprosessien kankeus. [Medjahed et al., 2005] Liiketoimintaprosessit, kuten laskun lähetys ja sen sisältämät tiedot, on ennalta määrätty EDI-dokumenteissa. Standardeja tosin päivitetään ajoittain [ASC X12, 2006], mutta ennen hitaasti tapahtuvia muutoksia yritysten on hankalaa välittää toisilleen standardimuotoisten viestien ulkopuolista informaatiota.

3.3. RosettaNet

RosettaNet on vuonna 1998 luotu standardi, joka määrittelee liiketoimintaprosesseja sähköiseen kaupankäyntiin [Dogac et al., 2002]. Liiketoimintaprosessit on määritelty XML- ja UML-kielillä [RosettaNet, 2006]. Ne sisältävät viestinnässä käytettävät termit sekä prosessin aikana vaihdetut viestit. RosettaNet-standardi koostuu kehys-ohjelmistosta, sanakirjasta sekä prosessikuvauksista. Kehysohjelmisto huolehtii viestien välittämisestä ennalta määritellyille prosessin osapuolille. Sanakirja sisältää mahdolliset

käytössä olevat viestin termit, kuten laskunumeron. Prosessikuvaus sisältää termien käyttöä koskevat säännöt, kuten viestien järjestyksen ja rakenteen. [Dogac et al., 2002]

RosettaNet-standardin tavoitteena on vähentää organisaatioiden logistiikka- ja investointikustannuksia sekä transaktioiden prosessointiin kuluvaa aikaa. RosettaNet-dokumenttien standardit ovat saatavilla verkossa ilman eri maksua. [RosettaNet, 2006] Ohjelmiston sisältämässä sanakirjassa on myös yrityksen sähköisen rajapinnan perustiedot, jotka vuorostaan eivät ole julkisia [Dogac et al., 2002]. RosettaNet on keskittynyt automatisoimaan sähköistä liiketoimintaa olemassa olevien liike-kumppaneiden kesken [Dogac et al., 2002]. Medjahedin ja muiden [2005] mukaan RosettaNet on keskittynyt tiukasti kytkeytyneiden yritysten väliseen liiketoimintaan. Dogac [2002] mainitsee RosettaNet-standardin kehitystarpeeksi julkisen liiketoiminta-rekisterin. Tällöin yritysten olisi vaivattomampaa löytää tarvitsemiaan sähköisiä palveluja ja solmia liikesuhteita uusien kumppaneiden kanssa. RosettaNet-standardin heikkoudeksi lasketaan myös se, että liiketoimintaprosessien kulku on ennalta määrättyä [Dogac et al., 2002].

3.4. EbXml

EbXml on OASIS:n ja UN/CEFACT:n (United Nations Centre for Trade Facilitation and Electronic Business) kehittämä infrastruktuuri [Dogac et al., 2002]. Sen tavoitteena on jakaa sähköisessä kaupankäynnissä tarvittavaa tietoa turvallisesti ja alustariippumattomasti [ebXml, 2006]. Se koostuu liiketoiminnan osapuolien kuvauksesta, liiketoimintaprosessien kuvauksesta, viestienvälityskerroksesta sekä dokumenttien rekisteristä [Dogac et al., 2002]. Liiketoiminnan osapuolien kuvauksessa määritellään yrityksen tietoja sähköisen liiketoiminnan näkökulmasta. Siinä kerrotaan esimerkiksi yrityksen sähköinen rajapinta sekä yhteystiedot. Liiketoimintakuvaus kuvaa yrityksen tarjoamia sähköisiä palveluita ja rajapintoja. Molemmat kuvaukset ovat saatavilla sekä DTD- että XML-skeema -formaatissa ilman lisämaksua. [Dogac et al., 2002] Viestikerros vastaa viestien välittämisestä yleisten protokollien kuten SMTP:n, HTTP:n, and FTP:n avulla [Medjahed et al., 2005]. Rekisteri puolestaan tallentaa viestit. Rekisterissä on tallessa esimerkiksi ostolaskut sähköisessä muodossa kirjanpitoa varten. [Dogac et al., 2002].

EbXml:n tavoitteena on helpottaa sähköisen liiketoiminnan käynnistämistä uusien liikekumppaneiden kanssa. EbXml:n avulla yritys voi julkaista kuvauksen tarjoamistaan sähköisistä palveluista. Tällöin rekisterin lukijat voivat tietojen perusteella käynnistää liiketoiminnan palveluntarjoajan kanssa [ebXml, 2006]. Palvelukuvausten helppo löytyminen vaatii kuitenkin tuekseen hakemiston, jossa on viitteet erillisiin palveluntarjoajiin [Dogac et al., 2002]. Tällainen hakemisto on esimerkiksi UDDI [Oasis, 2004].

3.5. Java-RMI

Java-RMI on oliopohjainen teknologia, joka mahdollistaa olioiden kutsumisen etäkoneilla. Teknologian tavoitteena on tehdä etäkoneen funktioiden käyttäminen helpoksi. Etäkutsu toteutetaan määrittämällä funktio etäfunktioksi sekä huolehtimalla funktion kutsujan oikeudesta käyttää etäfunktiota. Java-RMI-teknologiaa voi käyttää normaalien URL-protokollien avulla kuten HTTP, FTP ja FILE. [Sun Microsystems, 2006b]

Java-RMI-teknologian vahvuudeksi on mainittu Javan polymorfinen perintä, joka tekee teknologian käytöstä joustavampaa [Govindaraju et al., 2000]. Lisäksi etäkutsujen avulla voidaan käsitellä olioita perinteisten tietotyyppien sijaan [Engelen, 2004]. Teknologian huonona puolena on sen kieliriippuvuus. Lisäksi takaisinkutsut palvelimelta sovelmalle eivät toimi, jos tietoliikenteeseen käytetään julkisia tunnettuja portteja, kuten portti 80, joten yksi teknologian eduista jää hyödyntämättä. Takaisinkutsu toimii, jos käytetään muita portteja, mutta silloin palomuurien asetuksia täytyy muuttaa. [Sun Microsystems, 2006b] Tämä puolestaan johtaa ohjelmiston alustariippumattomuuden sekä mukautuvuuden vähenemiseen. [Damiani et al., 2001]. Teknologia on myös RPC-mallin mukaisia teknologioita tehottomampi, koska tiedon jäsentäminen siirrettävään muotoon ja siitä takaisin lisäävät prosessointia [Jun et al., 2006].

3.6. DCOM

Microsoft COM -teknologian tarkoituksena on helpottaa eri prosesseja tai ohjelmia jakamaan niiden käyttämiä objekteja [Microsoft, 2006a]. DCOM on Microsoft COM-teknologiaan pohjautuva väliohjelmistokehys [Medjahed et al., 2005]. Sen avulla objekteja käyttävät prosessit ja ohjelmat voivat sijaita eri koneilla [Microsoft, 2006a]. Microsoftin .NET-teknologia [Microsoft, 2006b] tukee COM-objekteja ja Microsoft kehottaakin toteuttajia käyttämään tuoreempaa .NET-teknologiaa [Microsoft, 2006a]. DCOM-ohjelmistokehys on alustariippumaton, mutta sitä käytetään yleensä Windows-käyttöjärjestelmissä [Engelen, 2004]. DCOM-teknologia käyttää COM-objektien välittämiseen dynaamisia portteja. Niiden käyttö vaatii palomuurin asetusten muuttamista, joka puolestaan heikentää ohjelmiston alustariippumattomuutta sekä mukautuvuutta. [Damiani et al., 2001]

3.7. CORBA

CORBA (Common Object Request Broker Architecture) on OMG:n avoin ja toimittajariippumaton standardi. Toisin sanoen CORBA-standardiin perustuvat eri toimittajien ohjelmat ovat yhteensopivia. Asiakaan ja palvelun tarjoajajan ohjelmistot voivat olla koodattu eri kielillä, kunhan ne käyttävät viestin muodostamiseen CORBA-kirjastoa, joka tukee käytetyimpiä kieliä kuten Java ja C++. CORBA-standardi perustuu

viestien määrittelykieleen (OMG IDL) sekä viestejä välittäviin protokolliin (IIOP ja GIOP) [OMG, 2006]. CORBA-standardin sanotaan olevan suhteellisen raskas [Damiani et al., 2001], ja se onkin yleensä käytössä enimmäkseen suurilla ja osittain myös keskisuurilla yrityksillä [OMG, 2006]. Kokonsa puolesta CORBA-standardi sopii huonosti sulautettuihin järjestelmiin [Damiani et al., 2001], mutta siihenkin löytyy sopivat ohjelmistot [OMG, 2006]. Standardin huonona puolena on sen käyttämä IIOP-protokolla, joka käyttää dynaamisia portteja. Se vaatii siis toimiakseen palomuurin asetusten muuttamista sekä asiakkaalta että palvelujen tarjoajalta [Damiani et al., 2001].

3.8. Vertailua

Yllämainitut teknologiat ovat vain pieni osa tarjolla olevista ratkaisuvaihtoehdoista yritysten väliseen sähköiseen viestintään. Osa teknologioista kilpailee tehokkuudella, osa puolestaan tukitoimintojen kattavuudella. EDI, RosettaNet ja ebXml ovat tukitoiminnoiltaan melko kattavia. EDI-ohjelmiston huonona puolena on hinta, joka muodostuu sekä yksityisen verkon käytöstä että patenttimaksuista [Medjahed et al., 2005]. RosettaNet-ohjelmiston dokumentointi on sen sijaan saatavilla ilman lisämaksua [RosettaNet, 2006]. EbXml puolestaan käyttää tietoliikenteessä HTTP-protokollaa, joten sille ei kerry erillistä lisämaksua verkon käytöstä [Medjahed et al., 2005]. Toisaalta EDI-standardin käyttämä yksityinen verkko on turvallisempi kuin julkinen verkko. Sekä RosettaNet-, ebXml- että EDI-ohjelmistoja on moitittu käyttöönottokustannuksista sekä ohjelmien hitaasta muuttumisesta liiketoiminta-prosessien muuttuessa [Dogac et al., 2002].

RPC-malli sisältää liiketoimintaprosessien määrittelyn sijaan vain mallin viestinvälitykseen [IETF, 2006]. Malli voidaan toteuttaa eri protokollilla [Sun Microsystem, 2006]. RPC-mallin hyvänä puolena on mainittu sen tehokkuus [Jun et al., 2006], joka tosin riippuu todennäköisesti myös mallin toteutuksesta. RPC-malli voidaan toteuttaa esimerkiksi XML-kielellä tai XDR-kielellä. Vaikka eri toteutusten viestit noudattavat RPC-mallia, ne eivät kuitenkaan ole keskenään yhteensopivia [Jun et al., 2006]. RPC-malli jättää mallin toteuttajan vastuulle esimerkiksi tietoliikenteestä ja tietoturvasta huolehtimisen. Suuremmissa väliohjelmistoratkaisuissa nämä ja muut tukitoiminnot on jo toteutettu yleisellä tasolla, joka ei tosin aina välttämättä vastaa täysin asiakkaan toiveita.

Java-RMI-, CORBA- ja DCOM-teknologiat perustuvat objektien siirtoon etäkoneiden välillä [Damiani et al., 2002]. Näissä teknologioissa siirrettäviä viestejä tai tietotyyppejä ei ole ennalta määritelty. DCOM- ja CORBA-teknologioiden huonona puolena on, että tietoliikenteen toimiminen vaatii palomuurien asetusten muuttamista [Damiani et al., 2001]. Java-RMI ja CORBA ovat avoimia standardeja, joten käyttöönotosta ei koidu lisäkustannuksia [Sun 2006; OMG, 2006]. Java-RMI-teknologian huonona puolena on

sen kieliriippuvaisuus. CORBA-teknologia on puolestaan hieman raskas esimerkiksi sulautettuihin järjestelmiin [Damiani et al., 2001]. Java-RMI-, CORBA- ja DCOM-teknologioissa viestien muodostus ja päivittäminen ovat helpompia toimenpiteitä kuin valmiilla ohjelmistoilla kuten EDI-standardi. Mutta ne toteuttavat vain viestinnän välttämättömät tehtävät kuten viestin välityksen. Tällöin käyttäjän täytyy toteuttaa itse tarvitsemansa tukitoimet kuten tietoturva, viestien varastointi tai rekisterin ylläpito.

3.9. Web-palvelu

3.9.1. Määritelmiä

W3C:n määrittelee web-palvelun ohjelmistoksi, jonka ulkoiset rajapinnat on määritelty ja kuvattu XML-kielellä. Ohjelmiston tarjoamat palvelut on löydettävissä verkosta ja ohjelmisto voidaan identifioida sen URI-osoitteeen [Network Working Group, 1998] perusteella. Muut ohjelmistot voivat käyttää web-palvelun tarjontaa XML-muotoisten viestien avulla internet-protokollien kautta. [W3C, 2004]

Web-palvelun osapuolia ovat palvelujen tuottaja, asiakas ja rekisteri [Medjahed et al., 2005]. Web-palvelun tuottaja kuvaa palveluja WSDL-kielellä. Tuottajan ja asiakkaan välisiä viestejä välitetään SOAP-protokollan määrittelemässä muodossa. Viestien siirtoprotokollana käytetään yleensä HTTP-protokollaa. Lisäksi tarjolla olevat palvelut voidaan tallentaa helposti saatavilla olevaan rekisteriin, kuten UDDI [Engelen, 2004]. Medjahed ja muut [2005] korostavat web-palvelun roolia asiakkaan ja palvelun tarjoajan välissä. Palveluntarjoaja julkaisee aluksi tietoja tarjoamistaan palveluista UDDI-rekisterissä, josta asiakas voi ne hakea. Tämän jälkeen asiakas käyttää palvelua muodostamalla XML-skeeman mukaan SOAP-viestejä, jotka vastaavat WSDL-kielen palvelunkuvauksia [Baligand and Monfort, 2004].

W3C:n määritelmä web-palvelusta korostaa sen kykyä kuvata ja julkaista tarjolla olevia palveluita. Casati ja Shan [2001] puolestaan määrittelevät web-palvelun olevan liiketoimintaa, joka on saatavilla internetissä ja jota voivat käyttää sekä ihmiset että ohjelmat. Määritelmä on laajempi ja korostaa web-palvelun kaupallisia mahdollisuuksia. Tsurin ja muiden [2001] määritelmän mukaan web-palvelu koostuu heikosti kytkeytyneistä eri yritysten ohjelmistoista, jotka käyttävät viestinnässä avoimia alustariippumattomia standardeja. He korostavat määritelmässään web-palvelun kilpailukykyisiä puolia, kuten alustariippumattomuutta.

3.9.2. Web-palvelun osat

WSDL (Web Services Description Language) on W3C:n kehittämä XML-pohjainen kieli. Sen avulla voidaan kuvata web-palvelun käyttöliittymä sekä sen toteutus. Kuvaus

sisältää esimerkiksi sekä tarjolla olevan palvelun että sille lähetettävän palvelupyynnön nimen. [Christensen et al,. 2001] Muita web-palvelun arkkitehtuurin kuvauskieliä ovat mm. WSFL [IBM Coorperation, 2001], WSCI [W3C, 2002a] ja BPEL4WS [IBM Coorperation, 2002a].

Palveluntarjoajat voivat tallentaa palvelukuvauksia julkiseen rekisteriin. UDDI (Universal Description, Discovery and Integration) on rekisteriohjelmisto, joka sisältää käyttöliittymän sekä tietojen tallennukseen että tietojen hakuun [Oasis, 2004]. Asiakkaat voivat etsiä tarvitsemiensa palvelujen tarjoajia palvelukuvausten perusteella rekisteristä.

SOAP (Simple Object Access Protocol) on W3C:n kehittämä protokolla, jonka avulla voidaan välittää XML-muotoisia viestejä [W3C, 2003]. Se on lisäksi riippumaton ohjelmointikielestä, alustasta ja siirtoprotokollasta [Govindaraju et al., 2000]. SOAP-protokolla voidaan toteuttaa esimerkiksi protokollien kuten HTTP:n, FTP:n ja SMTP:n päälle [Govindaraju et al., 2000]. Jos SOAP-muotoisia viestejä lähetetään HTTP-protokollan avulla, ne läpäisevät automaattisesti mahdolliset asiakkaan ja palveluntarjoajan välillä olevat palomuurit. Tällöin web-palvelun käyttö ei vaadi muutoksia palomuuriin, toisin kuin esimerkiksi Java RMI tai DCOM-teknologioiden käyttö. Tämä ominaisuus tekee web-palvelusta helpommin käytettävän ja laajennettavan vähentämällä osapuolten välisiä riippuvuuksia [Damiani et al., 2001].

SOAP-protokolla määrittää mallin, jonka avulla voidaan välittää komentoja ja parametreja asiakkaan ja palveluntarjoajan välillä [Damiani et al., 2001]. Mallin määritelmä käyttää apunaan XML-nimiavaruutta [W3C, 2006a] sekä XML-skeemamäärittelykieltä [W3C, 2004]. Viestin sisältönä oleva komento voi olla esimerkiksi hinnan kysyminen ja parametri tuotteen sarjanumero. Tällöin asiakas lähettää palveluntarjoajalle SOAP-muotoisen pyynnön ja palveluntarjoaja lähettää asiakkaalle vasteen. Viestinnässä ei tarvitse aina olla mukana lähetettä ja vastetta. Se voi koostua myös yksisuuntaisista viesteistä tai pidemmistä viestidialogeista. [Govindaraju et al., 2000] Esimerkki SOAP-muotoisesta lähetteestä koodikatkelmassa 1 ja vasteesta koodikatkelmassa 2.

SOAP-viesti koostuu kirjekuoresta (Envelope), joka on viestin juurielementti, sekä vartalo-osasta (Body), joka sisältää itse viestin. Sekä kirjekuori- että vartalo-osat ovat pakollisia. Kirjekuori voi myös sisältää otsikko-osan (header), johon voidaan tallentaa ohjelmakohtaisia tietoja esimerkiksi koodauksesta. Vartalo-osa voi viestin lisäksi sisältää virhe-osan (fault), johon kootaan lisätietoja ilmenneestä virheestä.

Koodikatkelmat 1 ja 2 ovat kuvitteellisesta laivanupotus-pelistä, jossa asiakas lähettää haluamansa koordinaatit pelille ja peli vastaa osuiko ammus.

```
POST /Peli HTTP/1.1
Host: www.peli.fi
Content-Type: text/xml; charset="UTF-8"
Content-Length: nnn
SOAPAction: "http://peli.fi/laivanupotus#ammu"
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-NV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:ammu xmlns:m="http://peli.com/laivanupotus">
         <x>5</x>
         <y>5</y>
      </m:ammu>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Koodikatkelma 1. SOAP-lähdeviesti

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="UTF-8"
Content-Length: nnn
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:ammuVaste xmlns:m="http://peli.com/laivanupotus">
         <osuma>1</osuma>
      </m:ammuVaste>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Koodikatkelma 2. SOAP-vasteviesti

4. Web-palvelun arviointikriteerejä

Web-palvelua käytetään useimmiten julkisessa verkossa, ja sen kautta kulkee useita viestejä. Nämä ominaisuudet aiheuttavat haasteita sekä palvelun turvallisuudelle että tehokkuudelle. Palvelussa kulkevat viestit voivat sisältää luottamuksellista tietoa ja niiden kautta voidaan vaikuttaa niitä prosessoivaan koneeseen. Tällöin palvelun tarjoajan tulee huolehtia siitä, että tarjoaa asiakkaalle ja itselleen riittävästi tietoturvaa. Palvelun tulee olla myös kilpailukykyinen suhteessa muihin vastaaviin palveluihin, joten sen täytyy tarjota turvallisuuden lisäksi tehokasta ja virheetöntä palvelua. Kun viestien koko voidaan pitää matalana ja tarvittava prosessointi vähäisenä, palvelun käyttäjäkuntaa voidaan kasvattaa esimerkiksi langattomiin päätelaitteisiin.

4.1. Tietoturva web-palvelussa

Web-palveluja käytetään nimensä mukaisesti useimmiten julkisessa suojattomassa verkossa. Tietoturva on tämän vuoksi tärkeä näkökulma web-palvelun suunnittelussa. Damiani ja muut [2002] muistuttavat, että alkuperäinen web-palvelun spesifikaatio ei mainitse tietoturvaa, vaan se on jätetty palvelun toteuttajan vastuulle. Tietoturva viestiperustaisessa viestinnässä perustuu autentikointiin, auktorisointiin, yhtenäisyyteen ja luottamuksellisuuteen. Autentikoinnilla tarkoitetaan viestin lähettäjän tunnistamista. Auktorisointi viittaa lähettäjän oikeuteen käyttää kyseistä palvelua. Yhtenäisyydellä tarkoitetaan sitä, että viestin täytyy tulla lähettäjältä kutsujalle samassa muodossa ilman muutoksia. Luottamuksellisuudella tarkoitetaan viestin sisällön paljastumista ainoastaan sen lähettäjälle sekä vastaanottajalle. [Brose, 2003]

4.1.1. Haasteita

Damiani ja muut [2002] nimeävät web-palvelujen tietoturvan suurimmiksi haasteiksi viestin lähettäjän autentikoinnin sekä tiedon luottamuksellisuuden takaamisen. Lähettäjällä tarkoitetaan nimenomaan viestin alkuperäistä lähettäjää, ei konetta, jonka kautta viesti on viimeksi kulkenut matkallaan viestin vastaanottajalle [Brose, 2003]. Molempiin ongelmiin on tarjolla ratkaisuja, jotka ovat käytössä normaalissa tietoliikenteessä, kuten palomuurin käyttö tai SSL. Mikään näistä ei kuitenkaan yksinään riitä takaamaan web-palvelun tietoturvaa [Damiani et al., 2002].

On tärkeää yhtenäistää web-palvelun ja SOAP-protokollan tietoturvaratkaisuja, jottei kehity monia erilaisia yhteensopimattomia ratkaisuja. Tällöin SOAP-protokollaan perustuvat ohjelmistot eivät olisi enää alusta- ja ohjelmointikieliriippumattomia, vaan ne täytyisi valita siten, että osapuolilla on käytössään sama tietoturvaratkaisu. [Damiani et al., 2002]

Web-palvelun rakenne muodostuu kolmesta kerroksesta: tietoliikennekerros, viestikerros ja sovelluskerros. Tietoliikennekerros huolehtii viestien välittämisestä. Viestikerroksen vastuulla on SOAP-muotoisten viestien rakenne. Sovelluskerros huolehtii viestien jatkotoimenpiteistä. Tietoliikennekerroksena on web-palvelun tapauksessa HTTP. Tietoturvaratkaisut voivat periaatteessa olla millä tahansa näistä kolmesta kerroksesta. Käytännössä kuitenkin SOAP-viestit halutaan todennäköisesti pitää riippumattomana siirtoprotokollasta, jolloin tietoturvaratkaisut tulisi sijoittaa viestikerrokseen. [Brose, 2003]

4.1.2. Liikenteen suodatus

Ennen kuin tietoliikennekerros voi siirtää viestejä sovellusohjelmalle, SOAP-muotoiset viestit täytyy suodattaa muusta tietoliikenteestä. Palomuuri voi toimia ensimmäisenä ylimääräisten viestien suodattajana. Tämän lisäksi viestejä voidaan suodattaa HTTP-otsikossa olevilla kentillä.

Palomuurin käyttö on yleinen suojautumistapa monissa yrityksissä. Palomuuri voidaan asettaa siten, että HTTP-protokollan kautta kulkeva liikenne läpäisee sen automaattisesti. SOAP-muotoisia viestejä voidaan lähettää mm HTTP-protokollan avulla, jolloin niitä varten ei tarvitse avata erillisiä portteja. Phanin ja muiden [2006] mukaan SOAP-protokolla suunniteltiinkin alun perin korvaamaan DCOM-, CORBA- ja Java RMI -teknologiat, jotka vaativat muutoksia palomuurin asetuksiin. Palomuurin avulla voidaan myös rajata sallitut yhteydet vain ennalta tunnettuihin IP-osoitteisiin. Web-palvelu ei kaikissa tapauksissa voisi hyödyntää tätä ominaisuutta, koska web-palvelun osapuolien välillä ei välttämättä ole suoraa yhteyttä, vaan viesti voi kulkea monen eri välittäjän läpi. [Damiani et al., 2002] Vaikka osapuolten ei tarvitsekaan avata erillisiä portteja palomuuriin web-palvelua varten, palomuurin lisäksi tarvitaan hienojakoisempi tapa huolehtia web-palvelun tietoturvaratkaisuista [Brose, 2003; Damiani et al., 2002]

Yksi tapa tunnistaa SOAP-viestiliikenne muista HTTP-protokollaa käyttävistä viesteistä on HTTP-otsikko. SOAP-viesteissä otsikko sisältää SOAPAction-kentän. Muissa viesteissä kenttää ei ole. Kentän sisältönä on viestin kohde URI-muodossa [W3C, 2003]. Koodikatkelmassa 1 kentän arvona on `http://peli.com/kysely#annaKirjain`. Periaatteessa palomuuri voisi käyttää kentän arvoa perusteena suodattaa tai sallia liikenne. Tällöin kuitenkin palvelu tulisi riippuvaiseksi palomuurin asetuksista. Toinen tapa tunnistaa XML-muotoinen HTTP-kutsu on HTTP-otsikon sisältämä ’content-Type’-kenttä. XML-muotoisessa viestissä kentän arvona on text/xml. Tämä tosin sisältää SOAP-viestien lisäksi myös muut XML-muotoiset viestit. Lisäksi SOAP-protokolla on alun perin suunniteltu siirtoprotokollasta riippumattomaksi, joten

käyttäjän tunnistusta tai liikenteen sallimista ei saisi jättää alla olevan protokollan, kuten HTTP tai TCP, varaan [Damiani et al., 2002].

4.1.3. Käyttäjien tunnistaminen ja valtuuttaminen

Palvelun käyttöä voidaan rajata vaatimalla käyttäjän autentikointia. Tällöin vain käyttäjät, jotka ovat etukäteen sopineet palveluntarjoajan kanssa käyttöoikeudesta, voivat käyttää tarjottuja palveluja. Autentikointi voidaan tehdä esimerkiksi käyttäjätunnuksen ja salasanan avulla kuten SAML-teknologiassa [Oasis, 2005a] tai julkisen avaimen avulla kuten SSL-teknologiassa [Netscape, 2006]. Käyttäjän tunnistuksen jälkeen käyttäjä täytyy vielä valtuuttaa kutsumaan viestissä olevaa palvelua. Käyttäjillä voi olla erilaisia oikeuksia, joiden avulla palveluntarjoaja päättelee, mitkä palveluista ovat asiakkaan käytössä. Valtuutus voidaan tehdä perustuen rooleihin sekä kutsuttavan palvelun kuvaukseen kuten Damianin ja muiden [2002] tiedonvalvontajärjestelmässä.

SSL-teknologian avulla voidaan huolehtia palvelimen ja asiakkaan autentikoinnista, tiedon salaamisesta sekä tiedon eheyden säilymisestä. Palvelin ja asiakas voidaan tunnistaa sertifikaattien ja julkisten avainten avulla, jotka on myöntänyt kolmas taho. [Netscape, 2006] Tällöin voidaan varmistua siitä, että viestin lähettäjä on varmasti yritys, joka viestissä on mainittu [Damiani et al., 2002]. SSL-teknologia on yleisesti käytössä verkkoliikenteessä, mutta se ei ole yksinään riittävä SOAP-liikenteen turvaamiseen, koska se ei hyödynnä XML-muotoisen liikenteen erityispiirteitä. SSL-teknologia toimii tiedonsiirrossa, joka tapahtuu kahden koneen välillä. Kun tieto kulkee lähettäjän ja vastaanottajan lisäksi muiden koneiden kautta, sen salattavuutta ei voida enää taata ainoastaan SSL-teknologian avulla. [Brose, 2003; Damiani et al., 2002]

Käyttäjän tunnistamiseen ja valtuuttamiseen XML-kieleen pohjautuvassa viestinnässä voidaan käyttää Oasiksen kehittämää SAML-standardia. SAML-viestit ovat XML-muotoisia, joten teknologian käyttö on alustariippumatonta. SAML-viestejä voidaan lähettää SOAP-viestin sisällä ja sisältävät käyttäjän tunnistukseen liittyvää informaatiota, kuten käyttäjätunnuksen ja salasanan. Tätä ominaisuutta voidaan käyttää yhdessä muiden teknologioiden, kuten WS-security -mallin [Oasis, 2006b], kanssa huolehtimaan web-palvelun turvallisuudesta. Ensimmäinen versio SAML-teknologiasta julkaistiin vuonna 2002 ja version 2.0 julkaistiin vuonna 2003. [Oasis, 2005a] Damianin ja muiden [2002] mukaan SAML-standardi ja XACML-standardi [Oasis, 2005b] luovat hyvän pohjan web-palvelujen tietoturvaratkaisuille, mutta eivät kata kaikkia avoimia asioita, kuten käytettävää arkkitehtuuria.

Oasiksen vuonna 2003 kehittämä XACML-standardi on sovellusriippumaton ja joustava kieli käyttöoikeuksien kuvaamiseen. Se sisältää mm. rooleihin ja aikaan perustuvan käyttöoikeuden käsittelyn. Käyttöoikeuden kohteena ovat erilaiset resurssit ja standardin avulla määritellään säännöt siitä, kenellä on oikeus käyttää kyseistä resurssia. XACML-standardi laajentaa SAML-protokollan sisältämää autentikointia, mutta sen on tarkoitus toimia myös muiden protokollien yhteydessä. [Oasis, 2005b]

Damiani ja muut [2002] esittelevät tiedonkäytön valvontajärjestelmän, joka hallitsee rooliin perustuvia käyttöoikeuksia. Auktorisointisuodattimeksi kutsuttu ohjelma ottaa vastaan lähettäjän SOAP-viestin ja tarkistaa, onko lähettäjällä oikeus kutsua viestin sisältämää palvelua. Tarkistuksen perusteella viesti voi lähteä varsinaiselle käsittelijälle muuttumattomana, muutettuna tai olla kokonaan lähtemättä. Jos viestin lähettäjällä on oikeus kutsua palvelua viestin sisältämässä muodossa, viesti lähtee eteenpäin muuttumattomana. Toisaalta, jos lähettäjällä ei ole ollenkaan oikeutta kutsua palvelua, sitä ei lähetetä eteenpäin palvelun toteuttavalle komponentille. Jos viestin lähettäjällä on oikeus käyttää palvelua, mutta ei niin laajassa muodossa kuin viestissä on määritelty, valvontaohjelma muokkaa viestiä suppeammaksi ja lähettää sen eteenpäin. Viestejä voidaan käsitellä samojen sääntöjen mukaan riippumatta siitä, otetaanko niitä vastaan vai lähetetäänkö niitä.

Jotta auktorisointisuodatinta voitaisiin käyttää viestin lähettäjille, täytyy asiakkaalle määritellä seuraavia ominaisuuksia: identiteetti, sijainti ja rooli. Identiteetti kuvaa palvelun kutsujaa ja sijainti kutsun lähtöpaikkaa. Rooli kertoo, mitä toimintoja asiakas voi kutsua. Rooleja voi olla nolla tai useampia. Toisin kuin yleensä, rooleja ei hallinnoi palvelun tarjoaja, vaan ne riippuvat viestikutsun mukana seuraavista sertifikaateista. Näin siksi, että verkkopalvelulla voi olla useita käyttäjiä, joista osa käyttää palvelua vain kerran. Jokaisen palvelunkäyttäjän identifioinnin sijaan voidaan suodatuksen perusteena käyttää roolia. Viestin lähettäjän ominaisuudet sisältyvät Damianin ja muiden [2002] määrittelemään SOAP-oletusotsikkoon. Pakollinen identiteetti-elementti sisältää käyttäjätunnuksen ja kryptatun salasanan. Sijainti-elementti sisältää verkko-osoitteen ilmaistuna joko symbolisesti tai numeerisesti. Rooli-elementti sisältää sertifikaatin, roolin tunnisteen, sertifikaatin avaimen, sertifikaatin haltijan ja sertifikaatin voimassaoloajan.

Varsinainen auktorisointi tapahtuu sääntöjen avulla. Sääntö sisältää subjektin, objektin ja arvon. Subjekti voi olla käyttäjän tunnus, rooli tai sijainti, jonka sallitaan tai ei sallita käyttää palvelua. Objekti identifioi, mitä palvelua kyseinen sääntö koskee. Arvo kertoo, sallitaanko kyseisen subjektin käyttää objektin kertomaa palvelua vai ei. Rakenteen

avulla voidaan ilmaista esimerkiksi seuraava sääntö: jälleenmyyjät voivat tallentaa tilauksen, jos ottavat yhteyttä tietystä aliverkosta.

Auktorisointisuodattimen käyttöönotto ei välttämättä vaadi muutoksia SOAP-kutsujen varsinaiseen käsittelijään, jos ne kommunikoivat HTTP-kutsujen avulla. Asiakkaan täytyy lisätä viestiinsä SOAP-otsikon tiedot, ja osalle asiakkaista ne voidaan generoida oletuksena samanmuotoisiksi. Salasanan kryptaus vaatii myös toimenpiteitä asiakkaalta. Sertifikaattien generointi vaatii toimenpiteitä palvelun tuottajalta.

4.1.4. Tiedon salaus ja yhtenäisyyden säilytys

Kun web-palvelun käyttäjä on autentikoitu ja auktorisoitu, täytyy vielä varmistaa, että lähetetty viesti on alkuperäinen. Tämä voidaan varmistaa tutkimalla viestin yhtenäisyyttä esimerkiksi digitaaliseen allekirjoitukseen perustuvien ratkaisujen avulla. Jos viestissä olevat tiedot ovat luottamuksellisia, kuten esimerkiksi luottokortin numero, täytyy huolehtia myös viestin osien salaamisesta. Tämä tehdään esimerkiksi kryptaamalla viestin sisältö tai osa siitä.

XML-muotoisen tiedon salaamiseen ja yhtenäisyyden säilyvyyteen voidaan käyttää esimerkiksi W3C:n kehittämiä standardeja kuten XML-allekirjoitusta [W3C, 2002b] sekä XML-kryptausta [W3C, 2002c]. Tekniikoita voi käyttää erikseen tai yhtä aikaa. Ratkaisut sopivat hyvin XML-tiedostojen salaukseen. Ne eivät kuitenkaan ole yksinään tarpeeksi kattava tietoturvaratkaisu, koska ne eivät hyödynnä XML-skeemoja. [Damiani et al., 2002] Lisäksi niiden avulla ei voida määritellä yksiselitteisesti, missä dokumentin kohdassa ja montako kertaa allekirjoituksella suojatut kentät esiintyvät. Tämä jättää dokumentin altiiksi XML-uudelleenkirjoitushyökkäykselle. [McIntosh and Austel, 2005]

XML-allekirjoitus on W3C:n ja IETF:n vuonna 1999 kehittämä standardi. Sen avulla voidaan varmentaa, että allekirjoituksen sisältämät kentät ovat kulkeneet muuttumattomana lähettäjältä vastaanottajalle. Allekirjoitus muodostetaan lisäämällä XML-muotoiseen viestiin uusi rakenne, joka sisältää salattavan tiedon. Uusi rakenne sisältää allekirjoituksen todentamista varten tarvittavat avaimet. [W3C, 2002b]

XML-kryptausta voidaan käyttää joko koko dokumentin tai sen osan salaamiseen. Sen avulla voidaan varmistaa, että ulkopuoliset eivät voi lukea dokumentin salattua sisältöä. Saman dokumentin sisällä voi olla useita eri salausavaimella avattavia tietoja. Tällöin saman dokumentin voi lähettää usealle asiakkaalle, ja he voivat lukea dokumentista vain ne osat, joihin heillä on avain. [Damiani et al., 2002] Tästä voi olla hyötyä esimerkiksi silloin, kun käyttäjillä on eri rooleja, kuten jälleenmyyjä tai tilaaja, ja heille

halutaan näyttää eri määrä tietoja. Esimerkiksi tilaaja ja jälleenmyyjä voivat nähdä erilaisia tietoja tuotteen hinnasta ja sen muodostumisesta. Kryptattu tieto muodostetaan korvaamalla XML-viestin salattava tieto kryptatulla tiedolla ja sen avaamiseen tarvittavalla avaimella [Damiani et al., 2002].

Web-palvelun tietoturvaratkaisuissa täytyy varautua monenlaisiin riskeihin [Oasis, [2006b]. Käyttäjän tunnistaminen on tärkeää, jotta vältetään väärinkäytöksiä. Viestin yhtenäisyyden tarkistaminen suojaa puolestaan siltä, että web-palvelu ei prosessoi viestejä, joiden sisältö on muuttunut matkalla, jolloin viestin alkuperäinen lähettäjä ei saisi odottamaansa vastausta. Nämä riskit sisältyvät XML-uudelleenkirjoitushyökkäykseen, johon lasketaan kuuluvaksi mm. viestin manipulointi, sieppaaminen ja estäminen. Web-palvelut voivat joutua myös palvelunestohyökkäyksen kohteeksi, jolloin XML-jäsentäjää yritetään lamauttaa ylikuormittamalla. Hyökkäykset ovat mahdollisia, jos luotetaan vain yhteen tietoturvaratkaisuun. [Bhargavan et al., 2005a]

Esimerkiksi XML-allekirjoitus yksinään ei riitä estämään tietoturvahyökkäyksiä, koska se ottaa kantaa vain allekirjoitettuihin kenttiin. Kun viesti sisältää XML-allekirjoituksen, ei voida olla täysin varmoja, että suojatut kentät esiintyvät viestissä vain kerran. Hyökkääjä on voinut lisätä uusia kenttiä, jotka estävät allekirjoituksella suojattujen kenttien lukemisen ja tämän jälkeen kirjoittaa vaaditut kentät uudelleen uusilla arvoilla. [Rahaman et al., 2006] SOAP-viestin haitallinen muokkaus on usein mahdollista viestin moniselitteisen rakenteen vuoksi. Rakenne voi mahdollistaa esimerkiksi saman elementin sijoittamisen viestiin useammin kuin kerran eri paikkaan eri arvoilla tai uusien kenttien lisäämisen ilman että viestin prosessointi keskeytyy. [McIntosh and Austel, 2005] Kun erilaisia tietoturvaratkaisuja yhdistetään, pystytään tällaisia turva-aukkoja vähentämään.

WS-perheen malleilla voidaan ratkaista osa näistä ongelmista määrittelemällä tarkemmin, miten kenttien tulee sijoittua XML-viestissä sekä laatimalla säännöt kenttien lukumääriin [Bhargavan et al., 2005a]. Rahaman ja muut [2006] ehdottavat ratkaisuksi SoapAccount-konseptia, jonka avulla pidetään kirjaa elementtien määrästä ja viestin rakenteesta. Tällöin uusien kenttien lisääminen aiheuttaa virheilmoituksen. Rakennetta kuvaavan osan muuttaminen puolestaan aiheuttaa sen, että allekirjoitus ei ole enää pätevä.

Web-palvelu käyttää viestin siirtoon HTTP-protokollaa, jolloin viesti ei siirry suoraan lähettäjältä vastaanottajalle vaan kulkee monien välikäsien kautta. Jos tiedon yhtenäisyyden ja luotettavuuden tarkistus jätettäisiin siirtoprotokollaan perustuvan tietoturvaratkaisun, kuten SSL:n, vastuulle, kaikkien välikäsien tulisi myös olla

luotettavia, koska näillä on mahdollisuus käsitellä viestiä. Web-palvelun SOAP-muotoinen viestikerros ei itsessään sisällä tietoturvaratkaisuja. Tämän vuoksi Brose [2003] toteaakin, että tietoturvaratkaisu tulee sisällyttää joko sovelluskerrokselle tai erilliselle välittäjäkerrokselle, joka sijoittuu sovelluksen ja viestikerroksen väliin. Tällöin pysytään takaamaan viestin salattavuus lähettäjältä vastaanottajalle, vaikka viesti kulkisi myös välikäsien kautta.

Web-palvelun viestikerrokselle sijoittuville tietoturvaratkaisuille on kehitetty monia malleja, kuten WS-Security [Oasis, 2006b], WS-Policy [W3C, 2006b], WS-Trust [IBM et al., 2005] ja WS-SecurityPolicy [Oasis, 2005c]. Ne tarjoavat periaatteita ja ohjeita web-palvelujen kuvaamiseen, mutta eivät tarjoa valmista toteutusta. Tämän vuoksi ne ovat melko joustavia. Tosin joustavuuden parantaminen aiheuttaa usein tehokkuuden vähenemistä, josta tarkemmin seuraavassa luvussa [Bhargavan et al., 2005b; Damiani et al., 2002].

Mallien on tarkoitus parantaa esimerkiksi palvelujen laatua, luotettavuutta ja salattavuutta [W3C, 2006b]. WS-perheen mallit on määritelty XML-kielellä, ja niiden avulla voidaan muodostaa puumainen sisäkkäinen kuvaus säännöistä, joiden avulla tarkistetaan viestin sisältämiä omaisuuksia. [Govindaraju et al., 2000] Käytettyä mallia voidaan kutsua palvelukohtaiseksi käytännöksi, koska se ohjaa palvelun toimintaa. IBM [2002b] määrittelee käytännön (policy) siten, että se tarkoittaa sääntöjoukkoa, joka kuvaa yhtä palvelua. Sääntö (policy statement) puolestaan on väitteiden joukko ja väite (policy assertion) käsittelee yhtä vaatimusta, joka voi olla tosi tai epätosi.

Oasiksen kehittämä WS-Security-standardi tukee SOAP-viestien yhtenäisyyttä ja luottamuksellisuutta sekä SOAP-viestin osien salaamista. Standardissa esitettävät mallit eivät yksinään tai yhdessä tarjoa täysin kattavaa kokonaisratkaisua, vaan ne on tarkoitettu käytettäväksi muiden WS-perheen mallien ja salausteknologioiden, kuten SSL:n, tukena. WS-Security-standardi määrittelee mm. XML-securityheader-rakenteen. Sen sisältämän tiedon avulla viestin vastaanottaja voi päätellä, miten viestin aitous voidaan tarkistaa. Rakenne voi sisältää esimerkiksi SAML-standardin mukaisen viestin, sertifikaatin, tai avaimen XML-allekirjoitukseen tai XML-kryptaukseen. [Oasis, 2006b]

WS-Policy on W3C:n kehittämä XML-kieleen pohjautuva standardi, jossa esitellään web-palvelua kuvaavien sääntöjen malli. Säännöt kuvaavat, mitä elementtejä palvelu vaatii saapuvalta viestiltä ja mitä se kykenee ymmärtämään. Kentät eivät liity viestin varsinaiseen asiasisältöön, jota puolestaan määrittelee XML-skeema, vaan niiden avulla voidaan kuvata viestin yleisempiä ominaisuuksia kuten yhtenäisyys. Malli koostuu säännöistä, joiden sisältämät väitteet voidaan ryhmitellä ’All’ tai ’ExactlyOne’ -

komentojen avulla. Edellisessä kaikkien ryhmän sisällä olevien väitteiden tulee toteutua ja jälkimmäisessä vain yksi väitteistä voi ja sen tulee toteutua. Koodikatkelma 3 on esimerkki käytännöstä, jonka mukaan viestissä voi ja tulee olla toinen mainituista algoritmeista. [W3C, 2006b]

```
<wsp:Policy …>
   <wsp:ExactlyOne>
     <sp:Basic256Rsa15 />
     <sp:TripleDesRsa15 />
   </wsp:ExactlyOne>
 </wsp:Policy>
```

Koodikatkelma 3. WS-Policy esimerkki

WS-SecurityPolicy on Oasiksen kehittämä malli, joka noudattaa WS-Policyn mallia. Se keskittyy erityisesti ominaisuuksiin, joilla voidaan taata viestin luotettavuus ja yhtenäisyys. Sääntöjen avulla voidaan rajata esimerkiksi viestin sisältämien allekirjoitusten määrää ja laatua. WS-SecurityPolicy-rakenteen avulla voidaan tarkistaa esimerkiksi, sisältääkö viestin WS-Security-standardin mukainen securityHeader-elementti vaaditut tiedot. Elementti voi noudattaa WS-Security-standardia ilman, että se välttämättä täyttää kaikkia WS-SecurityPolicy-mallin mukaisia sääntöjä. Viestit, jotka eivät täytä mallin mukaisia sääntöjä esimerkiksi siksi, että kolmas osapuoli on muuttanut viestiä, eivät läpäise suodatusta. [Bhargavan et al., 2005b]

WS-Policy-mallin mukaan määritelty käytäntö voidaan toteuttaa osana ohjelmakoodia tai erillisenä asetustiedostona. Sopivin tapa riippuu valitusta SOAP-kirjastosta. Kun käytetään erillistä tiedostoa, turvallisuuteen liittyvät säännöt voidaan selkeästi erottaa omaksi kokonaisuudekseen muista viestiin liittyvistä piirteistä. Tällöin esimerkiksi turvallisuuskäytännön ylläpito ja sen katselmointi helpottuvat. WS-Policy-kielen avulla voidaan määritellä omia rakenteita, joiden avulla tarkistetaan, esiintyykö viestissä vaaditut ja ainoastaan vaaditut turvallisuuteen liittyvät elementit. Voi kuitenkin olla hankalaa saada rakenteista täysin kattavia. WS-Policy-mallin heikkoutena on sen alttius uudelleenkirjoitushyökkäyksille. Tällöin hyökkääjä voi esimerkiksi sijoittaa jonkin turvallisuuteen liittyvistä kentistä viestin eri kohtaan, jolloin viestin tulkinta muuttuu alkuperäisestä. Tätä voidaan käyttää välineenä palvelunestohyökkäyksessä tai viestin perillemenon estämisessä. Toinen WS-SecurityPolicy-mallin heikkous on rajoittuminen matalan tason tarkistamiseen, kuten WS-Security-mallin mukaiseen SecurityHeader-kokonaisuuteen. Olisi hyödyllistä, että tarkastus voitaisiin suorittaa tietyn mallin sijaan ohjelman korkeamman tason ominaisuuksille, kuten autentikoinnille tai salattavuudelle. [Bhargavan et al., 2005b]

Yksi WS-SecurityPolicy-mallin toteuttavista SOAP-kirjastoista on Microsoftin web-palveluja tukeva kirjasto ’Web Services Enhancements for Microsoft .NET’ (WSE). Se toteuttaa WS-Security-mallin ja mahdollistaa sen tarkastamisen WS-SecurityPolicy-sääntöjen avulla. [Bhargavan et al., 2005a]

4.1.5. Haasteisiin vastaaminen

Web-palvelun käyttö julkisessa verkossa tuo mukanaan joukon haasteita, johon valitun tietoturvaratkaisun tulee vastata. Ratkaisun tulee pystyä vaikeuttamaan tai estämään palvelimen tahallinen kuormitus sekä tunnistamaan viestien uudelleenkirjoitus ja uudelleenohjaus. Koska Web-palvelu käyttää XML-muotoisia SOAP-viestejä, sen tulee varautua myös XML-ylikirjoitushyökkäyksiin. [Bhargavan et al., 2005a].

Web-palvelun luonteesta riippuen sitä voivat käyttää ennalta tunnetut asiakkaat, jotka asioivat usein, tai ennalta tuntemattomat asiakkaat, jotka eivät välttämättä käytä palvelua yhtä kertaa enempää. Edellistä tapaa käytetään yritysten välisessä kaupankäynnissä. Jälkimmäinen on tyypillistä web-palvelulle, jonka asiakkaana voi olla yksittäisiä ihmisiä [Damiani et al., 2002]. Tämän piirteen hallinta tuo haasteita käyttäjien auktorisoinnille. Jos moni käyttäjistä on ennalta tuntematon ja käyttää palvelua vain harvoin, palveluntarjoaja voisi kuluttaa tarpeettomia resursseja tällaisten käyttäjien käyttöoikeuksien hallintaan. Käyttäjän tunnistus ja oikeuksien hallinta voidaan ratkaista esimerkiksi käyttämällä erilaisia käyttäjärooleja ja määrittämällä rooli kolmannen osapuolen myöntämällä sertifikaatilla [Netscape, 2006].

Kun viesti kulkee julkisen verkon läpi, lähettäjän ja vastaanottajan välillä on useita välikäsiä. Tämä tuo osaltaan lisää haasteita sopivalle tietoturvaratkaisulle. Sen sijaan, että käytetään ratkaisua, joka takaa viestin luotettavuuden ja yhtenäisyyden kahden koneen välillä, tulee taata viestin yhtenäisyys lähettäjältä välikäsien kautta vastaanottajalle. [Brose, 2003; Damiani et al., 2002] Tämä ominaisuus rajoittaa tietoturvaratkaisun sijoittelua web-palvelun arkkitehtuurin kerroksille.

Edellä mainittiin julkisen verkon tuomia haasteita tietoturvalle. Ratkaisulta odotetaan turvallisuuden takaamisen lisäksi usein myös tehokkuutta, joustavuutta ja riippumattomuutta. Nämä ominaisuudet ovat useissa ratkaisuissa vastakkaisia; esimerkiksi mitä joustavampi ratkaisu on, sitä tehottomampi se saattaa olla. Ominaisuuksien tärkeysjärjestys riippuu palvelun luonteesta. Tehokkuuden huononnuminen voi vaikuttaa esimerkiksi siihen, kuinka nopeasti asiakas saa vastauksen viestiinsä. Riippumattomuudella tarkoitetaan sitä, että tietoturvaratkaisua ei ole sidottu tiettyyn ohjelmointikieleen tai palvelinalustaan. Web-palvelu on alustariippumaton, ja tietoturvaratkaisujen tavoitteena on säilyttää tämä ominaisuus. [Damiani et al., 2002].

Joustavuus vaikuttaa muutoksiin, joita ratkaisun vuoksi joudutaan tekemään esimerkiksi palomuuriin. Molemmat ominaisuudet vaikuttavat siihen, kuinka nopeasti palvelu voidaan ottaa käyttöön uudessa ympäristössä. Viestiperustaisten järjestelmien tietoturva voidaan jakaa neljään tärkeään osa-alueeseen: autentikointiin, auktorisointiin, yhtenäisyyteen ja luottamuksellisuuteen [Brose, 2003].

Web-palvelussa voidaan käyttää perinteisiä verkkoliikenteessä käytössä olevia tietoturvaratkaisuja kuten SSH ja palomuuri. Lisäksi voidaan käyttää XML-kieleen perustuvia ratkaisuja, kuten XML-allekirjoitus ja XML-kryptaus, koska SOAP-viestit ovat myös XML-muotoisia. Tällöin ei lisättäisi web-palvelun riippuvuutta erillisestä teknologiasta. Perinteiset ja XML-pohjaiset tietoturvaratkaisut voivat toimia osana web-palvelulle kehitettävää tietoturvaratkaisua, mutta tämän lisäksi tarvitaan ratkaisu, joka ottaa huomioon web-palvelun erityispiirteet kuten XML-skeemat ja SOAP-teknologian sekä kattaa eri turvallisuusnäkökohdat [Brose, 2003; Damiani et al., 2002]. Näitä erityisesti web-palvelulle sopivia tietoturvaratkaisuja on kehitetty 2000-luvulla, esimerkiksi SAML-teknologia ja WS-perheen mallit, joita voidaan käyttää myös yhdessä [Oasis, 2006b].

Web-palvelun alustariippumattomuus ja joustavuus voidaan säilyttää siten, että tietoturvaratkaisua ei sidota tiettyyn ohjelmointikieleen, alustaan tai asiakaskohtaisesti asetettavaan ratkaisuun. Palomuuri voidaan käyttää yleisen verkkoliikenteen suodattamiseen ilman suurempia asiakaskohtaisia muutoksia. Jos sen avulla haluttaisiin suodattaa SOAP-viestejä tai vielä hienojakoisemmin SOAP-viestin sisältämiä palvelukutsuja, palomuuria tulisi räätälöidä ratkaisun ehdoilla [Damiani et al., 2002]. Normaalisti jos palomuuri on asetettu sallimaan HTTP-liikenne, se sallii automaattisesti myös SOAP-viestit, koska ne voidaan sisällyttää HTTP-kutsuun. SOAP-viestien lajittelu jo palomuurin avulla saattaisi nopeuttaa viestien käsittelyä, mutta toisaalta se voisi heikentää palomuurin tarjoamaa tietoturvaa ja näin lisätä tietoturvariskejä. Lisäksi ratkaisu voisi vähentää palvelun siirrettävyyttä eri ympäristöihin, koska se olisi riippuvainen palomuurin kyvystä erotella SOAP-viestit muusta HTTP-liikenteestä.

Sopiva tietoturvaratkaisu voi riippua siitä, minkälainen asiakaskunta web-palvelulla on. Jos asiakkaat ovat suurimmaksi osaksi ennalta tunnettuja, tietoturvaratkaisun osana voi olla SSH-teknologia, jossa asiakas tunnistetaan julkisen avaimen avulla tai kolmannen osapuolen tarjoaman sertifikaatin avulla. Jos ei tiedetä ennalta, tuleeko asiakaskunta muuttumaan, ratkaisu ei saisi rajoittaa sitä, tuleeko palvelupyyntö tuntemattomasta vai ennalta tunnetusta osoitteesta. SSH-teknologia ei yksinään riitä web-palvelun kattavaksi tietoturvaratkaisuksi, koska se ei huomioi palvelun erityispiirteitä [Damiani et al., 2002].

SAML-teknologia ja WS-perheen mallit ovat XML-muotoisia. Niiden käyttö osana tietoturvaratkaisua ei rajoita web-palvelua tietyn teknologian käyttöön, koska myös SOAP-viestit ovat XML-muotoisia. Tämän vuoksi niiden käyttö on myös alustariippumatonta ja niitä voidaan lähettää SOAP-viestin osana. [Oasis, 2006b] SAML-teknologia ja WS-perheen mallit antavat keinoja käyttäjän tunnistamiseen ja valtuuttamiseen. Käyttäjän valtuuttaminen voidaan hoitaa esimerkiksi rooliperustaisesti, kuten Damianin ja muiden [2002] esittelemässä tiedonkäytön valvontajärjestelmässä. Tällöin käyttäjillä voi olla erilaisia rooleja, joita saadaan käyttämällä kolmannen osapuolen tarjoamia sertifikaatteja. Roolien avulla päätellään, onko käyttäjällä oikeus kutsumaansa palveluun. Ratkaisu ei ota kantaa, onko palvelun kutsuja etukäteen tuttu palveluntarjoajalle vai käyttääkö tämä palvelua ensimmäistä kertaa. Tällöin ratkaisu ei ole sidottu palvelun käyttäjien luonteeseen, joten sitä voidaan käyttää erityyppisissä palveluissa.

XML-allekirjoitus ja XML-kryptaus on toteutettu nimensä mukaan XML-teknologialla, joten niitä voidaan käyttää osana web-palvelun tietoturvaratkaisua ilman, että ne vähentävät palvelun joustavuutta. Niitä voidaan käyttää esimerkiksi salaamaan käyttäjän salasana. Ne eivät kuitenkaan tuo riittävän hienojakoista suojaa web-palveluille, koska ne eivät hyödynnä palvelun erityispiirteitä kuten XML-skeemaa [Damiani et al., 2002].

Edellä on esitelty monia yksittäisiä teknologioita, joiden avulla voidaan huolehtia web-palvelun tietoturvaratkaisun eri osista, kuten autentikoinnista tai auktorisoinnista. WS-perheen mallien avulla voidaan kuvata yhtenäisesti, mitä tietoturvaan liittyviä asioita palvelukutsuun tulee sisällyttää, jotta viesti voidaan tunnistaa aidoksi. Kuvatut tietoturvaominaisuudet kuten tietyn salausalgoritmin käyttö voidaan toteuttaa vapaavalintaisella teknologialla, joten WS-perheen mallit eivät sido toteutusteknologiaa. Mallien käyttö lisää joustavuutta, koska ratkaisu voidaan kuvata yhtenäisellä tavalla. Toisaalta ratkaisu vähentää myös web-palvelun tehokkuutta [Bhargavan et al., 2005b; Damiani et al., 2002].

Joustavienkin tietoturvaratkaisujen tekninen toteutus vaikuttaa viime kädessä siihen, ovatko ne riittävät turvallisia. Bhargavan ja muut [2005b] muistuttaa, että jos WS-perheen mallit ja tarkastussäännöt päätetään varastoida muutamaan tiedostoon, sitä voidaan käyttää tietoturvahyökkäyksen lähtökohtana. Lisäksi teknisen toteutuksen tulisi olla sellainen, että ratkaisuun tehdyt haitalliset muutokset huomattaisiin helposti. Ratkaisu pitäisi siis olla sellainen, että se voidaan helposti katselmoida toteuttajien kesken tai tarkastaa erillisen ohjelman avulla. WS-SecurityPolicy -mallin heikkoudeksi mainitaan myös se, että sen avulla voidaan suorittaa vain matalan tason tarkistusta, esimerkiksi esiintyykö tietty allekirjoitus palvelupyynnössä vai ei. Tämän sijaan olisi

tarvetta korkeamman tason tarkastusmekanismeille, kuten onko viestin tärkeät osat riittävästi salattu vai ei. [Bhargavan et al., 2005b] Lisäksi käytetyn rakenteen tulisi olla riittävän yksiselitteinen kenttien rakenteen, lukumäärän ja sijainnin suhteen, jottei väärinkäytöksille jää mahdollisuuksia.

Tietoturvaratkaisun sijoittelu vaikuttaa myös osaltaan ratkaisun joustavuuteen ja riippumattomuuteen. Alkuperäinen web-palvelun spesifikaatio ei sisällä tietoturvaa, joten sen sijoittelu on jätetty web-palvelun toteuttajan vastuulle [Damiani et al,. 2002]. Web-palvelun arkkitehtuuri jaetaan usein tietoliikenne-, viesti- ja sovelluskerrokseen. Lisäksi joissakin tapauksissa arkkitehtuurissa voi esiintyä välittäjäkerros, joka sijoittuu sovellus- ja viestikerroksen väliin. Tietoturvan kannalta oleellisia asioita ei tulisi sijoittaa ainoastaan tietoliikennekerrokselle, koska tällöin voitaisiin turvata viestin salattavuus ainoastaan kahden koneen välillä, muttei välttämättä jokaisen välikäden osalta, joita voi olla monia viestin kulkiessa lähettäjältä vastaanottajalle. [Brose, 2003] Kattavuuden lisäksi ratkaisun sijoittelussa tulee huomioida, että se ei rajoittaisi tarpeettomasti web-palvelun joustavuutta tai siirrettävyyttä.

Muita tietoturvaratkaisun valintaan vaikuttavia seikkoja voivat olla esimerkiksi ratkaisun hinta, käyttöönoton nopeus ja ratkaisun pitkäaikaisuus sekä tuettavuus. Monien edellä mainittujen teknologioiden dokumentaatiot ovat vapaasti verkossa saatavilla. Tämän lisäksi palveluntarjoaja tarvitsee kirjaston, joka hyödyntää näitä teknologioita. Nämä eivät välttämättä ole vapaasti saatavilla, kuten esimerkiksi Microsoftin WSE-kirjasto. Ratkaisun valintaan voi vaikuttaa myös se, kuinka paljon sen käyttö aiheuttaa vaivaa asiakkaalle. Jos se vaatii paljon toimenpiteitä asiakkailta, he voivat kääntyä toisen palveluntarjoajan puoleen.

4.2. Web-palvelun tehokkuus

Kun valitaan sopivaa ratkaisua tiedon siirtoon julkisessa verkossa, ratkaisun riittävä tehokkuus on useasti turvallisuuden ohella tärkeä valintakriteeri. Tehokkuus määrittelee mm. miten nopeasti palvelun käyttäjä saa vastauksen pyyntöönsä, kuinka moneen asiakaspyyntöön palvelu pystyy vastaamaan, millainen verkko täytyy olla käytössä ja mitä palvelimelta vaaditaan. Ihannetapaus olisi todennäköisesti nopea, virheetön ja pienikokoinen vaste mahdollisimman pienellä käsittelytarpeella. Tehokkuuteen kiinnitetään huomiota myös silloin, kun ratkaisua valitessa halutaan varautua tulevaisuuden haasteisiin, esimerkiksi kasvaviin asiakasmääriin tai käsiteltävien viestien monimutkaistumiseen. Ratkaisun tulisi olla sovellettavissa muuttuneeseen käyttö-ympäristöön ilman mittavia kustannuksia.

4.2.1. Haasteita

SOAP-teknologian hyviksi puoliksi mainitaan usein rakenteen joustavuus, helppolukuisuus sekä riippumattomuus toteutuskielestä [Abu-Ghazaleh and Lewis, 2005; Damiani et al., 2001]. Mainitut ominaisuudet johtuvat siitä, että viestien siirtoformaattina käytetään tekstimuotoista XML-kieltä [Takase et al., 2005]. ASCII-formaatin käyttö johtaa heikentyneeseen tehokkuuteen, kun SOAP-teknologian käyttöä verrataan kilpaileviin teknologioihin, kuten Java-RMI- ja CORBA, jotka käyttävät siirtoformaattina binäärimuotoa. [Abu-Ghazaleh and Lewis, 2005; Gray, 2004].

Teknologioiden tehokkuutta verkonkulutuksen näkökulmasta voidaan vertailla esimerkiksi mittaamalla verkossa liikkuvan viestin kokoa sekä tarvittavien verkkokutsujen määrää. Tehokkuuteen vaikuttaa myös valitun viestiformaatin vaatima käsittely sekä viestin lähettäjän että viestin vastaanottajan toimesta. Sekä verkon kulutus että viestien käsittely korostuvat entisestään ympäristöissä, joissa ne ovat rajoittavina tekijöinä. Näitä ympäristöjä ovat esimerkiksi langattomat päätelaitteet sekä tieteellinen laskenta. Langattomat päätelaitteet ovat haastavia rajoitetun virtalähteen ja verkkotyyppinsä vuoksi [Kangasharju et al., 2003]. Tieteellinen laskenta tuo mukanaan monimutkaisien ja suurten viestien vastaanoton ja prosessoinnin [Govindaraju et al., 2000]. Koska SOAP-teknologia vaati ASCII-muotoisten viestien vuoksi paljon käsittelyä sekä tehokkaan verkon viestien lähettämiseen, se toimii parhaiten ympäristössä, jossa prosessointiteho tai verkonkulutus eivät ole suurin rajoittava tekijä.

SOAP-teknologian tehokkuuden optimoinnissa on keskitytty käsiteltävien tavujen määrän sekä tavukohtaisen käsittelyn vähentämiseen [Kostoulas et al., 2006]. Käsiteltävien tavujen määrää voidaan vähentää esimerkiksi pienentämällä viestin kokoa tai prosessoimalla vain osa vastaanotetusta viestistä. Tiedostokokoa voidaan pienentää pakkaamalla tiedosto [Kangasharjun et al., 2003] tai korvaamalla osia ASCII-muotoisesta XML-viestistä binäärikoodeilla [W3C, 1999]. Riippuen käytetystä pakkausalgoritmista tai binäärikoodauksesta viestin käsittelyn määrä saattaa kasvaa [Bayardo et al., 2004], mutta käsiteltävien tavujen määrä on pienempi. Riippuen viestiliikenteen luonteesta käsiteltävien tavujen määrää voidaan ehkä myös vähentää vertaamalla lähtevää tai saapuvaa viestiä edeltävään ja käsittelemällä vain eriävät osat [Abu-Ghazaleh and Lewis, 2005].

Tavuihin kohdistuvaa käsittelyä voidaan vähentää kiinnittämällä huomiota tyyppimuunnosten vähentämiseen. XML-viestin käsittelyn aikana voi esiintyä käsittelijästä riippuen muunnoksia UTF-8- ja UTF-16 -muotojen välillä sekä muunnoksia string-tietotyypistä int-tietotyyppiin ja takaisin. Muunnoksia tarvitaan, jos ohjelman ulkopuolinen ja sisäi-

nen koodaus eroavat toisistaan ja jos käsiteltävät arvot tarkistetaan skeemaa vasten. [Takase et al., 2005] Käsittelijää valitessa tai toteuttaessa tulee kiinnittää huomiota moninkertaisten tyyppimuunnosten eliminointiin, koska ne tuovat ylimääräistä prosessointia.

SOAP-teknologian käyttöä on tutkittu kiinteän verkon lisäksi myös langattomilla päätelaitteilla koska se tarjoaa standardin erilaisten päätelaitteiden välille [Phan et al., 2006]. XML-kielen ja SOAP-viestien tehokas käsittely korostuu entisestään langattomille päätelaitteille tarkoitetuissa sovelluksissa, koska akkukäyttöisissä päätelaitteissa on pienempi prosessointiteho ja verkon kaistanleveys on kapeampi. Myös yhteyden suuren latenssin vaikutus näkyy selvemmin langattomilla päätelaitteilla kuin kiinteässä verkossa. [Kangasharju et al., 2003]. Kun kaistanleveys on kapeampi, verkon läpi voi lähettää yhdellä kertaa vähemmän tietoa, jolloin lähetettävien tiedostojen koko tulisi saada mahdollisimman pieneksi. Päätelaitteet vaativat pientä prosessointitehoa, jotta ne voisivat toimia pidempään samalla virtalähteellä. Tällöin vastaanotettavat tiedostot eivät saisi vaati liian monimutkaista käsittelyä. Myös muistin määrä laitteissa on rajoitettu, joten käsittely ei myöskään saisi vaatia rajattomasti välimuistia. [Phan et al., 2006] Korkea latenssi kertoo siitä, että pakettien perille meno kestää kauan. Jos palvelun käyttäminen vaatii monta yhteydenottokertaa ja valittu verkkoprotokolla vaatii moniportaisen kättelyn, vaikutus näkyy erityisesti langattomilla päätelaitteilla. Langattomassa ympäristössä korostuu entisestään tehokkuuden piirteet, jotka XML-kieleen perustuvan SOAP-teknologian käytössä tulee huomioida: tiedoston koko, tarvittava verkkoliikenteen määrä ja prosessointitarve.

SOAP-teknologian suurimpia tehokkuutta heikentäviä tekijöitä ovat XML-viestien ASCII-muoto ja siitä johtuva prosessointi [Jun et al., 2006]. Verkkoliikennettä tarvitaan enemmän XML-viestien kuin tietosisällöltään vastaavien kooltaan pienempien binäärimuotoisten viestien lähettämiseen [Govindaraju et al., 2000]. Lisäksi verkkolatenssia lisää web-palvelun käyttämä TCP/IP-protokolla, joka vaatii kolmiportaisen kättelyn [Ciancarini et al., 2002]. Seuraavassa käsitellään tarkemmin erilaisia ratkaisumalleja sekä vaihtoehtoisia toteutustapoja web-palvelun ja SOAP-teknologian tehokkuusnäkökulmiin.

4.2.2. Verkkolatenssi

Ciancarini ja muiden [2002] mukaan yksi SOAP-teknologian hitauteen vaikuttavista tekijöistä on se, että SOAP-teknologiaan pohjautuva ratkaisu on yleensä rakennettu HTTP-protokollan päälle. Tämä käyttää puolestaan TCP-protokollaan, joka käyttää monia järjestelmä- ja verkkokutsuja viestin lähettämiseen. Tällöin tarvitaan tavallista enemmän verkkoliikennettä ja prosessointia jokaisen viestin välittämiseen. Esimerkkejä syistä korkeaan verkkoliikenteen määrään ovat HTTP-protokollan käyttämä kolmiportainen kättely sekä tarvittavien yhteyksien määrä. Kun HTTP-protokollan

päälle toteutettu SOAP-muotoinen viesti lähetetään, käytetään yleensä RPC-mallin mukaista kättelyä. Tällöin ennen kuin viesti voidaan lähettää, yhteys tarvitsee avata kolmiportaisen kättelyn avulla [Kangasharju et al., 2003]. HTTP-protokollan 1.0 versiossa on lisäksi toinen ominaisuus, joka lisää verkon kulutusta. Kyseinen ominaisuus sallii käyttää kättelyn avulla avattua yhteyttä vain yhden viestin lähettämiseen. Joten jokaista viestiä varten täytyy avata ja sulkea uusi yhteys. Tämä ominaisuus ei esiinny enää HTTP-protokollan versiossa 1.1. [Kangasharju et al., 2003]

Phan ja muut [2006] ovat vertailleet SOAP-teknologian käyttöä langattomassa ympäristössä HTTP-protokollan lisäksi myös TCP- ja UDP-protokollien päälle rakennetussa ratkaisussa. UDP-protokollan [Postel, 1980] tehokkuus perustuu siihen, että se ei varmista pakettien perillemenoa ja käyttää näin vähemmän verkkoa ja pitää pakettikoon pienempänä [Phan et al., 2006]. Combs ja muut [2004] ovat ehdottaneet standardia UDP-protokollan käyttöön SOAP-teknologian kanssa. Protokollien vertailussa tuli ilmi, että TCP- ja UDP-protokollien avulla lähetetyt viestit ovat melko samankokoisia. HTTP-protokollan avulla lähetty viesti on puolestaan noin 200 tavua suurempi sen sisältämien otsikkotietojen vuoksi. Sekä HTTP- että TCP-protokolla käyttävät kolmiportaista kättelyä, joka tarvitsee noin 200 tavua. UDP-protokolla käyttää samaan 8 tavua, koska se ei muodosta pysyvää yhteyttä. Myös vasteajoissa UDP-protokolla on nopein, koska se ei vaadi viestien koodausta. Koska UDP-protokolla ei varmenna pakettien perille menoa, pakettien katoamisen todennäköisyys suurenee viestin koon ja asiakkaiden määrän suurentuessa. Tämän vuoksi sitä suositellaan vain lyhyiden viestien käyttöön ja järjestelmiin, joissa viestien tehokas liikkuminen on tärkeämpi vaatimus kuin perille menon luotettavuus.

Myös valitulla SOAP-kirjastolla on vaikutusta verkkoliikenteen määrään. Davis ja Parashar [2002] vertailevat SOAP-protokollan verkon kuormitusta eri toteutuksilla. Eri SOAP-kirjastojen välillä erot ovat paikoin yli kaksinkertaiset. Erot selittyvät sillä, että eri kirjastoissa lähetettävä viesti voidaan jakaa yhteen tai useampaan pakettiin. Apache SOAP [Apache, 2003] jakaa lähetettävän HTTP-viestin header- ja body-osioihin. Mitä useampi paketti lähetetään, sitä kauemmin viestin vastaanotto kestää. Pakettien lähetykseen kuluvan ajan lisäksi viivettä lisää Nagle-algoritmi sekä TCP-protokollan viivästetty ACK-algoritmi. Näiden vuoksi jälkimmäinen paketti ei lähde heti vaan pienen viiveen kuluttua. Algoritmien tarkoitus on kerätä lähetettävästä paketista mahdollisimman suuri, jotta vältetään useita paketteja. Niiden vaikutus voi näkyä asiakkaan tai palvelimen puolella riippuen siitä, kummalla puolella ollaan lähettämässä kahta peräkkäistä viestiä. Verrattuna Java RMI ja CORBA-teknologioihin SOAP-kirjastojen latenssi on paikoin satakertainen.

4.2.3. Viestien sarjallistaminen

Web-palvelu käyttää SOAP-teknologiaa, jonka yksi tehokkuutta vähentävä tekijä on viestien sarjallistaminen. Sarjallistamisprosessi muuntaa vastaanotettavan tietovirran muistirakenteeksi. Viesti sarjallistetaan vastaanotettaessa. Kun viestiä lähetetään, prosessi tapahtuu päinvastaisessa järjestyksessä. Prosessin aikana tietovirta luetaan sokettikerrokselta puskurimuistiin, muunnetaan merkeiksi, konvertoidaan tietotyypit muistirakenteessa käytettyyn formaattiin ja täytetään muistirakennetta. [Abu-Ghazaleh and Lewis, 2005] Jos prosessin aikana suoritetaan myös tiedoston tarkistusta skeemaa vasten, niin saman syötteen oikeellisuus saatetaan tarkistaa eri työvaiheita suorittavissa osaohjelmissa useaan kertaan. Lisäksi eri työvaiheet voivat tehdä ylimääräisiä tyyppimuunnoksia, esimerkiksi muuntaa merkkijonon double-tyyppiseksi muuttujaksi tarkistaakseen sen sopivuuden ja välittää sen seuraavalle vaiheelle edelleen merkkijonona. [Kostoulas et al., 2006]

XML-viestien sarjallistaminen on muistia kuluttavampi prosessi verrattuna esimerkiksi binäärimuotoisiin viesteihin, koska XML-viestit ovat tekstimuotoisia [Jun et al., 2006]. Esimerkiksi javaolioiden sarjallistaminen XML-muotoon vie noin kymmenen kertaa enemmän muistia kuin saman viestin sarjallistaminen binäärimuotoon [Abu-Ghazaleh and Lewis, 2005].

Yksi XML-viestin sarjallistamiseen vaikuttava tekijä on joustavuuden takaaminen. Koneellisen luvun kannalta XML-viestien ei tarvitse sisältää yhtään rivinvaihtoa tai ylimääräistä välilyöntiä XML-elementtien välissä. Jotta viestien tarkistaminen olisi helppoa ohjelman toteuttajalle, viestit voivat kuitenkin sisältää välilyöntejä, joiden avulla elementtejä sisennetään sekä erilaisia rivinvaihtoja (LF-merkki, CR-merkki tai molemmat). Viestien sarjallistajan täytyy tunnistaa, mitkä välilyönneistä ovat osa siirrettävää viestiä ja mitkä eivät. Samoin sarjallistajan tulee konvertoida erilaiset rivinvaihdot yhdeksi LF-merkiksi. [Takase et al., 2005]

Sarjallistamiseen vaikuttaa myös erilaisten tyyppimuunnosten määrä. Muunnoksia voidaan tehdä sekä koodausten, kuten UTF-8- ja UTF-16 -muodot, että tietotyyppien, kuten numero ja merkkijono, välillä. Jos vastaanotettavan viestin koodaus, ts. ulkoinen koodaus, on erilainen kuin viestiä käsittelevän ohjelman koodaus, ts. sisäinen koodaus, ennen käsittelyn aloittamista täytyy tehdä tyyppimuunnos. Esimerkiksi XML-viestit noudattavat yleensä UTF-8 -koodausta ja Java-ympäristö puolestaan UTF-16 -koodausta. Jos siis XML-viestejä käsitellään Javalla toteutetulla ohjelmalla, viestit muunnetaan käsittelyn ajaksi UTF-8 -muodosta UTF-16 -muotoon. [Takase et al., 2005] Jos sarjallistamisen yhteydessä saapuva viesti tarkistetaan skeemaa vasten, elementtien numeroarvot tulee muuntaa tekstistä binäärimuotoon, jotta vertailu voidaan tehdä.

Saattaa olla, että edellä mainitut muutokset suoritetaan sarjallistamisen aikana useammin kuin kerran riippuen siitä, miten eri työvaiheet on jaettu osaohjelmiin [Kostoulas et al., 2006].

4.2.4. Jäsentäjistä

XML-viestien sarjallistamiseen on olemassa useita jäsentäjiä, jotka voidaan jakaa puuperustaiseen ja tapahtumaperustaiseen käsittelyyn. Puuperustainen jäsentäjä muuntaa XML-dokumentin muistiin puumaiseksi tietorakenteeksi käsittelyn ajaksi [Abu-Ghazaleh and Lewis, 2005]. Tapahtumaperustainen jäsentäjä käy läpi XML-dokumenttia ja reagoi erilaisiin tapahtumiin, kuten elementin alkaminen ja loppuminen [Megginson, 2004]. Jäsentäjissä useimmiten käytetty XML-tiedoston puurakenne perustuu DOM-työryhmän suositukseen [W3C, 2005]. Tapahtumaperustaisia malleja ovat esimerkiksi SAX [Megginson, 2004] ja XPP [Slominski, 2005]. SAX-jäsentäjä ilmoittaa dokumentin käsittelijälle tapahtumista. XPP-jäsentäjä antaa käsittelijän kysyä seuraavaa tapahtumaa.

Puuperustaisen tietorakenteen käyttö helpottaa dokumentin käsittelyä siten, että dokumenttia voi käsitellä tagien järjestyksestä riippumatta. Esimerkiksi samaa tietoa voi käsitellä useammin kuin kerran, koska se on saatavilla muistista koko käsittelyn ajan. Rakenne on kätevä käsiteltäessä monimutkaista tietorakennetta, jossa dokumentin eripuolilla olevat tiedot vaikuttavat toistensa käsittelyyn. Toisaalta rakenteen käyttö on myös muistia kuluttavaa, koska koko dokumentti on kerralla muistissa [Abu-Ghazaleh and Lewis, 2005].

Tapahtumaperustainen jäsentäjä puolestaan ei varastoi koko dokumenttia muistiin, vaan käsitellessään dokumenttia jäsentäjä poimii sieltä erilaisia tapahtumia kuten elementin alkaminen tai elementin sisältö [Abu-Ghazaleh and Lewis, 2005]. Jäsentämisen aikana ohjelman kontrolli voi olla joko jäsentäjällä tai käsittelijällä [Kangasharju et al., 2005]. SAX-jäsentäjä jäsentää ensin koko dokumentin ja ilmoittaa sitten käsittelijälle yksi kerrallaan kohdatuista tapahtumista. Käsittelijä reagoi niihin takaisinkutsun avulla. Ohjelman kontrolli säilyy jäsentämisen ajan jäsentäjällä [Abu-Ghazaleh and Lewis, 2005]. Kun kontrolli annetaan käsittelijälle, se voi alkaa dokumentin käsittelyn ennen kuin jäsentäjä on käynyt läpi koko dokumentia. Tällöin käsittelijä kysyy jäsentäjältä seuraavaa tapahtumaa, kun on käsitellyt edellisen. Tämä on tehokasta, jos dokumentin alkuosan käsittely ei ole riippuvainen loppuosasta ja tietoa voidaan prosessoida, ennen kuin koko dokumentti on vastaanotettu. [Abu-Ghazaleh and Lewis, 2005] Tällaisen mallin toteuttavat mm. XPP ja StXA [BEA Systems Inc, 2003].

4.2.5. Optimoituja jäsentäjiä

Useimmiten käytettyjen jäsentäjien lisäksi on toteutettu eri tilanteisiin sopivia optimoituja ratkaisuja. Kostoulas ja muut [2006] ovat toteuttaneen XML Screamer -jäsentäjän, joka yhdistää sarjallistamisen työvaiheita ja käytettyjä kerroksia. Jäsentäjän suunnitteluperiaatteena on toistettavien toimenpiteiden vähentäminen ja vastaanotettavien merkkien käsitteleminen ainoastaan kerran, jos mahdollista. Tieto yritetään käsitellä mahdollisimman pitkälle heti, kun se on luettu, jotta siihen ei tarvitse palata uudelleen. Käsittelyllä tarkoitetaan esimerkiksi tarkistusta skeemaa vasten. Tyypillisen sarjallistajan työvaiheita ovat tietovirran luku, tiedon jäsentäminen, tarkistaminen ja sarjallistaminen. XML Screamer -jäsentäjässä tarkistaminen ja sarjallistaminen on yhdistetty siten, että ne voidaan suorittaa tietovirran koodaukselle, eikä dataa tarvitse vielä tässä vaiheessa muuttaa ohjelman sisäiselle koodaukselle. XML Screamer -jäsentäjä tukee vain UTF-8- ja UTF-16 -koodauksia sekä tavallisimpia skeeman elementtejä.

Abu-Ghazaleh ja Lewis [2005] ovat kehittäneet DS- ja DDS-jäsentäjät, joiden optimointi perustuu viestien samankaltaisuuteen. DS-jäsentäjä muuntaa muistirakenteen XML-muotoon ja sitä käytetään viestin lähettäjän päässä. DDS-jäsentäjä toimii päinvastoin ja sitä käytetään viestin vastaanottajan päässä. Näitä ei tarvitse käyttää samassa palvelussa, vaan ne soveltuvat eri tilanteisiin. Samankaltaisuuteen perustuvasta optimoinnista on eniten hyötyä silloin, kun väliohjelmisto ottaa vastaan samankaltaisia viestejä muilta väliohjelmistoilta [Takase et al., 2005]. Saapuvat viestit ovat XML-muodossa, joten samaa skeemaa noudattavat viestit ovat samankaltaisia riippumatta siitä, millä ohjelmointikielellä viesti on alun perin muodostettu.

DDS-jäsentäjä perustuu siihen, että vastaanotettavat viestit omat samanlaisia lukuun ottamatta muutamaa muuttuvaa tietoa. Näin voi olla esimerkiksi hakupalvelimen vastaanottamissa viesteissä, joissa muuttuva tieto on hakulauseen sisältävä elementti. DDS-jäsentäjä säilyttää edellisen vastaanotetun viestin sarjallistamattomassa ja sarjallistetussa muodossa. Uutta saapunutta viestiä verrataan jäsentämättömään viestiin, ja vain viestien eroavat kohdat sarjallistetaan. Vertailu tehdään tallentamalla tarkastussummia viestien eri kohdista ja vertaamalla näitä. Ratkaisun tehokkuus riippuu siitä, mikä on samankaltaisten viestien prosentuaalinen osuus kaikista viesteistä. Lisäksi tehokkuuteen vaikuttavat tarkastussummien muodostamisen ja varastoinnin tehokkuus. [Abu-Ghazaleh and Lewis, 2005]

DS-jäsentäjä toimii viestin lähettävässä päässä saman periaatteen mukaan kuin DSS-jäsentäjä. Se varastoi edellisen lähetetyn viestin sekä sarjallistetussa että sarjallistamattomassa muodossa. Kun uutta viestiä ollaan lähettämässä, ohjelma vertaa

uutta viestiä vanhaan ja sarjallistaa ainoastaan muuttuneet tiedot. Ratkaisu sopii ympäristöihin, joissa viestejä lähettävä kone on tehokas ja lähettää useita samankaltaisia viestejä peräkkäin. [Abu-Ghazaleh et al., 2004]

Takasen ja muiden [2005] kehittämä Deltaser-jäsentäjä käyttää myös samankaltaisuutta optimoinnin lähteenä kuten DSS-jäsentäjä. Se varastoi edellisen XML-viestin sekä sen lähettämät tapahtumat. Tapahtumat varastoidaan äärelliseen automaattiin, joka koostuu tavujonoista ja tapahtumista. Jäsentäjä vertaa uuden tiedoston tavujonoa tallennettuun tiedostoon ja ilmoittaa kutsujalle tilakoneen mukaisista tapahtumista niin kauan, kun tiedostot vastaavat toisiaan. Kun uusi tiedosto eroaa vanhasta, eroava osa jäsennetään perinteisellä tavalla ja tulos tallennetaan tilakoneeseen seuraavaa vertailua varten. Ratkaisu sisältää myös tarkastuksen, joka tapahtuu inkrementaalisesti vertaamalla uutta tiedostoa jo tarkastettuun tiedostoon. Deltaser-jäsentäjän suorituskykyä on verrattu Xerces- ja Piccolo- jäsentäjään. Tulokset näyttävät, että Deltaser-jäsentäjä käyttää hieman vähemmän aikaa tiedostojen parsimiseen, mutta käyttää enemmän muistia kuin kaksi muuta jäsentäjää.

Viestien samankaltaisuuden hyödyntäminen on viety äärimmilleen Devaramin ja Andresen [2003] esittelemässä lähetettävien viestien varastoinnissa välimuistiin. Ratkaisu on hyödyllinen asiakkaille, jotka lähettävät usein samanlaisia viestejä, joiden tarkoitus on esimerkiksi valuuttakurssin tai pörssikurssin kysyminen. Tällaiset viestit voidaan varastoida ja indeksoida esimerkiksi kohdeyrityksen tai kyselyn sisällön mukaan XML-muodossa tiedostoon. Joten kun viesti halutaan lähettää, ohjelma käy ensin katsomassa välimuistista indeksin perusteella, joko lähetettävälle viestille on olemassa sarjallistettu muoto ja jos näin on, viestin sarjallistettu muoto voidaan lähettää ilman toistuvaa sarjallistamista. Jos viestiä ei ole olemassa, ohjelma toimii perinteisen sarjallistajan kaltaisesti ja muodostaa tarvittavan SOAP-viestin muistista XML-tiedostoksi. Lähettämisen lisäksi ohjelma tallentaa sarjallistetun tiedoston sekä siihen viittavan indeksin välimuistiin. Ratkaisua on verrattu perinteiseen SOAP-sarjallistamiseen sekä Java RMI -teknologiaan. Tuloksissa huomattiin, että kun viestit ovat samanlaisia, Java RMI ja välimuistia hyödyntävä SOAP käyttävät suurin piirtein yhtä kauan aikaa viestin välitykseen. Perinteinen SOAP-teknologia käytti noin 75 % enemmän aikaa. [Devaramin and Andresen, 2003]

Sarjallistettujen viestien varastointi välimuistiin sopii käytettäväksi viestien muodostajan päässä sellaisissa yhteyksissä, jossa lähetettävät viestit ovat usein keskenään identtiset. Edellä mainittua DS-jäsentäjää voidaan käyttää myös silloin, kun tiedostot ovat vain XML-rakenteeltaan samankaltaiset, mutta eroavat tietosisällöltään. Viestejä varastoivien ratkaisujen tehokkuuteen vaikuttaa varastointirakenteen tehokkuus

tietojen haun ja tallennuksen osalta sekä koneen tehokkuus, jolla ohjelma ajetaan. [Abu-Ghazaleh and Lewis, 2005] Ratkaisun valintaan vaikuttanee myös viestiliikenteen nykyinen rakenne ja rakenteen mahdollinen muuttuminen. Tietoa varastoivien ratkaisujen tehokkuuteen vaikuttaa viestien osittainen tai täydellinen samankaltaisuus. Jos se on vähäistä tai sen rakenne saattaa muuttua vähemmän samankaltaiseksi kuin hankintahetkellä, viestien samankaltaisuuteen perustuva ratkaisussa ei voida hyödyntää ratkaisun koko optimointipotentiaalia. Tiedon varastointiin perustuvassa ratkaisussa tulee ottaa huomioon myös sen mukanaan tuoma turvallisuusriski. Joko luottamuksellisia tietoa sisältäviä viestejä ei varastoida ollenkaan välimuistiin tai niiden tallennukseen käytetään salausalgoritmeja, joiden käyttö osaltaan vaikuttaa viestien lähetyksen tehokkuuteen.

4.2.6. Tiedoston koon pienentäminen

HTTP-protokollaa käyttävät SOAP-viestit ovat usein kooltaan suurempia kuin muiden protokollien, kuten TCP- tai UDP-protokollan avulla lähetetyt viestit. HTTP-protokolla rakentuu TCP-protokollan päälle ja sisältää lisäksi mm. header-osan. [Kangasharju et al., 2003] Yksi syy viestin suureen kokoon on nimenomaan HTTP-protokollan viesteissä käyttämä otsikkokenttä. Toinen syy suuriin tiedostoihin on SOAP-viestien käyttämä XML-kieli, joka on ASCII-muotoista. Tekstimuotoiset viestit ovat suurempia kuin vastaavat tiedot sisältävä binäärimuotoinen tiedosto. Myös kolmas syy liittyy XML-kieleen. Se mahdollistaa viestien rakenteisuuden, eli elementtien sisäkkäisen sijoittelun ja kuvailun, jolloin tietojen kuvailu kasvattaa viestien kokoa.

SOAP-viestit ovat ASCII-muotoisia XML-kielisiä tiedostoja. Tämä tuo mukanaan monia hyviä puolia, kuten esimerkiksi virheiden etsimisen helppous, kun tiedosto on ihmiselle luettavassa muodossa [Kangasharju et al., 2003]. Lisäksi SOAP-viestejä voidaan lähettää helposti HTTP-viestin sisällä, SOAP-teknologian päälle on melko vaivatonta rakentaa palveluja sekä viestien lukijoita ja muodostajia toteutuskielestä riippumatta [Govindaraju et al., 2000]. Viestejä on lisäksi helpompi käsitellä kuin vastaavia binäärimuotoisia viestejä [Kangasharju et al., 2003]. Toisaalta tämä hyödylllinen ominaisuus on yksi niistä, jotka huonontavat SOAP-teknologian tehokkuutta. ASCII-muotoisten SOAP-viestien koko on suurempi kuin vastaavien binäärimuotoisten viestien koko, kuten esimerkiksi Java RMI -teknologian viestit [Govindaraju et al., 2000]. Tiedostojen koolla on merkitystä erityisesti esimerkiksi langattomissa päätelaitteissa, joissa vastaanotettavan tiedon määrä ja tiedon prosessointi voivat olla rajoittavin tekijä tietoliikenteessä [Kangasharju et al., 2003].

Govindaraju ja muut [2000] mainitsevat konkreettisen esimerkin ASCII-muotoisia ja binäärimuotoisia viestejä lähettävien protokollien eroista. Kun Java RMI lähettää

double-muotoisen numeron, sen koko on aina 8 tavua. Kun sama lähetetään XML-viestin osana, se esitetään tiedostossa maksimissaan 16 merkkiä pitkänä numerosarjana. Tällöin sen koko on 16 tavua UTF-8 -koodaustavalla. Lisäksi XML-viestiin sisältyy myös numeroa kuvailevat alku- ja loppuelementit, kuten esimerkiksi <double>. Tällöin XML-viestin kooksi double-muotoisen numeron osalta tulisi 33 tavua. Jos käytetään UTF-16 -koodaustapaa, viestin koko on kaksinkertainen. ASCII-muotoisen viestin koko verrattuna binäärimuotoiseen voi olla helposti moninkertainen, varsinkin jos viesti sisältää paljon numeroarvoja.

Yksi SOAP-viestin kokoa kasvattavista tekijöistä on XML-kielen rakenteisuus. XML-viesti sisältää sekä lähetettävän datan että sen kuvauksen. Viestin suunnittelija voi itse muodostaa viestin kuvauksen ja päättää, miten tieto rakentuu ja millä sanoilla varsinaisen tiedon sisältäviä elementtejä kuvataan. Jos SOAP-viestiä kuvaava XML-rakenne on syvä ja tiedon kuvailuun on käytetty pitkiä sanoja, viestin koko kasvaa huomattavasti verrattuna viestiin joka sisältäisi ainoastaan siirrettävän tiedon. Kuvaukseen sisältyvien merkkien määrä voi pian ylittää itse datan merkkimäärän. [Kangasharju et al., 2005]

Gray [2004] vertaa web-palvelun, Java RMI ja CORBA-teknologioiden suorituskykyä. Tuloksista nähdään, että pienillä viesteillä web-palvelu on siirtää dataa noin kymmenen kertaa enemmän kuin Java RMI ja CORBA-toteutukset. Kun tiedostot ovat isoja, erot tasaantuvat. Tällöin web-palvelu siirtää kahdesta kolmeen kertaa enemmän tietoa. Optimointikeinoa valitessa tulee siis ottaa huomioon viestiliikenteen laatu. Jos viestit ovat isoja, SOAP ei ole kovin paljon tehottomampi kuin kilpailijansa. Suurten tiedostojen siirrossa Java RMI lähettää enemmän paketteja kuin CORBA ja SOAP, joten verkkoliikennettä on enemmän. Pakettien määrä ei kuitenkaan aiheuta lisääntyvää prosessointia viestin vastaanottavassa päässä, vaan laskentatehon kulutus on pienempi verrattuna SOAP-teknologiaan ja web-palveluun.

Tiedostokoon pienentämiseksi on esitetty useita ratkaisuja, esimerkiksi viestin pakkaus sekä viestin osittainen muuttaminen binäärimuotoon. Viestin pakkaukseen on useimmiten käytetty Gzip-pakkausohjelmaa. Viestin muuntamiseen binäärimuotoon on esitetty erilaisia tapoja, jotka eroavat esimerkiksi koodattavien elementtien osalta.

SOAP-viestit ovat yleensä melko pieniä, joten pakkausalgoritmit, kuten esimerkiksi Gzip, eivät välttämättä pienennä viestiä riittävästi, koska viesteissä ei esiinny riittävästi samankaltaista tietoa [Kangasharju et al., 2005; Kangasharju et al., 2003]. Toinen syy siihen, että pakkaus ei välttämättä ole sopiva vaihtoehto tiedostokoon pienentämiseen, on pakkauksen muodostamisen ja purkamisen vaatima lisäprosessointi. SOAP-viestien

pakkaaminen erityisesti pienillä päätelaitteilla voisi kuluttaa niin paljon virtaa, että pienemmän tiedoston lähetyksessä saatava hyöty vähenisi. Kangasharjun ja muiden [2003] tutkimuksessa vertailtiin Gzip-pakkausohjelman vaikutusta viestin kokoon ja sen lähetyksen tehokkuuteen. ASCII-muotoisen viestin koko pieneni 40 % ja binäärimuotoisen 50 %. Kun tähän lisättiin pakkaamiseen ja purkamiseen tarvittava aika ja prosessointiteho, pakkauksen hyödyllinen vaikutus arvioitiin melko pieneksi.

Toinen tapa pienentää viestin kokoa on sen osittainen muuntaminen binäärimuotoon. XML-viestin muuntaminen binäärimuotoon vähentää tiedoston kokoa samoin kuin tiedoston pakkaaminen, mutta lisäksi se vähentää myös jäsentämiseen kuluvaa aikaa. Binääri-XML on pienempi esitys ASCII-muotoisesta XML-viestistä, koska osa XML-tiedoston merkkijonoista voidaan korvata lyhemmillä binäärikoodeilla. Binäärimuotoisten osien kohdalla säästytään myös ASCII-muotoon muuntamiselta. [Kangasharju et al., 2005] Tämä mainittiin ensimmäisen kerran WAP Binary XML -ehdotuksessa, jonka tarkoituksena on vähentää XML-dokumentin kokoa ja parantaa sen käytettävyyttä pienillä kaistanleveyksillä [W3C, 1999].

Binäärimuoto säilyttää XML-rakenteen, mutta se sisältää osan viestistä binäärimuodossa. Esimerkiksi usein toistuvat merkkijonot, kuten elementtien nimet, voidaan korvata yhden tai kahden tavun jonolla. Tällöin tiedostosta tulee lyhempi ja sen koneellinen lukeminen helpottuu, kun lukijaohjelman ei tarvitse muodostaa merkkijonoja, vaan se voi lukea suoraan tavuja. [Kangasharju et al., 2005]

Binäärimuoto muodostetaan siinä vaiheessa, kun XML-muotoinen viesti sarjallistetaan. Sarjallistamisella tarkoitetaan XML-viestin muuntamista muistista siirrettävään olomuotoon kuten tiedostoon tai tietovirtaan [Abu-Ghazaleh and Lewis, 2005]. XML-viesti sarjallistetaan käymällä XML-elementit läpi ja muodostamalla halutun muotoinen tiedosto, esimerkiksi teksti- tai binäärimuotoinen tiedosto. XML-viesti voidaan sarjallistaa yhteen tai useampaan eri kohteeseen. Tällöin olisi mahdollista sarjallistaa XML-viestin sisältämä tieto ja tietoa kuvaava rakenne omiin kohteisiinsa. Kun sarjallistamisessa käytettäisiin useampaa kuin yhtä kohdetta, viestin vastaanottajalta vaadittaisiin välimuistin käyttöä. Näin siksi sarjallistettua tietoa ei voisi käsitellä ennen kuin rakenne olisi vastaanotettu ja toisin päin. [Bayardo et al., 2004] Tämä ei siis soveltuisi sellaisille XML-viestin jäsentäjille, jotka lukevat ja käsittelevät viestin samalla kuin lukevat sitä tietovirrasta. Toisaalta rakenteesta voi olla hyötyä, kun viestiä käsiteltäessä halutaan hakea tietoa eri puolilta viestiä.

On olemassa erilaisia tapoja binäärisarjallistaa XML-tiedostoja. Ne muuntavat eri tyyppisiä elementtejä binäärimuotoon, ja niiden käsittelyssä on eroja. Triviaali

binäärikoodaus lisää XML-tiedostoon erottimia, jotka kertovat viestin rakenteesta, kuten esimerkiksi kuinka monta lapsisolmua solmuilla on. Tällaisen viestin lukeminen on yhtä tehokasta kuin tekstimuotoisen viestin lukeminen, joten viestin jäsentämiseen kuluva aika ei kasva. Viestiä voi myös vastaanottaa tietovirtana, koska viestin käsittelyä varten ei tarvita kovin paljon muistia. Verrattuna tekstimuotoiseen viestiin muistia tarvitaan enemmän vain silloin, kun lasketaan solmujen lapsiluku. Triviaalia binäärikoodausta voidaan käyttää jäsentäjien kanssa, jotka voivat varastoida osan tiedostoa välimuistiin käsittelyä varten. Tällöin esimerkiksi tietovirrasta lukeva ja siihen kirjoittava SAX-jäsentäjä ei tue triviaalia binäärikoodausta.

Toinen tapa binäärisarjallistaa XML-tiedostoja on korvata toistuvat merkkijonot, kuten tietoa kuvaavat elementit, tavujonolla tai int-tyyppisellä numerolla. Tällä tavoin sarjallistettua tiedostoa ei voi käsitellä elementti kerrallaan käyttäen mahdollisimman vähän muistia, koska viestin käsittelyyn tarvitaan sanakirja, jonka avulla tavut voidaan muuntaa takaisin niitä vastaaviksi merkkijonoiksi. [Bayardo et al., 2004]

On olemassa myös sarjallistajia, jotka käyttävät hyväkseen skeemaa tai parantavat XML-tiedostossa navigoitavuutta. Ensimmäinen luo riippuvuuden alkuperäisen dokumentin ja skeeman välille. Lisäksi skeeman täytyy myös olla aina käytettävissä sekä lähettäjällä että vastaanottajalla. Navigoitavuus puolestaan vaatii muutoksia XML-tiedostoa käsittelevään ohjelmakirjastoon, joten myös se vähentää ratkaisun riippumattomuutta toteutustekniikoista. Lisäksi navigoitavuuden parantaminen lisää itse käsittelyä, joka tarvitaan viestin sarjallistamisessa ja sen purkamisessa. [Bayardo et al., 2004] Tällöin tulee harkita, tuottaako tiedoston koon pieneneminen ja siihen sisällytetty lisäinformaatio hyötyä yhtä paljon kuin tehokkuudessa sitä hävitään

Binäärisarjallistaminen lisää viestien käsittelyyn tarvittavaa tehoa ja aikaa. Nämä ominaisuudet voivat vähentää tiedostokoon pienenemisestä saatavaa hyötyä. Binääri-sarjallistaminen ei myöskään välttämättä vielä tue kaikkia valittuja tietoturvaratkaisuja. Esimerkiksi XML-kryptaus ja XML-allekirjoitus perustuvat tiedoston alkuperäiseen kokoon, joten kun viesti muunnetaan muodosta toiseen, niiden tarkistaminen hankaloituu. [Kangasharju et al., 2005]

4.2.7. Yhteenveto

Web-palvelu koostuu XML-muotoisista SOAP-viesteistä, jotka kulkevat HTTP-protokollan päällä. Erilaisten optimointien yhteydessä on ehdotettu, että SOAP-viestejä voitaisiin lähettää myös muiden protokollien, kuten esimerkiksi UDP-protokollan, avulla [Combs et al., 2004]. Tämä onnistuisi palveluissa, joissa viestien taattu perillemeno ei ole tärkein vaatimus. Lisäksi on ehdotettu ASCII-muotoisen XML-

viestin korvaamista osittain binäärimuotoisella XML-viestillä, mikä pienentäisi tiedostokokoa [Kangasharju et al., 2005]. Pal ja muut [2003] muistuttavat, että laajalti käytössä olevaa ja standardoitua ASCII-muotoista XML-formaattia ei tule hylätä liian nopeasti binäärimuodon vuoksi ilman pakottavaa tarvetta.

Erilaisten optimointitekniikoiden haasteina on säilyttää SOAP-teknologian ominaisuudet, jotka tekevät siitä varteenotettavan vaihtoehdon viestiperustaisen liikenteen käytössä. SOAP-teknologian vahvuuksia ovat mm. alusta- ja kieliriippumattomuus [Govindaraju et al., 2000]. Optimointivaihtoehdot eivät saisi vähentää liiaksi esimerkiksi käyttöönoton helppoutta käyttällä harvinaista pakkausalgoritmia tai binäärimuotoa. Lisäksi XML-formaatin muuttaminen osittain binäärimuotoiseksi voi osaltaan hankaloittaa palvelun toteutusta ja testausta, kun viestit eivät ole kehittäjälle luettavassa muodossa. Pitkällä aikavälillä optimointiratkaisua suunnitellessa tulee ottaa huomioon mahdollisesti käytetyn standardin kuten pakkausalgoritmin tai binäärimuodon tuen ja käytön jatkuminen. Valittaessa viestiliikenteen rakenteeseen perustuvaa optimointia tulee huomioida, voidaanko olettaa viestiliikenteen pysyvän samankaltaisena. Tehokkuusratkaisuja suunnitellessa tulee myös ottaa huomioon valitun tietoturvaratkaisun tuen jatkaminen ja varoa tuottamasta uusia riskejä, kuten luottamuksellisen tiedon jättäminen välimuistiin, viestejä vertailtaessa.

SOAP-viestien suurimmat optimointihaasteet liittyvät viestin kokoon ja viestin prosessointiin. Viestin kokoa saadaan pienemmäksi pakkausalgoritmeilla tai muuntamalla viestit osittain binäärimuotoon. Binäärimuodon käytöllä voidaan myös vaikuttaa käsittelyn määrään [Kangasharju et al., 2005]. Käsittelyn määrää voidaan myös vähentää viestiliikenteen rakenteesta riippuen asiakas- tai palvelinpäässä käsittelemällä vain edellisestä viestistä eroava osa. Haasteeksi jäänee tehokkuus ja tietoturvaratkaisujen yhteensovittaminen

5. Yhteenveto

Väliohjelmistojen tarkoitus on helpottaa erilaisten osapuolten välillä tapahtuvaa viestiliikennettä. Väliohjelmistona toimiva ratkaisu voi huolehtia viestien perille menon lisäksi esimerkiksi viestin tarkastuksesta sekä väliaikaisesta varastoinnista yhtäaikaisten viestien vastaanoton tapauksessa. Ratkaisu voi sisältää valmiita liiketoimintaprosessin malleja kuten esimerkiksi RosettaNet tai vain tavan kutsua prosesseja etäkoneesta kuten esimerkiksi Java-RMI. Ratkaisut vaihtelevat mm. hinnaltaan, käyttöönottonopeudeltaan, kattavuudeltaan sekä tehokkuudeltaan. Niissä voi olla sisäänrakennettuja laatuominaisuuksia kuten turvallisuusratkaisut. Jos ratkaisut sijaitsevat arkkitehtuurin pohjakerroksilla, ratkaisun käyttäjä ei voi vaikuttaa valittuun tietoturvaratkaisuun. Kevyemmässä ratkaisussa käyttäjän vastuulle jää huolehtia laatuominaisuuksien toteutuksesta. Sopivan ratkaisun valintaan vaikuttaa käyttöympäristö, kuten asiakkaiden ja viestien määrä sekä viestien koko ja käytettävissä oleva verkko ja prosessointiteho.

Web-palvelu on yksi ratkaisuvaihtoehto yritysten väliseen viestintään julkisessa verkossa. Se koostuu SOAP-, WSDL- ja UDDI-teknologioista. Ratkaisun hyviä puolia ovat alustariippumattomuus ja joustavuus, huonoja puolia tekstipohjaisesta muodosta johtuva tehottomuus sekä tietoturvaratkaisujen vakiintumattomuus.

Väliohjelmistojen haasteet sekä niiden suunnittelussa että niitä valittaessa tulevat tuskin vähenemään. Lisähaastetta tuovat uusien teknologioiden kehitys, jotka vanhojen korvaamisen sijaan toimivat niiden rinnalla. Tällöin versioyhteensopimattomuuden saattavat lisääntyä. Lisäksi yksittäisten teknologioiden hyviä ominaisuuksia ei välttämättä pysytä täysin hyödyntämään, jos ei voida olla varmoja, tukeeko toisten osapuolten ratkaisu kyseisiä ominaisuuksia tai kyseistä teknologian versiota. Välttämättä ei voida myöskään tietää, kuinka pitkään uudelle ratkaisulle on luvattu tukea valmistajalta.

Erilaiset väliohjelmisto soveltuvat erilaisiin käyttöympäristöihin. Ratkaisua valitessa voi olla hankala ennustaa, minkälainen käyttöympäristö ja asiakaskunta ratkaisulla on vuosien kuluttua, jolloin ratkaisun valinta hankaloituu. Lisähaasteita tuovat myös sulautetun tekniikan visiossa ounasteltu erilaisten päätelaitteiden lisääntyminen, joiden prosessointiteho on rajattu.

Viiteluettelo

[Abu-Ghazaleh and Lewis, 2005] Nayef Abu-Ghazaleh and Michael J. Lewis, Differential Deserialization for Optimized SOAP Performance. In: *International Conference for High Performance Computing, Networking, and Storage*, 21-31.

[Abu-Ghazaleh et al., 2004] Nayef Abu-Ghazaleh, Michael J. Lewis and Madhusudhan Govindaraju, Performance of Dynamically Resizing Message Fields for Differential Serialization of SOAP Messages. In: *International Symposium on Web Services and Applications*, 783-789.

[Apache, 2003] Apache web services project, Soap, available as http://ws.apache.org/soap/. Checked 15.4.2007.

[ASC X12, 2006] The Accredited Standards Committee X12, available as http://www.x12.org. Checked 15.4.2007.

[Baligand and Monfort, 2004] Fabien Baligand and Valérie Monfort. A concrete solution for web services adaptability using policies and aspects. In: *International Conference on Service-Oriented Computing*, 134-142.

[Bayardo et al., 2004] Roberto J. Bayardo, Daniel Gruhl, Vanja Josifovski and Jussi Myllymaki. An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing. In: *The 13. International World Wide Web Conference*, 345-354.

[BEA Systems Inc, 2003] BEA Systems Inc, Streaming API for XML, Oct. 2003. (JSR-173 Specification), available as http://ftpna2.bea.com/pub/downloads/jsr173_1.0.pdf. Checked 15.4.2007.

[Bhargavan et al., 2005a] Karthikeyan Bhargavan, C´edric Fournet, Andrew D. Gordon and Greg O'Shea, An Advisor for Web Services Security Policies. In: *ACM Workshop on Secure Web Services*, 1-9.

[Bhargavan et al., 2005b] Karthikeyan Bhargavan, C´edric Fournet, Andrew D. Gordon and Greg O'Shea, Verifying Policy-Based Security for Web Services. Technical Report, available as ftp://ftp.research.microsoft.com/pub/tr/TR-2004-84.pdf. Checked 15.4.2007.

[Brodie, 2000] Michael L Brodie, The B2B E-commerce Revolution: Convergence, Chaos, and Holistic Computing. In: *Information System Engineering: State of the Art and Research Themes*, S. Brinkkemper, E. Lindencrona and A. Sølvberg (eds.), Springer-Verlag Ltd., London. June 2000.

[Brose, 2003] Gerald Brose, A Gateway to Web Services Security – Securing SOAP with Proxies. In: *The International Conference on Web Services*, *Lecture notes in computer science* **2853**, 101-108.

[Bussler, 2001] Christoph Bussler, B2B protocol standards and their role in semantic B2B integration engines. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **24**, 1 (2001), 3-11.

[Castati and Shan, 2001] Fabio Casati and Ming-Chien Shan, Models and languages for describing and discovering E-services. In: *ACM SIGMOD international conference on Management of data*, *ACM SIGMOD Record* **30**, 2 (June 2001), 626.

ACM SIGMOD [Christensen et al,. 2001] Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana, Web services description language (WSDL) 1.1, available as http://www.w3.org/TR/wsdl. Checked 15.4.2007.

[Ciancarini et al., 2002] Paolo Ciancarini, Robert Tolksdorf and Franco Zambonelli, Coordination middleware for XML-centric applications. In: *ACM Symposium on Applied Computing*, 336-343.

[Combs, 2004] Harold Combs, Martin Gudgin (editor), John Justice, Gopal Kakivaya, David Lindsey, David Orchard, Alain Regnier, Jeffrey Schlimmer, Stacy Simpson, Hiroshi Tamura, Don Wright, Kenny Wolf, SOAP-over-UDP, available as http://specs.xmlsoap.org/ws/2004/09/soap-over-udp/soap-over-udp.pdf. Checked 15.4.2007.

[Damiani et al., 2001] Ernesto Damiani, Sabrina De Capitani di, Stefano Paraboschi and Pierangela Samarati, Fine grained access control for SOAP E-Services. In: *Tenth International World Wide Web Conference*, 504-513.

[Damiani et al., 2002] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi and P. Samarati. Securing SOAP e-services. *International Journal of Information Security* **1**, 2 (Feb. 2002), 100-115.

[Davis and Parashar, 2002] Dan Davis and Manish Parashar, Latency Performance of SOAP Implementations. In: *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 407.

[Demir et al., 2005] Ömer Erdem Demir, Premkumar Devanbu, Eric Wohlstadter and Stefan Tai, Optimizing layered middleware. In: *Fifth International Workshop on Software Engineering and Middleware*, 33-38.

[Devaram and Andresen, 2003] Kiran Devaram and Daniel Andresen, SOAP optimization via clientside caching. In: *Proceedings of the First International Conference on Web Services*, 520-524.

[Dogac et al., 2002] Asuman Dogac, Yusuf Tambag, Pinar Pembecioglu, Sait Pektas, Gokce Laleci, Gokhan Kurt, Serkan Toprak and Yildiray Kabak, An ebXML infrastructure implementation through UDDI registries and RosettaNet PIPs. In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 512-523.

[ebXml, 2006] ebXml, available as http://www.ebxml.org/. Checked 15.4.2007.

[Engelen, 2004] Robert van Engelen, Code Generation Techniques for developing lightweight XML web services for embedded devices. In: *2004 ACM Symposium on Applied Computing*, 854-861.

[Gergic et al., 2000] J. Gergic, J. Kleindienst, Y. Despotopoulos, J. Soldatos, G. Patikis, A. Anagnostou and L. Polymenakos, An approach to lightweight deployment of web services. In: *14th International Conference on Software Engineering and Knowledge Engineering*, 635- 640.

[Govindaraju et al., 2000] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley and Dennis Gannon, Requirements for and Evaluation of RMI Protocols for Scientific Computing. In: *Conference on High Performance Networking and Computing.*

[Gray, 2004] N.A.B. Gray, Comparison of Web Services, Java-RMI, and CORBA service implementations. In: *Fifth Australasian Workshop on Software and System Architectures.*

[IBM Coorperation, 2001] IBM Cooperation, Web Services Flow Language (WSFL) 1.1, available as http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf. Checked 15.4.2007.

[IBM Coorperation, 2002a] IBM Coorperation, Business Process Execution Language for Web Services, available as http://www.ibm.com/developerworks/library/ws-bpel/. Checked 15.4.2007.

[IBM Coorperation, 2002b] IBM Coorperation, Web Services Policy Assertions Language (WS-PolicyAssertions) 1.0, available as ftp://www6.software.ibm.com/software/developer/library/ws-polas.pdf Checked 15.4.2007.

[IBM et al., 2005] IBM, et al., Web Services trust language, available as http://www-128.ibm.com/developerworks/library/specification/ws-trust/ Checked 15.4.2007.

[IETF, 2006] The Internet Engineering Task Force, Rpc specification, available as http://tools.ietf.org/html/rfc707. Checked 15.4.2007.

[Jun et al., 2006] Wei Jun, Hua Lei and Niu Chunlei, Speed-up SOAP processing by data mapping template. In: *The 2006 International Workshop on Service Oriented Software Engineering*, 40-46.

[Kangasharju et al., 2003] Jaakko Kangasharju, Sasu Tarkoma, and Kimmo Raatikainen, Comparing SOAP Performance for Various Encodings, Protocols, and Connections. In: *8th International Conference, Personal Wireless Communications*, 2003, Lecture Notes in Computer Science, 2775. 397-406.

[Kangasharju et al., 2005] Jaakko Kangasharju, Tancred Lindholm and Sasu Tarkoma, Requirements and design for XML messaging in the mobile environment. In: *Second International Workshop on Next Generation Networking Middleware*, 29-36.

[Kostoulas et al., 2006] Margaret G. Kostoulas, Morris Matsa, Noah Mendelsohn, Eric Perkins and Abraham Heifets, XML Screamer: An Integrated Approach to High Performance XML Parsing, Validation and Deserialization. In: *World Wide Web Conference*, 93-102.

[McIntosh and Austel, 2005] Michael McIntosh and Paula Austel, XML signature element wrapping attacks and countermeasures. In: *Secure web services*, 20 - 27.

[Medjahed et al., 2005] Brahim Medjahed, Boualem Benatallah, Athman Bouguettaya, Anne H. H. Ngu and Ahmed K. Elmagarmid, Business-to-business interactions: issues and enabling technologies. *The VLDB Journal* **12** (2003), 59-85.

[Megginson, 2004] David Megginson, SAX 2.0.1: The Simple API for XML, available as http://www.saxproject.org. Checked 15.4.2007.

[Microsoft, 2006a] Microsoft, COM: Component Object Model Technologies, available as: http://www.microsoft.com/com/default.mspx. Checked 15.4.2007.

[Microsoft, 2006b] Microsoft, .Net documentation, available as http://www.microsoft.com/net/default.mspx. Checked 15.4.2007.

[Netscape, 2006] Netscape, SSL 3.0 Specification, available as http://wp.netscape.com/eng/ssl3/. Checked 15.4.2007.

[Oasis, 2004] Oasis, Universal Description Discovery and Integration specifications, available as http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm. Checked 15.4.2007.

[Oasis, 2005a] Oasis, Security Services (SAML) V2.0, available as http://www.oasis-open.org/committees/security/ Checked 15.4.2007.

[Oasis, 2005b] Oasis, eXtensible Access Control Markup Language (XACML) , available as http://www.oasis-open.org/committees/xacml/ Checked 15.4.2007.

[Oasis, 2005c] Oasis, WS-SecurityPolicy v1.0, available as http://www.oasis-open.org/committees/download.php/15979/oasis-wssx-ws-securitypolicy-1.0.pdf Checked 15.4.2007.

[Oasis, 2006a] OASIS Standards, available as http://www.oasis-open.org/. Checked 15.4.2007.

[Oasis, 2006b] OASIS, Web Services Security (WSS), available as http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss Checked 15.4.2007.

[OMG, 2006] Object management group, CORBA Component Model Specification, available as http://www.omg.org/docs/formal/06-04-01.pdf. Checked 15.4.2007.

[Phan et al., 2006] Khoi Anh Phan, Zahir Tari and Peter Bertok, A benchmark on SOAP's transport protocols performance for mobile applications. In: *2006 ACM Symposium on Applied Computing*, 1139-1144.

[Pal et al., 2003] Shankar Pal, Jonathan Marsh and Andrew Layman, A case against standardizing binary representation of xml. In: *Workshop on Binary Interchange of XML Information Item Sets*, 2003. Available as http://www.w3.org/2003/08/binary-interchange-workshop/29-MicrosoftPosition.htm. Checked 15.4.2007.

[Postel, 1980] J. Postel, User Datagram Protocol. Internet Engineering Note IEN-88, available as http://tools.ietf.org/html/rfc768. Checked 15.4.2007.

[Rahaman et al., 2006] Mohammad Ashiqur Rahaman, Andreas Schaad and Maarten Rits, Towards secure SOAP message exchange in a SOA. In: *Secure Web Services*, 2006, 77- 84.

[RosettaNet, 2006] RosettaNet, available as http://www.rosettanet.org/. Checked 15.4.2007.

[Satyanarayanan, 2001] M. Satyanarayanan, Pervasive computing: vision and challenges. *IEEE Personal Communications* **8**, 4 (Aug. 2001), 10-17.

[Slominski, 2005] Aleksander Slominski, MXP1: Xml Pull Parser 3rd Edition (XPP3), available as http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html. Checked 25.2.2007.

[Sun Microsystems, 1987] Sun Microsystems, Xdr specification, available as http://rfc.sunsite.dk/rfc/rfc1014.html. Checked 15.4.2007.

[Sun Microsystem, 2006a] Sun Microsystems, Rpc specification, available as http://rfc.sunsite.dk/rfc/rfc1057.html. Checked 15.4.2007.

[Sun Microsystem, 2006b] Sun Microsystems, Java Rmi specification, available as http://java.sun.com/products/jdk/rmi/. Checked 15.4.2007.

[Takase et al., 2005] Toshiro Takase, Hisashi Miyaashita, Toyotaro Suzumura and Michiaki Tatsubori, An Adaptive, Fast, and Safe XML Parser Based on Byte Sequences Memorization. In: *World Wide Web Conference*, 692-701.

[Tsur et al., 2001] S. Tsur, S. Abiteboul, R Agrawal, U. Dayal, J. Klein and G. Weikum, Are web services the next revolution in e-commerce. In: *2001 Very Large Data Bases Conference*, 614-617.

[W3C, 1999] World Wide Web Consortium. WAP Binary XML Content Format, available as http://www.w3.org/TR/wbxml/. Checked 15.4.2007.

[W3C, 2002a] World Wide Web Consortium, Web Service Choreography Interface (WSCI) 1.0, available as http://www.w3.org/TR/2002/NOTE-wsci-20020808/. Checked 15.4.2007.

[W3C, 2002b] World Wide Web Consortium, XML-Signature Syntax and Processing, available as http://www.w3.org/TR/xmldsig-core/. Checked 15.4.2007.

[W3C, 2002c] World Wide Web Consortium, XML Encryption Syntax and Processing available as http://www.w3.org/TR/xmlenc-core/. Checked 15.4.2007.

[W3C, 2003] World Wide Web Consortium, SOAP Version 1.2, available as http://www.w3.org/TR/soap/. Checked 15.4.2007.

[W3C, 2004] World Wide Web Consortium, XML Schema (Parts 1 and 2), 4-7-00., available as http://www.w3.org/TR/xmlschema-1/. Checked 15.4.2007.

[W3C, 2005] World Wide Web Consortium, Document object model, available as http://www.w3c.org/DOM/. Checked 15.4.2007.

[W3C, 2006a] World Wide Web Consortium, Namespaces in XML, 1-14-99, available as http://www.w3.org/TR/REC-xml-names/. Checked 15.4.2007.

[W3C, 2006b] World Wide Web Consortium, Web Services Policy 1.2 – Framework, available as http://www.w3.org/Submission/WS-Policy/. Checked 15.4.2007.

[Weiser, 1993] Mark Weiser, Some computer science issues in ubiquitous computing. *Communication of the ACM* **36**, 7 (July 1993), 75-84.

[Winer, 1999] David Winer, Xml-rpc specification, available as http://www.xmlrpc.com/spec/. Checked 15.4.2007.