Heikki Luoma-aho

VÄLIOHJELMISTOT INTEGRAATIOTEKNIIKKANA

Tietojärjestelmätieteen
kandidaatintutkielma
25.3.2009

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Luoma-aho, Heikki Matias
Tietojärjestelmätieteen kandidaatintutkielma / Heikki Luoma-aho
Jyväskylä: Jyväskylän yliopisto, 2009.
35 s.
Kandidaatintutkielma

Tässä tutkielmassa käsitellään tietojärjestelmien integraatiotekniikkana paljon käytettäviä väliohjelmistoja (middleware). Tutkielmassa keskitytään erityisesti viestipohjaisiin väliohjelmistoihin (Message Oriented Middleware, MOM), ja tutkielman tarkoituksena on luoda yleiskuva siitä mitä viestipohjaiset väliohjelmistot ovat ja mihin ne soveltuvat käytettäväksi. Tutkielmassa selvitetään myös viestipohjaisten väliohjelmistojen vahvuuksia ja heikkouksia toimia tietojärjestelmien integraatiorajapintana vertailemalla niitä kahteen muuhun perinteiseen väliohjelmistotyyppiin; etäproseduurikutsuihin (Remote Procedure Call, RPC) ja objektipohjaisiin väliohjelmistoihin (Object Oriented Middleware, OOM). Tutkielman lopussa luodaan myös katsaus siihen mitä web-palvelutekniikat (web services) tuovat mukanaan mietittäessä yritysten järjestelmäintegrointia (Enterprise Application Integration, EAI), ja miten web-palvelutekniikat ja viestipohjaiset väliohjelmistot eroavat toisistaan ja nivoutuvat yhteen.

Tutkielma on toteutettu kirjallisuuskatsauksena, ja sen tulokset osoittavat että erilaiset väliohjelmistomallit saattavat poiketa suurestikin ominaisuuksiltaan ja käyttötarkoituksiltaan mitä tulee yritysten järjestelmäintegrointiin. Sopivan integrointitekniikan valitseminen onkin aina tapauskohtaista, ja usein parhaaseen lopputulokseen päästään yhdistelemällä eri tekniikoiden parhaita puolia.

AVAINSANAT: integraatiotekniikat, sanomajono, väliohjelmisto, viestipohjainen väliohjelmisto.

Ohjaaja: Petri Maaranen

Tietojenkäsittelytieteiden laitos

Jyväskylän Yliopisto

Tarkastaja: Petri Maaranen

Tietojenkäsittelytieteiden laitos

Jyväskylän Yliopisto

SISÄLLYSLUETTELO

1 JOHDANTO

Modernissa yhteiskunnassa yritykset kohtaavat haasteita monesta eri suunnasta. Niiden liiketoiminta, käytettävät teknologiat ja organisaatio ovat alituisen muutoksen alla. Tuotteet tulisi saada markkinoille yhä nopeammin, ja niiden tulisi olla yhä joustavampia ja parempia.

Nämä samat vaatimukset mukautumisesta ja jatkuvasta muutoksesta koskevat myös yrityksissä käytettäviä tietojärjestelmiä. Olipa taustalla olevan muutoksen syynä yritysten fuusioituminen, uusien palvelujen tarjoaminen tai jo olemassa olevien palvelujen levittäminen laajempaan käyttöön, ei menestyvän liiketoiminnan kannalta ole aina mahdollista luoda tietojärjestelmiä uudestaan. Tässä tilanteessa onkin äärimmäisen tärkeää, että uudet toiminnot tai komponentit saadaan integroitua jo olemassa oleviin järjestelmiin mahdollisimman tehokkaasti.

Epäyhtenäisten ja hajautettujen tietojärjestelmien uudelleenluonti yhteensopiviksi jo olemassa olevien järjestelmien kanssa on usein vaativaa, ja se voi kuluttaa paljon resursseja. Usein paras ja järkevin tapa integroida hajautettuja järjestelmiä onkin käyttää näiden välissä erillistä väliohjelmistoa (Middleware), joka mahdollistaa järjestelmien keskinäisen kommunikoinnin niiden eroavaisuuksista huolimatta. (Issarny ym. 2007)

Tässä tutkielmassa käsitellään tietojärjestelmien integraatiotekniikkana paljon käytettäviä väliohjelmistoja. Tutkielmassa keskitytään erityisesti viestipohjaisiin väliohjelmistoihin (Message Oriented Middleware, MOM), ja tutkielman lähtökohtana on luoda yleiskuva sille mitä viestipohjaiset väliohjelmistot ovat, miten ne soveltuvat käytettäväksi rajapintana erilaisten tietojärjestelmien integroinnissa, ja miten ne eroavat muista käsiteltävistä väliohjelmistomalleista. Tavoitteena on siis kartoittaa eri väliohjelmistojen mahdollisuuksia toimia 'yhdistävänä liimana', löyhästi integroituvien hajautettujen tietojärjestelmien välillä. Tutkielma on luonteeltaan kirjallisuuskatsaus ja siinä tutustutaan ensin yleisesti

väliohjelmistot ja viestipohjaiset väliohjelmistot – käsitteisiin, sekä tutustutaan aikaisempaan tutkimukseen aihealueesta. Luvussa neljä tarkastellaan viestipohjaisten väliohjelmistojen käytännön vahvuuksia ja heikkouksia toimia eri järjestelmien välisenä integraatiokerroksena, sekä selvitetään mitä tulee ottaa huomioon viestipohjaisen väliohjelmiston valinnassa ja käyttöönotossa. Luvussa viisi vertaillaan viestipohjaisia väliohjelmistoja tarkemmin kahteen muuhun perinteiseen väliohjelmistomalliin etäproseduurikutsuihin (Remote Procedure call, RPC) ja objektipohjaisiin väliohjelmistoihin (Object Oriented Middleware, OOM). Luvun lopussa luodaan katsaus siihen mitä web-palvelutekniikat (web services) tuovat mukanaan mietittäessä yritysten järjestelmäintegrointia (Enterprise Application Integration, EAI), ja miten web-palvelutekniikat ja viestipohjaiset väliohjelmistot voidaan nivoa yhteen.

2 VÄLIOHJELMSTOT

Väliohjelmisto on tietoteknisenä terminä ja käsitteenä jo melko vanha. Ensimmäinen tiedossa oleva viittaus termiin väliohjelmisto löytyy vuodelta 1968. Tällöin pidettiin Saksassa Garmisch Partenkirchenissä kuuluisa Nato-konferenssi, jossa käsiteltiin ohjelmistotuotannon kriisiä. Tuosta samaisesta konferenssista katsotaan myös juontavan juurensa koko ohjelmistotekniikan ala ja termi ohjelmistotekniikka (software engineering) (Emmerich ym. 2008). Tietotekniikkaa käsittelevissä tieteellisissä kirjoituksissa termin väliohjelmistot toi suurempaan tietoisuuteen kuitenkin vasta vuonna 1996 Philip A. Bernstein julkaistessaan Communication of the Acm – lehdessä artikkelin Middleware: a model for distributed system services.

Mitä väliohjelmistoilla sitten tarkoitetaan? Yleisimmän määrityksen mukaan ne ovat tietoliikenteessä ja palvelinperiaatteen mukaisessa järjestelmässä osien välisiä rajapintoja tai palveluja toteuttavia ohjelmistoja. Toisin sanoen väliohjelmistot ovat komponentteja, jotka ovat sovellusten välissä, ja mahdollistavat erilaisten kokonaisuuksien kommunikoinnin keskenään (Issarny ym. 2007). Väliohjelmistot siis yksinkertaistavat eri organisaatioiden tietojärjestelmien rakennetta, koska niiden avulla voidaan sovelluskehittäjiltä piilottaa mm. käyttöjärjestelmätaso ja käytettävä verkkoteknologia. (Charles 1999)

Väliohjelmisto on siis ohjelmistokerros jonka pääasiallinen tarkoitus on ”selättää” ongelmat liittyen hajautettujen tietojärjestelmien epäyhtenäisyyteen. Tämän lisäksi ne mahdollistavat hyvin tunnettuja ja toimivia ratkaisuja ongelmiin koskien tietojärjestelmien hajanaisuutta, yhteensopimattomuutta, tietoturvaa, luotettavuutta ja käyttövarmuutta. Tietoverkkojen ja järjestelmien tunkeutuessa yhä enemmän osaksi jokapäiväistä elämää, on väliohjelmistoilla yhä suurempi rooli toimia merkittävänä palasena kehitettäessä tulevaisuuden tietojärjestelmiä. (Issarny ym. 2007)

Kirjallisuudessa väliohjelmistot voidaan jaotella niiden toteutusteknologiaa mukaillen neljään eri kategoriaan: Tapahtumapohjaisiin väliohjelmistoihin (transactional middleware), etäproseduurikutsuihin (remote procedure calls), objektipohjaisiin väliohjelmistoihin (object oriented middleware) ja viestipohjaisiin väliohjelmistoihin (message oriented middleware). (Pinus 2004)

3 VIESTIPOHJAISET VÄLIOHJELMISTOT

Viestipohjainen väliohjelmisto on ohjelmisto, joka sijaitsee sekä palvelin että asiakaspäässä hajautettua järjestelmää. Suurin osa viestipohjaisista väliohjelmistoista tukeutuu toiminnoissaan sanomanvälitystekniikkaan jossa käytetään sanomajonoja, mutta on olemassa myös ohjelmistoja jotka perustavat toimintonsa yleislähetys (broadcast) ja ryhmälähetys (multicast) sanomanvälitystekniikoihin. (Idlbek ym. 2005)

Sanomanvälitystekniikkaan perustuvien ohjelmistojen tehtävänä on mahdollistaa ja helpottaa sovellusten välistä kommunikointia sanomajonojen avulla ottamatta kantaa sanoman sisältöön. Viestipohjaisten väliohjelmistojen avulla sovellusten on mahdollista keskustella joko synkronisesti tai asynkronisesti. Yksi viestipohjaisten väliohjelmistojen vahvuus onkin juuri sen mahdollistama asynkroninen kommunikointi. Tämä tarkoittaa sitä, että sanomanvälityksessä ei noudateta pyyntö/vastaus (request/repply)-mallia. Esimerkiksi asiakas/palvelin (client/server) – tyyppisessä ratkaisussa, kun asiakas on lähettänyt sanoman palvelimelle, ei sen tarvitse jäädä odottamaan vastausta vaan voi jatkaa suoraa muihin tehtäviin. Järjestelmä takaa sanoman eteenpäin toimituksen. (Curry 2004)

Asynkronisessa kommunikoinnissa asiakas ja palvelin eivät siis tarvitse loogista yhteyttä kommunikoidakseen, vaan järjestelmät voivat olla aktiivisena eri aikaan. Näin ollen asiakas tai palvelin voi olla tekemässä aivan jotain muuta tai olla kokonaan toimettomana/inaktiivisena silloin kun kommunikointi niiden välillä tapahtuu. Viestipohjaiset väliohjelmistot mahdollistavat myös sanomien pysyvän tallentamisen sanomajonoihin. Tämän ansiosta ongelmatapauksissa, jos esimerkiksi sanoman vastaanottaja on jostain syystä saavuttamattomissa, tai itse väliohjelmistoon tulee jotain vikaa, eivät sanomat katoa mihinkään. (Emmerich ym. 2008)

3.1 Sanomantoimitusmallit

Viestipohjaiset väliohjelmistot tukevat tyypillisesti kahta eri sanomantoimitusmallia: Kohteesta kohteeseen (pointo to point), sekä julkaise ja tilaa (publish/subscribe). Kohteesta kohteeseen toimitusmalli mahdollistaa sanomien välityksen sanomajonojen avulla. Tässä mallissa on aina lähettäjä (sender), joka lähettää sanoman sanomajonoon ja vastaanottaja (receiver) joka vastaanottaa sanoman sanomajonosta. Kohteesta kohteeseen mallissa yhdelle tietylle sanomalle voi tyypillisesti olla vain yksi vastaanottaja. Hyvä esimerkki kohteesta kohteeseen mallia käyttävästä sovelluksesta voisi olla call center – puhelinpalvelu. Siinä puhelu ohjautuu jonoon, josta vastaanottaja sitten poimii sen. Puhelu ei kuitenkaan ohjaudu kun yhdelle vastaanottajalle. (Tai ym. 2003)

Julkaise ja tilaa – mallissa sanoman lähettäjää kutsutaan julkaisijaksi ja sanoman vastaanottajaa tilaajaksi. Tässä mallissa sanomaa ei lähetetä millekään yksittäiselle tilaajalle. Sen sijaan mallin toiminta perustuu siihen, että julkaistavat sanomat luokitellaan tietyin perustein, tietämättä sitä olipa sanomille tilaajia tai ei. Tämän jälkeen kaikki kyseisen luokituksen omaavista sanomista kiinnostuneet tilaajat vastaanottavat luokituksen mukaiset sanomat. Julkaise/tilaa – mallissa tiedon julkaisijan ei tarvitse tietää tiedon vastaanottajaa, eikä tiedon tilaajan tiedon lähdettä. Järjestelmän eri komponenttien toteutuksessa ei myöskään tarvitse kiinnittää huomiota miten yhteydet niiden välillä muodostetaan. Järjestelmä itse huolehtii julkaistun tiedon välityksestä tiedon tilanneille komponenteille. (Eugster ym. 2003)

Julkaise / tilaa – malli jaetaan yleensä kahteen eri muotoon sen mukaan miten haluttujen sanomien vastaanottaminen ja prosessointi on määritelty. Teemanmukaisessa (topic-based) mallissa julkaistavat sanomat identifioidaan nimensä mukaisesti teemoittain. Tämän jälkeen viestipohjainen väliohjelmisto lähettää tiedon teemoista kaikille siihen liittyneille sovelluksille. Sovellusten saatua tiedot teemoista, ne voivat halutessaan liittyä haluttuun teemaan. Liittymisen jälkeen tilaajasovellus vastaanottaa automaattisesti aina kaikki julkaisijasovelluk-

sen lähettämät tilatun teeman mukaiset sanomat. Teemanmukaisessa mallissa sanomien luokittelu teemoittain on julkaisijan vastuulla. (Eugster ym. 2003)

Eugster, Felber, Guerraoui ja Kermarrec (2003) ovat sitä mieltä, että sisältöön perustuva (content-based) malli on teemanmukaista mallia kehittyneempi ja tehokkaampi. Siinä sanomia tai tapahtumia ei ole jaoteltu ennalta määritellyn kriteerin, kuten sanoman teeman mukaisesti, vaan luokittelu tapahtuu sanoman sisällön tai ominaisuuksien mukaan. Sisältöön perustuvassa mallissa sanoman vastaanottaja tilaa haluamansa sanomat määrittelemällä erilaisia filttereitä, joiden avulla rajataan tilattavaksi oikean sisältöiset sanomat. Määritellyt filtterit koostuvat yleensä yksinkertaisista vertailuoperaattorein määritellyistä nimi-arvopareista. näin muodostettu yksinkertainen filtteröinti voisi olla esimerkiksi muotoa: (service = “parking”), (location = “100 Rebecca Drive”), (cost <“3 EURO”), (car-type ≥ “compact”). Sisältöön perustuvassa mallissa sanomat toimitetaan tilaajalle vain, jos niiden sisältö vastaa tilaajan tilattaville sanomille määrittelemiä arvoja tai rajoituksia. Tässä mallissa sanomien luokittelusta vastaa siis niiden tilaaja. (Fiege ym. 2003)

3.2 Viestipohjaiset väliohjelmistot tutkimuskohteena

Viestipohjaiset väliohjelmistot eivät ole tekniikkana vielä kovin vanhoja. Niiden käytön katsotaan alkaneen suuremmassa mittakaavassa vasta 1990-luvulla. Nykyisin niiden merkitys hajautettujen järjestelmien integroinnissa on kuitenkin erittäin merkittävä. Vuonna 2005 viestipohjaisten väliohjelmistojen ohjelmistolisenssien markkinoiden arvioitiin olevan noin miljardi Yhdysvaltain dollaria. Viestipohjaisten väliohjelmistojen markkinajohtajana on ollut jo vuosia IBM, Websphere MQ-tuotteellaan. Muita varteenotettavia toimijoita markkinoilla ovat mm. TIBCO, BEA/Oracle ja Microsoft. (Emmerich ym. 2008)

Viestipohjaisista väliohjelmistoista löytyy maailmalta melko paljon hyviä ja mielenkiintoisia tutkimuksia. Silva-Lepe, Ward ja Curbera (2006) käsittelevät

web-palvelutekniikoiden ja sanomanvälityksen integrointia, tavoitteena löytää mahdollisuuksia hyödyntää verkkopalveluiden toteutuksessa mahdollisimman paljon sanomanvälityksen avainominaisuuksia, vaikuttamalla samalla mahdollisimman vähän verkkopalveluiden ohjelmointimalliin. Hiukan samaa aluetta käsittelevät Tai, Mikalsen ja Rouvellou (2003) julkaisussaan, jossa kartoitetaan viestipohjaisten väliohjelmiston soveltuvuutta implementoitaessa luotettavaa ja käyttövarmaa sanomanvälitystä verkkopalveluille.

Myös Suomessa on tutkittu jonkin verran viestipohjaisia väliohjelmistoja. Aihkisalo ja Välitalo (2008), vertailevat ja analysoivat julkaisussaan *"A Comparative Analysis of the Resource Consumption in Message Oriented Middleware with XML and Binary Encoded Messages"* keskenään kolmea erilaista sanomankoodausmenetelmää toteuttamassaan viestipohjaisessa ohjelmistoympäristössä. Pääkkönen, Pakkala ja Sihvonen (2005) tutkivat omassa julkaisussaan *"An Optimized Message-Oriented Middleware Solution for Extending Enterprise Services to the Mobile Domain"* viestipohjaisten väliohjelmistojen mahdollisuutta toimia 'siltana' tai yhdistävänä tekijänä mietittäessä miten yleensä suojatun toimialueen (domain) sisällä käytettävät lähiverkkotekniikkaan perustuvat yritysjärjestelmät (Enterprise applications) saatettaisiin käytettäväksi ongelmitta myös mobiiliteknologioiden (GPRS, WLAN) kautta.

4 VIESTIPOHJAISTEN VÄLIOHJELMISTOJEN VALINTAAN VAIKUTTAVIA TEKIJÖITÄ

Löyhästi toisiinsa kytkettyihin sovelluksiin liittyvä kriittisin kysymys yritysten kannalta ei ole niinkään se tuleeko sovellukset integroida paremmin toisiinsa, vaan se miten tuo integraatio tulee tehdä. Mitä asioita organisaatioiden sitten tulisi osata ottaa huomioon viestipohjaisen väliohjelmiston valintatilanteessa tai käyttöönotossa? Eli mihin ne soveltuvat käytettäväksi, ja mitkä ovat niiden vahvuuksia ja heikkouksia? Entä mihin tai minkälaisiin tilanteisiin eri sanomantoimitusmallit soveltuvat parhaiten?

Mietittäessä sitä mihin viestipohjaiset väliohjelmistot soveltuvat käytettäviksi, ovat Idlbekin, Hip ja Mustapic (2005) listanneet läpikäytäväksi ainakin seuraavat asiat:

- Minkälaiset vasteaikavaatimukset organisaation sovelluksille on määritelty?
- Mikä on tai tulee olla järjestelmien vikasietoisuus, jos esimerkiksi tietoverkko tai jokin sen solmu kohtaa häiriön?
- Minkälaisia palveluita valitun viestipohjaisen väliohjelmiston halutaan tukevan? Onko esimerkiksi tarvetta sanomien lähettämiselle yhdeltä taholta monelle?
- Onko tarvetta yhdistää toisiinsa sovelluksia, jotka on kirjoitettu eri ohjelmointikielillä tai jotka sijaitsevat eri käyttöjärjestelmillä/alustoilla?
- Voiko yhden uuden komponentin (viestipohjainen väliohjelmisto) lisääminen olemassa olevaan tekniseen arkkitehtuuriin tuoda jotain ongelmia?

4.1 Viestipohjaisten väliohjelmistojen vahvuudet

Mietittäessä puhtaasti viestipohjaisten väliohjelmistojen vahvuuksia, ei voida ohittaa sitä tärkeätä tosiasiaa, että ne tarjoavat mahdollisuuden rakentaa löyhästi toisiinsa kytketyistä eri alustoilla sijaitsevista ja eri ohjelmointikielellä koodatuista järjestelmistä vakaan ja luotettavan systeemin, joka ei kärsi käyttökatkoista, vaikka yhdessä järjestelmän komponenteista olisi vikaa. On myös itsestään selvää, että yksi viestipohjaisten väliohjelmistojen suurimmista vahvuuksista, mietittäessä yrityksen tietojärjestelmien integraatiotekniikoita ja itse integraatiota, on niiden avulla mahdollisesti saavutettava merkittävä taloudellinen säästö.

Viestipohjaisten väliohjelmistojen selkeänä vahvuutena voidaan pitää myös sitä, että sanomanvälitykseen perustuvaa liikennettä on erittäin helppo monitoroida ja valvoa. Helppo monitoroitavuus taas osaltaan edesauttaa sitä, että sanomanvälitykseen perustuvien järjestelmien suorituskykyä voidaan säätää ja rajoitta halutulla tavalla melko yksinkertaisesti (Banavar ym. 1999). Merkittävä vahvuus on myös viestipohjaisten väliohjelmistojen tarjoama hyvä vikasietoisuus, koska sanomat voidaan tallettaa pysyvästi jonoihin, jotka sijaitsevat palvelimen kovalevyllä. Viestipohjaisten väliohjelmistojen ominaisuuksiin kuuluu myös takuu siitä, että sanomat välitetään varmasti kohdejärjestelmälle (at most once), ja ettei duplikaatteja sanomia lähetetä (exactly once). (Molina-Jimenez ym. 2007)

4.2 Viestipohjaisten väliohjelmistojen heikkoudet

Viestipohjaisten väliohjelmistojen käytölle on siis olemassa selkeitä ja hyvin merkittäviä perusteita. On kuitenkin hyvä muistaa, että viestipohjaisten väliohjelmistojen käyttöön liittyy myös epäkohtia. Idlbek, Hip ja Mustapic (2005) muistuttavat, että kaupalliset väliohjelmistot ovat yleensä melko kalliita, ja nii-

den vaatimat ohjelmistoasennukset voivat vaatia paljon työtunteja, ja lisäävät näin osaltaan kustannuksia.

Ongelmia voi seurata myös jos organisaation sisällä päädytään käyttämään vain yhtä implementaatiota viestipohjaisista väliohjelmistoista. Tämä aiheuttaa helposti riippuvuuden myös pitkälle tulevaisuuteen tähän yhteen ja samaan ohjelmistotoimittajaan. Yhteen tuotteeseen tai toimittajaan sitoutumalla sitoutetaan osittain myös järjestelmien ylläpito, tukitoiminnot ja tulevaisuuden tarpeet vuosiksi eteenpäin. Tällä voi olla merkittäviäkin vaikutuksia järjestelmien joustavuuteen, laajennettavuuteen ja yhteensovittamiseen tulevaisuudessa.

Yksi suurimmista ongelmista koskien viestipohjaisia sanomanvälitysohjelmia on yhteisen standardin puute. Käytännössä kaikilla suurilla toimittajilla on omat toteutuksensa, jotka käyttävät omia niihin räätälöityjä rajapintoja, ja hallinnointityökaluja. Tästä voi seurata ongelmatilanteita, esimerkiksi siinä tapauksessa, että kaksi yritystä sulautuu yhteen yritysoston kautta. Mitä tapahtuu jos fuusioituneilla yrityksillä on ollut käytössä eri toimittajien viestipohjaiset väliohjelmistot? Ja mistä löytyy yhteinen standardi rajapinta, joka yhdistäisi nämä kaksi eri tuotetta? Onneksi yhteisen standardin puutteeseen on olemassa kuitenkin erilaisia ratkaisuja. Eniten käytetty ja parhaiten tuettu sovellusliittymä (Application Programming Interface, API), joka mahdollistaa viestinvälitystoiminnot sovellusten osien tai eri sovellusten välillä on Sunin omistaman, mutta avoimeen standardiin perustuvan Java Enterprise Edition -alustan JMS-palvelu (Java Message Service). (O'Hara 2007)

Varsinaiseen standardin puutteeseenkin on viime vuosina yritetty keksiä ratkaisuja. Tällä hetkellä mielenkiintoisin ja pisimmällä oleva ratkaisu on AMQP (Advanced Message Queuing Protocol). AMQP on JPMorgan investointipankin arkkitehdin kehittämä avoin standardi viestipohjaisille väliohjelmistoille. Toisin kuin esimerkiksi JMS, AMQP ei pyri standardoimaan väliohjelmistoja sovellusliittymätasolla, vaan siinä on määritelty sekä tarvittavat verkkokäytännöt, että tarpeellinen määrä sanomanvälityksen omaa semantiikkaa. (O'Hara 2007)

Curryn (2004) mukaan ensisijainen ongelma liittyen viestipohjaisiin väliohjelmistoihin juontaa kuitenkin sen tarpeesta lisätä yksi komponentti (sanomia välittävä ohjelmisto) organisaation tekniseen arkkitehtuuriin. Uuden komponentin tai ohjelmiston lisääminen mihin tahansa arkkitehtuuriin on aina oma riskinsä. Uusi komponentti voi aiheuttaa suorituskyvyn ja luotettavuuden heikkenemistä, ja koko järjestelmän ylläpidosta voi tulla aiempaa vaikeampaa ja kalliimpaa.

4.3 Sanomantoimitusmallien vahvuudet ja heikkoudet

Suurin osa markkinoilla olevista viestinvälitysohjelmistoista tukee molempaa kohteesta kohteeseen ja julkaise tilaa sanomantoimitusmallia. Mallien ominaispiirteistä johtuen ne soveltuvat kuitenkin parhaiten käytettäväksi erilaisissa tilanteissa.

Kohteesta kohteeseen tapahtuva sanomantoimitus sopii parhaiten ns. rajoitetun liikkuvuuden laitteiden (nomadic applications) kuten matkapuhelimien ja kannettavien tietokoneiden sovellusten yhteydessä käytettäväksi. Yleisesti voidaan sanoa, että kohteesta kohteeseen tapahtuva malli sopii hyvin sovelluksille, jotka ovat jollain tapaa ajasta riippumattomia. Kohteesta kohteeseen – malli soveltuu käytettäväksi erinomaisesti myös asynkroniseen kommunikaatioon missä asiakas ja palvelin voivat toimia eri aikaan tai eri nopeudella. (Hadim ym. 2006)

Mihin asynkroninen kohteesta kohteeseen – malli sitten ei sovi parhaalla mahdollisella tavalla? Itsestään selvää on, että mallia ei voida käyttää tilanteessa jossa sovelluksen pitää saada vastaus palvelimelta tietyissä aikarajoissa. Tällaisessa tapauksessa onkin syytä käyttää jotain synkroniseen kommunikointiin perustuvaa muuta väliohjelmistomallia kuten etäproseduurikutsuja.

Julkaise ja tilaa mallin vahvuuksia ovat Cugola, Migliavacca ja Monguzzi (2007) kuvailleet seuraavasti:

- Mahdollisuus asynkronisuuteen, koska julkaisija ja tilaaja kommunikoivat rinnakkain omaan tahtiinsa.
- Monenvälisyys, koska sanomat lähetetään kaikille niistä kiinnostuneille.
- Nimettömyys, koska julkaisijan ja tilaajan ei tarvitse tuntea toisiaan.
- Implisiittisyys, koska tilaaja, ei niiden julkaisija määrittelee sanomien vastaanottajat.
- Tilattomuus, koska sanomat eivät jää järjestelmään, vaan ne vain lähetetään niille komponenteille jotka ovat tilanneet ne ennen julkaisemista.

Näiden yllä mainittujen ominaisuuksien ansiosta sanoman julkaisija ja tilaaja saadaan erotettua hyvin toisistaan, ja tämä taas helpottaa ohjelmistosuunnittelua vähentämällä tarvetta lisätä ja poistaa komponentteja ajonaikaiseen ohjelmistoarkkitehtuuriin. Jos on nimettömyys yksi julkaise ja tilaa mallin vahvuuksista, pitävät Cugola, Migliavacca ja Monguzzi (2007) sitä myös yhtenä järjestelmän heikkoutena. Monessa tapauksessa kun sanoman julkaisijan on ehdottoman tärkeätä saada jokin tieto siitä, että sanoma on vastaanotettu tai prosessoitu eteenpäin. Toisena ongelmana he pitävät julkaise ja tilaa mallin haasteellisuutta tietoturvamielessä.

4.4 Muita valintaan ja käyttöönottoon liittyviä huomioita

Jos organisaatiossa koetaan tarvetta integroida sovelluksia, ja tähän tarkoitukseen päätetään käyttää viestipohjaisia väliohjelmistoja, kannattaa itse tuotteen valintaan liittyen miettiä viestipohjaisten väliohjelmistojen ja sanomantoimitusmallien hyvien ja huonojen puolien lisäksi ainakin seuraavia eri tuotteiden tarjoamia palveluita, ja niiden toteutustapoja:

- Mikä on sanomien tiedostoformaatti? Toteutuksesta riippuen nykyiset viestipohjaiset väliohjelmistot tukevat lähes minkä muotoista tahansa dataa esimerkkeinä puhdas teksti, XML ja suoratoistettava multimedia (streaming multimedia). Pitää kuitenkin muistaa, se että viestipohjaiset väliohjelmistot eivät käytännössä tunnista tai ota kantaa lähetettävän sanoman formaattiin (Curry 2004). Olipa siis sanoman sisältö esimerkiksi binäärimuotoista dataa, merkkijonoja tai olioita, vastaanottavan sovelluksen/järjestelmän vastuulle jää datan lopullinen konvertointi sen haluamaan muotoon.

- Mikä on sanomien lähetyksessä käytettävä menetelmä tai metodi? Eli halutaanko sanomien olevan pysyviä (persistent) vai saavatko sanomat olla ei-pysyviä (non persistent), jolloin ne katoavat esimerkiksi siinä tapauksessa, että sanomien lähettäjä kohtaa jonkin ongelmatilanteen. Entä onko sillä merkitystä missä järjestyksessä sanomat halutaan lähettää tai vastaanottaa? Luetaanko sanomat jonosta first in first out – menetelmän mukaisesti, jolloin kauiten jonossa ollut sanoma otetaan käsittelyyn ensimmäisenä, vai halutaanko sanomille määritellä prioriteetti joka määrittelee käsittelyjärjestyksen? (Curry 2004)

- Miten transaktiot eli tapahtumat rekisteröidään ja miten niiden lokien keräys hoidetaan eri tuotteissa? Usein sanomanvälitysympäristö implementoidaan asentamalla erillinen palvelin, jolle käytettävät resurssit kuten sanomajonot määritellään. Näin ollen myös tapahtumien rekisteröinti ja lokin keräys tapahtuu yleensä tämän oman palvelimen kautta.

- Miten älykäs reititys (intelligent routing) on hoidettu? Eli millä tavalla taataan, että sanomat päätyvät oikeille vastaanottajille sen sisältämän tiedon perusteella? Tarvitaanko tähän erikseen siihen tarkoitettuja ratkaisuja kuten message broker -sanomanvälitystuotteita tai palvelukeskeiseen arkkitehtuuriin (Service Oriented Architecture, SOA) läheisesti

liittyvää ESB-palveluväylää (Enterprise Service Bus) hyödyntäviä ratkaisuja? (Endrei ym. 2004)

5 VÄLIOHJELMISTOMALLIEN VERTAILUA

Miten viestipohjaiset väliohjelmistot sitten eroavat kahdesta muusta perinteisestä väliohjelmistomallista, etäproseduurikutsuista (Remote Procedure Call, RPC) ja objektipohjaisista väliohjelmistoista (Object Oriented Middleware, OOM)? Entä mitä uutta viime vuosina erittäin tärkeään rooliin sovellusten integraatiosta puhuttaessa noussut Web Service -teknologia tarkoittaa ja tuo mukanaan?

5.1 MOM vs. RPC

RPC on protokolla, jota käytetään kahden prosessin väliseen tiedonvälitykseen kahden kohdejärjestelmän välillä. Protokollan avulla yhdessä kohdejärjestelmässä suoritettava prosessi voi kutsua toisessa kohdejärjestelmässä suoritettavan prosessin toimintoja. (Emmerich ym. 2008)

Yksi merkittävä ero viestipohjaisten väliohjelmistojen ja etäproseduurikutsujen välillä on niiden käyttämässä toimintalogiikassa. Etäkutsuissa laitteelle osoitettu kysymys-vastaus – kokonaisuus on synkroninen, tällöin kysyjä odottaa kunnes saa vastauksen lähettämäänsä kyselyyn. Viestipohjaisissa ohjelmistoissa tekniikka on usein asynkronista, jolloin kysymyksen lähetettyään kysyjä voi jatkaa toimintaansa odottamatta vastausta. (Tunturi 2004)

Toinen looginen ero näiden kahden mekanismin välillä on Tunturin (2004) mukaan voimakkuus/vahvuus, jolla väliohjelmisto sitoo komponentit toisiinsa. Etäproseduurikutsutekniikka vaatii, että komponentit tietävät toistensa etämetodit, kun taas viestipohjaisiin mekanismeihin tukeutuvien komponenttien pitää tietää vain lähetettävän sanoman muoto.

Etäkutsujen ja viestipohjaisten ohjelmistojen välillä on eroja myös siinä mitä tulee niiden käyttöön. Etäkutsut soveltuvat parhaiten pieniin ja yksinkertaisiin

järjestelmiin, ja niitä käytetään yleensä jakamaan ja yhdistämään asiakas- ja palvelinkomponentteja joissa asiakas on loppukäyttäjä, kun taas viestipohjaisissa järjestelmissä asiakkaat ovat usein hajanaisia ohjelmistokomponentteja jotka ovat yhteensopivia vain sanomanvälityksen keinoin. Merkittävimpiä viestipohjaisten väliohjelmistojen ja etäproseduurikutsujen välisiä eroja on koottu taulukkoon 1. (Do-Guen Jung ym.1999).

Taulukko 1. Viestipohjaisten väliohjelmistojen ja etäproseduurikutsujen erot (Do-Guen Jung ym.1999).

Ominaisuus	**Viestipohjaiset väliohjelmistot**	**Etäproseduurikutsut**
Vertauskohta	Postitoimisto	Puhelin
Kommunikointitapa	Asynkroninen	Synkroninen
Kommunikointityyli	Jonotus	Kutsu/paluu (call/return)
Kuormantasaus	Yksittäisille jonoille voidaan määritellä erilaisia sääntöjä koskien sanomien prioriteettia ja lukujärjestystä	Vaatii erillisin tapahtuma-monitorin (TPM, Transaction Processing Monitor), jolla voidaan seurata kahden tai useamman sovelluksen välisiä tapahtumia
Viestin perille menon varmistavan transaktio-naalisuuden tuki	Kyllä	Ei
Asynkroninen prosessointi	Kyllä. Vaatii jonoja ja liipaisimien (trigger) käyttöä	Rajoitettua. Vaatii monimutkaista koodia säikeitten hallinnointiin

Molemmilla tekniikoilla on omat hyvät ja huonot puolensa. Viestipohjaisia väliohjelmistoja pidetään yleisesti etäkutsutekniikkaa vikasietoisempana ja joustavampana ratkaisuna, koska useimpien ohjelmistotoimittajien toteutukset tukevat sekä synkronista ja asynkronista kommunikointia. Vastakohtaisesti joustavuudelle viestipohjaisesti toimivien sovellusten ohjelmointi on kuitenkin huomattavasti hankalampaa ja monimutkaisempaa, koska se ei ole tekniikkana ohjelmoijalle yhtä läpinäkyvää kuin etäkutsujen käyttäminen. (Menascé 2005)

Viestipohjaisten väliohjelmistojen heikkoutena voidaan pitää myös sitä, että asynkronisuudesta johtuen, se ei varaudu toimiessaan siihen ylikuormittaako se verkkoa. On myös mahdollista että asiakas kuormittaa itseään niin suurella työkuormalla, että resurssit joilla se ohjaa toimintojaan kulutetaan loppuun, ja järjestelmä kaatuu. Nämä ongelmat voidaan kuitenkin minimoida tai välttää monitoroimalla suorituskykyä, ja säätämällä järjestelmän läpi kulkevien sanomien virtaa. Monet sovellukset kommunikoivat myös luontaisesti synkronisesti. On hyvin tavallista, että tapahtuman lähettäjä haluaa saada vastauksen tapahtumalle ennen kuin jatkaa toimintojaan. Koska viestipohjaisten väliohjelmistojen luontainen tapa toimia on asynkronista, eivät ne tämmöisissä tilanteissa välttämättä ole paras mahdollinen toimintamalli. (Banavar ym. 1999)

Idlbek, Hip ja Mustapic (2005) ovat listanneet viestipohjaisten väliohjelmistojen vahvuuksia suhteessa etäproseduurikutsuihin seuraavasti:

- Yksinkertainen sovelluskehitysympäristö
- Järjestelmän kommunikointi ei kärsi yhtä helposti verkon käytettävyydessä ilmenneistä ongelmista
- Kommunikointimekanismi takaa sanomien perille toimittamisen
- Eri alustojen (Linux, Windows, Macintosh jne.) helppo integroitavuus
- Vikasietoisuuden ansiosta soveltuu käytettäväksi hyvin integroimaan järjestelmiä joiden välillä on pitkiä maantieteellisiä välimatkoja
- Sanomia voidaan priorisoida, ja niiden järjestelmälle aiheuttamaa kuormaa voidaan tarkkailla ja tasata
- Vähemmän tiedon ylikuormitusta järjestelmien kommunikoidessa ja sitä kautta matalammat kulut
- Sanomanvälitystekniikka ei ole vaativaa tietoliikennemielessä, joten se toimii hyvin myös hitaammilla verkkoyhteyksillä

- Sanomanvälityksellä voidaan integroida hajanaisia järjestelmiä, ilman että luovuttaisiin joustavuudesta

5.2 MOM vs. OOM

Objektipohjaiset väliohjelmistot on kehitetty etäproseduurikutsujen pohjalta, lisäämällä mukaan olioperustaisen ajattelun käsitteitä, kuten periytyminen (inheritance), olioviittaus (object references) ja poikkeustilanteet (exceptions). Etäproseduurikutsujen mukaisesti objektipohjaiset väliohjelmistot tukevat myös useita eri ohjelmointikieliä, joten objektien ei tarvitse olla kirjoitettu palvelin ja asiakaspäässä samalla kielellä. (Pinus 2004)

Objektipohjaisia väliohjelmistoja ilmennetään usein hajautettujen objektien kommunikoinnin verkon yli mahdollistavan komponentin (object request broker, ORB) avulla. Yksi käytetyimmistä ja yleisimmistä spesifikaatioista hajautetuille objekteille on CORBA (Common Object Request Broker Architecture). Muita suosittuja ilmentymiä ovat mm. Microsoftin DCOM (Distributed component object model) ja puhtaasti Java-kieltä varten suunniteltu olioiden hajautustekniikka Java RMI (Remote Method Invocation). (Curry 2004)

Eri spesifikaatioiden ansiosta Objektipohjaisiin väliohjelmistoihin sisältyy paljon hyviä ominaisuuksia ja palveluita. Esimerkiksi AMI (Asynchronous Method Invocation) suunnittelumalli, joka mahdollistaa asynkronisen kommunikaation. Objektipohjaisten väliohjelmistojen ominaisuudet takaavat myös sen, että sanomat toimitetaan varmasti perille (At most once), eikä duplikaatti-sanomia esiinny (exactly-once). Spesifikaatioiden mukana tulevat palvelut parantavat myös mm. kuormantasausta ja vikasietoisuutta. (Schmidt ym. 1999)

Objektipohjaisten väliohjelmistojen yksi vahvuus on siinä, että ne ovat hyvin yhteensopivia järjestelmäkehityksessä paljon käytetyn oliopohjaisen suunnittelun -ja toteutusmallin kanssa. Näin ollen ne yhdenmukaistavat ja tukevat hyvin järjestelmien kehitysprosessia. Koska merkittävä osa viime

vuosina toteutetuista sovelluksista pohjautuu olioperustaisiin kieliin, ovat objektipohjaiset väliohjelmistot hyvä ja luonnollinen valinta niiden integrointialustaksi. (Tai ym. 2001)

Objektipohjaisten väliohjelmistojen suurin ero viestipohjaisiin ohjelmistoihin on siinä, että niiden avulla luodut järjestelmät koostuvat tiiviisti toisiinsa kytketyistä komponenteista. Näin ollen objektipohjaisten väliohjelmistojen käyttäminen ei ole kovin suositeltavaa, jos niiden aikaansaama komponenttien toisiinsa kytkeytyminen koetaan ongelmalliseksi. Objektipohjaisia ohjelmistoja pidetään myös yleisesti viestipohjaisia ohjelmistoja monimutkaisempina ja raskaampina. Ongelmia voi tuoda myös se, että koska ne pohjautuvat olio-ohjelmointikieliin tapahtuu kommunikointi niissä historiallisesti synkroniselta pohjalta (Tai ym. 2001). Vahvuutena sen sijaan voidaan pitää sitä, että prosessit joilla korkean tason tieto muutetaan muotoon, joka sopii sellaisenaan tiedonsiirtoon (Marshalling), ja joilla vastaanotettu tieto jäsennetään takaisin samaksi korkean tason tiedoksi (Unmarshalling) generoituvat automaattisesti. Viestipohjaisissa väliohjelmistoissa vastaavat prosessit pitää ohjelmoijan itse implementoida. (Issarny ym. 2007)

Kun yrityksessä lähdetään toteuttamaan sovellusintegraatioita, on tärkeintä ymmärtää eri ratkaisujen edut ja vastuut tietojärjestelmiin liittyen. Jokainen järjestelmä on erilainen ja uniikki, ja niillä hoidetaan erilaisia tehtäviä. Usein onkin niin, että järkevin ratkaisu voi olla yhdistää objektipohjainen ja viestipohjainen malli. Esimerkiksi jonkin yhden järjestelmän sisällä voidaan toimia niin, että sanomanvälitystä käytetään kun kommunikoidaan erilaisten perinteisten taustajärjestelmien kesken, ja samanaikaisesti verkkoon sidotut edustajärjestelmät hyödyntävät objektipohjaista mallia. Näin yhdessä käytettynä objekti- ja viestipohjaiset väliohjelmistot pystyvätkin ratkaisemaan ison osan sovellusten integrointiin liittyvistä ongelmista. (Tai ym. 2001)

5.3 MOM vs. Web-palvelutekniikat

Viestipohjaisia väliohjelmistoja on käytetty onnistuneesti hyödyksi rakennettaessa avoimia ja joustavia järjestelmiä, ja integroitaessa näitä saumattomiksi kokonaisuuksiksi. Vaikka viestipohjaiset väliohjelmistot ratkaisevatkin useita tiedonkuljetukseen liittyviä ongelmia, on sovellusintegroinnissa ratkaistavana vielä useita suuria ongelmia. Yksi näistä ongelmista koskee tiedon esitystapaa, muotoa ja rakennetta. Jotta voitaisiin rakentaa oikeasti avoin järjestelmä, viestipohjaiset väliohjelmistot tarvitsevatkin kylkeensä avuksi muita teknologioita kuten XML-kuvauskieli (eXtensible Markup Language) ja web-palvelutekniikat. (Curry 2004)

Web-palvelutekniikka on varsin uusi teknologia, joka tarjoaa mahdollisuuden palveluiden käyttämiseen verkossa koneiden välillä ilman ihmisen vuorovaikutusta. Web-palvelutekniikka on siis kokonaisuus, joka sisältää välineet ja menetelmät web-palvelujen hyödyntämiseksi. Web-palvelutekniikoiden katsotaan kehittyneen etäproseduurikutsuista, ja niiden toiminta perustuu samaan periaatteeseen. Ne ovat siis funktioita joita voidaan kutsua ja käyttää verkon yli. Web-palvelutekniikat eroavat kuitenkin etäproseduurikutsuista muun muassa siinä, että niiden katsotaan liittyvän läheisesti palvelukeskeiseen arkkitehtuuriin (Service Oriented Architecture, SOA), ne pohjautuvat XML-kieleen ja toimivat useiden eri protokollien yli. Web-palvelutekniikoiden voidaankin katsoa olevan palvelukeskeisen integroinnin toteutusvälineitä. Palvelukeskeisen arkkitehtuurin perusidea on tehdä sovelluksista ja jopa niiden komponenteista joustavasti yhdisteltäviä itsenäisiä palveluita, jotka keskustelevat standardinmukaisten XML-tekniikoiden avulla. (Curry 2004)

Web-palvelutekniikka koostuu useista protokollista ja tekniikoista, joille on syntynyt eri tahojen määrittelemiä vaihtoehtoja, laajennuksia ja lisäyksiä. Web-palvelutekniikkaan kuuluvat perustekniikat ovat W3C:n määrittelemät SOAP (Simple Object Access Protocol) – protokolla ja WSDL (Web Services Description Language) – web-palvelun kuvaustekniikka, sekä OASIS (Organization for

the Advancement of Structured Information Standards)-konsortion hallitsema UDDI (Universal Description, Discovery and Integration) – hakemisto jossa voidaan julkaista web-services kuvauksia. Kaikki edellä mainitut tekniikat käyttävät tiedon esitysmuotona XML:ää, jonka avulla tiedon merkitys on kuvattavissa tiedon sekaan. (Emmerich ym. 2008)

Vaikka Web-palvelutekniikoiden katsotaan kehittyneen etäproseduurikutsuista, palveluiden väliseen viestintään tarkoitettua SOAP: a voidaan käyttää sidontojensa kautta sekä synkronista, että asynkronista kommunikointia toteuttavien protokollien kuten HTTP (Hypertext Transfer Protocol) ja SMTP (Simple Mail Transfer Protocol) kanssa (Emmerich ym. 2008). Toinen palvelupohjaisen arkkitehtuurin peruspiirteistä on palveluiden löyhä kytkeytyminen toisiinsa. Web-palvelutekniikoihin liittyvät eri spesifikaatiot ovatkin täysin riippumattomia toisistaan ajatellen niiden suhdetta mm. ohjelmointikieliin, käyttöjärjestelmiin ja laitteistoihin. Tämä mahdollistaa erittäin hajanaisten sovellusten integroinnin, ja hyvän yhteentoimivuuden ja skaalautuvuuden eri ohjelmistotoimittajien ratkaisujen välillä. Web-palvelutekniikoilla toteutetut sovellukset ja jopa sovellusten eri komponentit voivat siis olla hyvin löyhästi toisiinsa kytkettyjä. (Endrei ym. 2004)

Tarkastellessa web-palvelutekniikoita integroitavien järjestelmien näkökulmasta niiden käytön suurimpia hyötyjä on, että palvelut perustuvat itsensä kuvaaviin XML-sanomiin. XML-kielen myötä web-palvelutekniikoiden käyttö on alustariippumatonta ja hyvin skaalautuvaa. Tosin sanoen palveluntarjoaja ja asiakassovellus voivat olla ohjelmoitu eri ohjelmointikielillä, ja ne voivat toimia eri käyttöjärjestelmissä. Lisäksi sovelluksen suorittavat laitteet voivat olla täysin erilaisia, esimerkiksi matkapuhelin ja järeä sovelluspalvelin. Vahvuutena voidaan pitää myös sitä, että SOAP-viestit välitetään yleisimmin HTTP-protokollaa käyttäen, tällöin tieto reititetään saman tietoliikenneportin lävitse kuin verkkoselaintenkin liikenne. Tämän ansiosta vältytään yrityksen tietoturvapolitiikan kannalta ongelmallisten TCP/IP-porttien käytöltä. SOAP -protokolla on myös

yksinkertainen ja kevyt käyttää, eikä se vaadi suuria prosessointitehoja laitteilta. (Nevala 2008)

Web-palvelutekniikoita käytettäessä on myös mahdollista generoida automaattisesti, aivan kuten etäproseduurikutsuissa ja oliopohjaisissa väliohjelmistoissa, prosessit joilla korkean tason tieto muutetaan muotoon joka sopii sellaisenaan tiedonsiirtoon, ja joilla vastaanotettu tieto jäsennetään takaisin samaksi korkean tason tiedoksi. Web-palvelutekniikoissa tämän toiminnallisuuden mahdollistaa SOAP, jonka myötä on mahdollista toteuttaa implementaatioita jotka huolehtivat siitä, että XML-sanoma kääntyy automaattisesti kulloisenkin järjestelmän ymmärtämään muotoon. (Aloisio ym. 2003)

Web-palvelutekniikoiden ongelmana integraatiotekniikkana voidaan pitää sitä, että vaikka HTTP-protokollan käytöllä on omat vahvuutensa, ei se ole tietyiltä osin paras mahdollinen vaihtoehto. HTTP on alun perin suunniteltu tilattomaan synkroniseen dokumenttien hakemiseen verkkoselaimella, eikä se myöskään ole protokollana kovin luotettava sovellusten välisessä viestinnässä. SOAP/HTTP – yhdistelmän perusominaisuudet eivät siis ole aina riittäviä puhuttaessa luotettavasta sanomien perille toimituksesta (Endrei ym. 2004). HTTP – protokollan perusominaisuuksista johtuen viestien asynkronisuutta, sekä luotettavaa perillemenoa ei voida toteuttaa tiedonsiirtotasolla, vaan nämä tulee huolehtia jollain toisella tavalla. Tähän puutteeseen onkin pyritty vastaamaan kehittämällä OASIS – konsortion hyväksymä WS-Reliable Messaging – standardi. Standardi sisältää kuvaukset neljästä (At Most Once, At Least Once, Exactly Once ja In Order) erilaisesta semantiikasta, joita implementoimalla voidaan taata sanomien halutunlainen perille toimitus. (OASIS 2007)

Web-palvelutekniikoihin liittyvänä heikkoutena voidaan pitää myös sitä, että niihin liittyviä spesifikaatioita on kehitetty lukuisia, lukuisten eri osapuolten toimesta ja osittain päällekkäinkin. Tämä on hidastanut tekniikoiden kypsymistä ja laajamittaista hyväksyntää. Useat eri spesifikaatiot aiheuttavat ongelmia

myös siksi, ettei ole mitään selkeää ohjetta sille, mitä spesifikaatioita tulisi noudattaa palvelukeskeisen arkkitehtuurin toteuttamiseksi. (Nevala 2008)

Toistaiseksi web-palvelutekniikoita ja sanomanvälitystä on pääsääntöisesti pidetty toisistaan erillään olevina integrointitekniikkoina ja ohjelmointimalleina. Web-palvelutekniikoiden hyödyntäminen onkin pääsääntöisesti keskittynyt palvelin/asiakas -ratkaisuihin, joissa on käytetty synkronista HTTP-protokollan yli tapahtuvaa pyyntö/vastaus -vuorovaikutusmallia. Luotettava sanomanvälitys, joka takaa sanomien lähetyksen myös ongelmatilanteissa on kuitenkin oleellinen ja kriittinen osa suuressa osassa yritysjärjestelmiä. Näin ollen yhä useammin sovellusintegraatioissa onkin luontevinta käyttää jo olemassa olevaa viestipohjaista väliohjelmistoarkkitehtuuria yhdistettynä web-palvelutekniikoihin. (Silva-Lepe ym. 2006)

Näiden kahden integraatiotekniikan yhdistämiseen liittyen merkittävin palvelupohjaiseen arkkitehtuuriin läheisesti liittyvistä tekniikoista on väylätyyppinen ratkaisu ESB (Enterprise Service Bus). ESB on käytännössä viestinvälitysmoottori, joka yhdistää osaltaan web-palvelutekniikat ja viestipohjaiset väliohjelmistot. Se hoitaa sovellusten sanomien ja transaktioiden siirrot, ja tarjoaa myös muita sanomien välitykseen liittyviä ominaisuuksia kuten sanoman muuntaminen ja lokin keräys. ESB:n käyttö mahdollistaa myös sanomien reitityksen sisällön, asian tai vaikkapa bisnesprosessien mukaan. Koska ESB hoitaa itse monia perustehtäviä, ei tällaisia toimintoja tarvitse ohjelmoida erikseen ohjelmakomponentteihin. (Rao ym. 2006)

5.4 Yhteenveto eri integraatiotekniikoista

Alla olevaan taulukkoon (Taulukko 2.) on koottu tässä tutkielmassa esille tulleita neljän eri integraatiotekniikan keskeisimpiä piirteitä, sekä niiden hyviä ja huonoja puolia. Taulukko on pyritty rakentamaan niin, että se helpottaisi osaltaan tutkielmassa läpikäytyjen tekniikoiden vertailua, ja voisi näin ollen

toimia ohjenuorana mietittäessä eri integraatiotekniikoiden soveltuvuutta eri tilanteisiin.

Taulukko 2. Eri integraatiotekniikoiden ominaisuuksia, vahvuuksia ja heikkouksia

Ominaisuus	RPC	MOM	OOM	Web Service
Kommunikointi	Kutsu/vastaus, Luontaisesti synkronista	Sanomajonoin, Luontaisesti asynkronista	Palvelukutsuin, Luontaisesti synkronista, mutta Asynchronous Method Invocation mahdollistaa asynkronisuuden	SOAP, Synkronista tai asynkronista riippuen käytettävästä tietoliikenne-protokollasta
Komponenttien kytkökset	Tiiviisti kytkettyjä. Komponenttien tulee tietää toistensa etämetodit	Sovellukset ovat löyhästi toisiinsa kytkettyjä	Implementaatiot koostuvat yleensä tiiviisti toisiinsa kytketyistä komponenteista	Sovellukset ja jopa niiden komponentit ovat löyhästi toisiinsa kytkettyjä
Sanomien perilletoimitus	Takaa sanomien perille toimituksen (At most once)	Takaa sanomien perille toimituksen (At most once) Takaa ettei duplikaatteja sanomia (exactly once)	Takaa sanomien perille toimituksen (At most once) Takaa ettei duplikaatteja sanomia lähetetä (exactly once)	WS-Reliable Messaging – standardi sisältää neljä kuvausta joilla taataan sanomien perille toimitus (At Most Once, At Least Once, Exactly Once ja In Order)
Vikasietoisuus	Heikko vikasietoisuus	Hyvä vikasietoisuus, koska sanomat voidaan tallettaa pysyvästi jonoihin, jotka sijaitsevat palvelimen kovalevyllä	Palveluita jotka parantavat vikasietoisuutta	Hyvä, mutta HTTP-protokolla ei ole kovin luotettava sovellusten välisessä viestinnässä, ja viestien asynkronisuutta ja luotettavaa perille menoa ei voida toteuttaa tiedon-siirtotasolla
Kuormantasaus, lokien keräys ym.	Heikko kuormantasaus	Sanomanvälitys-palvelimen käyttö mahdollistaa sanomamuunnokset, filtteröinnin ja lokituksen	Palveluita jotka parantavat kuormantasausta	ESB mahdollistaa mm. lokien keräämisen ja sanomien älykkään reitittämisen

Yhteensopivuus ja siirrettävyys	Tukee useita ohjelmointikieliä ja käyttöjärjestelmiä Mahdollisuus tehdä etäkutsuja paikallisesti Tekniikkana ohjelmoijalle läpinäkyvää	Huono siirrettävyys ja yhteensopivuus, koska ei yhteistä standardia Väliohjelmisto ei tiedä sanoman formaattia, toisaalta ohjelmistokomponenttien ei tarvitse olla yhteensopivia kuin sanomanvälityksen keinoin	Tukee olioperustaisia ohjelmointikäsitteitä Tukee useita ohjelmointikieliä	XML-kielestä johtuen riippumaton alustasta ja ohjelmointikielestä Perustuu avoimiin standardeihin Tukee useita protokollia (mm. http, https ja smtp) Lukuisat eri spesifikaatiot ovat aiheuttaneet ongelmia ja hidastaneet tekniikan leviämistä HTTP-protokolla on suunniteltu alkujaan tilattomaan synkroniseen dokumenttien hakemiseen verkko-selaimella
Skaalautuvuus	Heikko skaalautuvuus	Rajattu skaalautuvuus, mutta publish-subscribe malli parantaa sitä	Heikko skaalautuvuus	Toteutustavasta riippuen mahdollisuus hyväänkin skaalautuvuuteen
Marshalling/Unmarshalling	Automaattista	Ei ole automaattista, ohjelmoijan pitää implementoida se itse	Automaattista	SOAP mahdollistaa automatiikan
Mihin ei sovi?	Laajoihin yritysjärjestelmiin, joille on tärkeää hyvä suorituskyky ja luotettavuus.	Ei sovellu tilanteisiin joissa sovelluksen pitää saada vastaus palvelimelta joissain tietyissä aikarajoissa	Ovat usein melko raskaita ja staattisia, ja sopivat huonosti tekniikaksi jos niiden aikaansaama komponenttien toisiinsa kytkeytyminen koetaan ongelmalliseksi	Sovelluksiin joissa http-protokollan mukanaan tuomat ongelmat koetaan liian suuriksi
Mihin sopii?	Pieniin ja yksinkertaisiin sovelluksiin	Korkeaa luotettavuutta, monitorointia ja asynkronisuutta vaativiin järjestelmiin	Olioperustaisten järjestelmien integrointiin	Kustannustehokas ja nopea tapa toteuttaa lähes kaikenlaista sovellusintegraatioita

6 YHTEENVETO JA POHDINTA

Tässä tutkielmassa tutustuttiin neljään eri organisaatioiden järjestelmäintegraatioissa käytettävään malliin; viestipohjaisiin väliohjelmistoihin, etäproseduurikutsuihin, objektipohjaisiin väliohjelmistoihin ja web-palvelutekniikoihin. Tutkielmassa käytiin läpi edelle mainittujen väliohjelmistojen perusominaisuuksia, sekä selvitettiin niiden hyviä ja huonoja puolia liittyen yritysten järjestelmäintegraatioon. Tutkielma on jaoteltu niin, että siinä on keskitytty kuvaamaan syvimmin viestipohjaisia väliohjelmistoja, ja pyritty sitten vertaamaan niitä kolmeen muuhun toteutusmalliin.

Tutkielmassa esiteltiin eri integraatiotekniikoita ja niiden ominaisuuksia, sekä niiden käyttöön liittyviä haasteita ja mahdollisuuksia varsin yleisellä tasolla. Tarkoituksena oli luoda perusnäkemys eri väliohjelmistomallien ominaisuuksista, sekä heikkouksista ja vahvuuksista toimia yritysjärjestelmien integraatiotekniikkoina. Vertailtaessa eri integraatiotekniikoita suurimpina eroavaisuuksina nousivat esille niiden luontainen tapa kommunikoida (synkronisuus vastaan asynkronisuus), sekä se kuinka tiukasti ohjelmistokomponentit sitoutuvat toisiinsa eri tekniikoita käytettäessä. Merkittävä ero ohjelmistojen välillä on myös siinä, mitä metodia ne käyttävät lähettäessään sanomia sovellusten välillä (etäkutsu, sanomajonot, palvelukutsut, SOAP). Eri väliohjelmistojen standardit eroavat myös suuresti toisistaan, ja tutkielman myötä kävi hyvin selväksi, että eri tekniikat soveltuvat perusominaisuuksiltaan hyvin erilaisiin käyttötarkoituksiin.

Integraatiotekniikkoihin ja väliohjelmistoihin liittyvä kirjallisuus on hyvin laajaa ja monipuolista. Aiheena eri integraatiotekniikoiden vertaileminen yleisellä tasolla oli kuitenkin erittäin vaikeaa ja aikaa vievää. Siksi aiheeseen liittyvä jatkotutkimus tulisi rajata huomattavasti selkeämmin koskettamaan esimerkiksi jonkun tietyn integraatiotekniikan soveltumista tietylle toiminta-alalle. Muita hyviä jatkotutkimusaiheita aiheeseen liittyen voisivat olla AMQP-protokolla tai

web-palvelutekniikoiden ja viestipohjaisten väliohjelmistojen yhdistäminen. Myös aiheeseen liittyvää kirjallisuutta tulisi arvioida laajemmin ja systemaattisemmin, jotta saavutettaisiin tarkempia ja yksityiskohtaisempia tutkimustuloksia.

LÄHDELUETTELO

Aihkisalo T. & Välitalo P., 2008. A Comparative Analysis of the Resource Consumption in Message Oriented Middleware with XML and Binary Encoded Messages. Proceedings of the 2008 The Fourth International Conference on Wireless and Mobile Communications. 158-166.

Aloisio G., Blasi E., Cafaro M., Fiore S., Lezzi D. & Mirto M., 2003. Web Services for a Biomedical Imaging Portal. Proceedings of the International Conference on Information Technology: Computers and Communications. 432-436.

Banavar G., Chandra T., Strom R. & Sturman D., 1999. A Case for Message Oriented Middleware. Lecture Notes In Computer Science Vol. 1693, 1-18.

Bernstein P. A., 1996. Middleware: A Model for Distributed System Services. Communications of the ACM 39(2), 86-98.

Charles J. 1999. Middleware moves to the forefront. Computer 32(5), 17-19.

Curry E. 2004. Message-Oriented Middleware. Teoksessa Q.H. Mahmoud (toim.) Middleware for Communications, John Wiley & Sons, 1-28.

Cugola G., Migliavacca M. & Monguzzi A., 2007. On Adding Replies to Publish-Subscribe, Proceedings of the 2007 inaugural international conference on Distributed event-based systems, Toronto, 128-138.

Do-Guen Jung, Kwang-Jin Paek & Tai-Yun Kim, 1999. Design of MOBILE MOM: Message Oriented Middleware Service for Mobile Computing, International workshops of parallel processing 434-439.

Emmerich W., Aoyama M. & Sventek J. 2008. The Impact of Research on the Development of Middleware Technology, ACM Transactions on Software Engineering and Methodology 17(4).

Endrei M., Ang J., Arsanjani A., Chua S., Comte P., Krogdahl P., Luo M., Newling T., 2004. Patterns: Service-Oriented Architecture and Web Services [online]. IBM [viitattu 16.3.2009]. Saatavilla www-osoitteesta: <http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf>

Eugster P., Felber P., Guerraoui R. & Kermarrec A-M. 2003. The Many Faces of Publish/Subscribe, ACM Computing Surveys 35(2), 114-131.

Hadim S., Al-Jaroodi J. & Mohamed N., 2006. Middleware Issues and Approaches for Mobile Ad hoc Networks, 3rd Consumer Communications and Networking Conference, Jan 8-10, 431-436.

Idlbek R., Hip O. & Mustapic Z. 2005. Integration of Distributed Applications in Message Oriented Environment, Carnet User Conference 2005, November 21-23.

Issarny V., Caporuscio M. & Georgantas N. 2007. A Perspective on the Future of Middleware-based Software Engineering. Future of Software Engineering (FOSE '07), May 23 – 25. 244-258.

Menascé D. A., 2005. MOM vs. RPC:Communication Models for Distributed Applications, Internet Computing 9(2), 90-93.

Molina-Jimenez C., Shrivastava S., & Cook N., 2007. Implementing Business Conversations with Consistency Guarantees Using Message-Oriented Middleware. 11th IEEE International Enterprise Distributed Object Computing Conference, Oct 15-19, 51-62.

Nevala P., 2008. Web Services -tekniikoiden hyödyntäminen Java Platform, Micro Edition -ympäristössä. Jyväskylän yliopisto. Tietojärjestelmätieteen pro gradu -tutkielma.

OASIS, 2007. Web Services Reliable Messaging standard (WS-1 ReliableMessaging) Version 1.1. [online]. [viitattu 13.3.2009] Saatavilla

www-osoitteesta: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.pdf

O'Hara J., 2007. Toward a Commodity Enterprise Middleware, QUEUE 5(4), 48-55.

Pinus H. 2004. Middleware: Past and Present a Comparison [Online]. [viitattu 20.1.2009]. Saatavilla osoitteessa: <http://userpages.umbc.edu/~dgorin1/451/middleware/middleware.pdf>.

Pääkkönen P., Pakkala D. ja Sihvonen M. 2005. An Optimized Message-Oriented Middleware Solution for Extending Enterprise Services to the Mobile Domain. Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS-ICNS 2005), Papeete, 23-28 Oct.

Rao F. Y., Fang R., Tian Z., Lane E., Srinivasan H., Banks L. & He L., 2006. Cache Mediation Pattern. 11th European Conference on Pattern Languages of Programs, Irsee, 5-9 July. [viitattu 20.1.2009]. Saatavilla internetistä: <http://hillside.net/europlop/europlop2006/workshops/A5.pdf>.

Schmidt D. C. & Vinoski S., 1999. Programming Asynchronous Method Invocations with CORBA Messaging, C++ Report 11(2), 58-62.

Silva-Lepe I., Ward M. J. & Curbera F., 2006. IntegratingWeb Services and Messaging. Proceedings of the IEEE International Conference on Web Services. 111-118.

Tai S., Mikalsen T. A. & Rouvellou I. 2003. Using Message-oriented Middleware for Reliable Web Services Messaging. Teoksessa C. Bussler (toim.) Web Services, E-Business, and the Semantic Web, LNCS 3095, Springer Berlin/Heidelberg, 89-104.

Tai S., Mikalsen T. A., Rouvellou I. & Sutton Jr. S. M. 2001. Dependency-Spheres: A Global Transaction Context for Distributed Objects and Messages. 5th International Enterprise Distributed Object Computing Conference (EDOC 2001), Seattle, 4-7 September, 105 – 115.

Tunturi J., 2004, Mobiili Java hajautetuissa mittaus- ja ohjausjärjestelmissä, Tampereen teknillinen yliopisto, Tietotekniikan diplomityö.