Jaakko Pallari

AMQP:n hyödyntäminen mobiileissa hajautetuissa järjestelmissä

Tietotekniikan
kandidaatintutkielma
20. joulukuuta 2012

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Jaakko Pallari
Yhteystiedot: jaakko.j.pallari@student.jyu.fi
Työn nimi: AMQP:n hyödyntäminen mobiileissa hajautetuissa järjestelmissä
Title in English: Using AMQP in mobile distributed systems
Työ: Tietotekniikan kandidaatintutkielma
Sivumäärä: 24
Tiivistelmä: Hajautettujen järjestelmien määrä kasvaa pilvipalveluiden ja virtualisoinnin suosion myötä. Kommunikointi hajautetuissa järjestelmissä tapahtuu tyypillisesti erilaisten väliohjelmien avulla. Mobiilien laitteiden yleistyessä tulee ottaa huomioon laitteiden vaatimukset uusia kommunikointiratkaisuja toteuttaessa. AMQP on eräs sovellusprotokolla, jonka avulla ohjelmat voivat kommunikoida keskenään viesteillä. Tässä tutkielmassa tarkastellaan mobiilien hajautettujen järjestelmien haasteita, ja kuinka hyvin AMQP:lla pystytään ratkaisemaan niitä.
Abstract: The amount of distributed systems increases along with the popularity of cloud computing and virtualization. The communication in distributed systems typically happens with the help of different kinds of middleware. As mobile devices become more common, the requirements of devices have to be taken into account when new communication solutions are implemented. AMQP is an application layer protocol. With the help of AMQP, programs can communicate with each other using messages. In this thesis, we examine the challenges of mobile distributed systems, and how well AMQP can solve those challenges.
Avainsanat: AMQP, hajautetut järjestelmät, väliohjelma
Keywords: AMQP, distributed systems, middleware

Esipuhe

Tutustuin ensimmäistä kertaa AMQP:aan työelämässä, johon perustin tutkimuksen aiheen valinnan. Organisaatiossa, jossa olin töissä, oli hyödennetty AMQP:aa jo aiemmin muun muassa laskentatyön jakamiseen tietokoneiden kesken. Kokemusta AMQP:sta minulle kertyi työni kautta, jossa pääsin sekä suunnittelemaan että toteuttamaan viestintäratkaisua AMPQ:lla. Erityisesti kiinnostuin siitä kuinka monenlaisia sovellusten välisen viestinnän ongelmia pystytään ratkomaan hyvin pitkälle pelkästään yhdistelemällä AMQP:n työkaluja. Tästä lopulta syntyi idea tutkia kuinka AMQP soveltuu eri ongelmiin.

Sanasto

AMQP Advanced Message Queuing Protocol. Viestijonomallia mukaileva sovellusprotokolla sovellusten väliseen viestintään.

middleware väliohjelma. Ohjelma joka tarjoaa tiedon välittämistä muiden ohjelmien välillä.

SOA palvelukeskeinen arkkitehtuuri (Service Oriented Architecture). Useista standardeihin pohjautuvista, itsenäisistä sovelluspalveluista koostuva järjestelmä.

konteksti Mikä tahansa ohjelman ympäristön ominaisuus, joka vaikuttaa ohjelman suoritukseen. Esimerkiksi verkkoyhteyden nopeus, lokaalisaatio tai laitteen fyysinen sijainti.

Sisältö

Esipuhe **i**

Sanasto **ii**

1 Johdanto **1**

2 Väliohjelman määritelmä ja ominaisuudet **2**
2.1 Laskennallinen kuorma 3
2.2 Kommunikaatiotavat 3
2.3 Konteksti ja sen esittäminen väliohjelmilla 4
2.4 Transaktiot 4

3 Väliohjelmamallit **5**
3.1 Etäkutsut ja hajautetut oliomallit 5
3.2 Muistin jakaminen väliohjelmilla 6
3.3 Viestikeskeinen väliohjelma 7
3.4 Palvelukeskeinen väliohjelma 8

4 AMQP **9**
4.1 Työkalut 10
4.2 Esimerkkejä 11
4.3 AMQP 1.0 13

5 AMQP mobiileissa hajautetuissa järjestelmissä **14**
5.1 Mobiilien hajautettujen järjestelmien ominaisuuksia 14
5.2 Väliohjelman keveys 15
5.3 Asynkronisuus ja irralliset liitokset 15
5.4 Kontekstit mobiileissa ympäristöissä 16
5.5 Palvelujen löydettävyys 17

6 Yhteenveto **17**

Viitteet **18**

1 Johdanto

Nykyaikana eri organisaatioissa voi olla yhtäaikaa käytössä useita kymmeniä eri sovellusjärjestelmiä. Uudelleenkäytettävyyden ja uusien ominaisuuksien saavuttamiseksi järjestelmien tulisi pystyä jakamaan dataa keskenään ilman, että käyttäjä tekee sen itse käsin. Sovellusten välinen kommunikointi korostuu erityisesti erilaisissa hajautetuissa järjestelmissä kuten pilvipalveluissa ja virtuaalisoiduissa ympäristöissä, koska niissä sovellukset eivät kykene jakamaan muistia suoraan keskenään vaan joutuvat turvautumaan viestimiseen esimerkiksi verkon välityksellä.

Viestintään liittyvät ongelmat ovat yhä suuremmat mobiileissa hajautetuissa järjestelmissä, joissa laitteen ei voida olettaa pysyvän pitkiä aikoja yhteydessä verkkoon. Samoin myöskään ei voida olettaa mobiilien laitteiden säilyvän samassa sijainnissa tietoverkossa. Luotettavan tiedonsiirron lisäksi mobiililaitteiden olisi myös kyettävä vastaanottamaan kontekstiriippuvaista dataa: laite vastaanottaa tietynlaista informaatiota riippuen esimerkiksi sen verkkoyhteyden tyypistä tai laitteen fyysisestä sijainnista.

Kommunikointi sovellusten välillä on perinteisesti hoidettu “middleware” -ohjelmilla eli väliohjelmilla. Väliohjelma käsitteenä tarkoittaa ohjelmaa, joka tarjoaa toisille sovelluksille palvelua, johon käyttöjärjestelmät eivät suoraan itse kykene. Väliohjelmia voidaan siis ajatella eräänlaisiksi “silloiksi” sovellusten välille. Väliohjelman käsite on erittäin laaja, eikä se ei ota itsessään mitään kantaa toteutukseen.

Perinteisesti väliohjelmat eri sovellusten väleihin on hankittu joko suoraan ohjelman valmistajalta tai joltain kolmannelta osapuolelta. Näin väliohjelman laajennus ja ylläpito on täysin ohjelman valmistajan käsissä, joka vaikeuttaa vanhojen järjestelmien päivitystä ja uusien järjestelmien käyttöönottoa. Yleensä tällaiset väliohjelmat tarjoavat ratkaisua vain muutaman sovelluksen välille, jolloin ulkopuolelle jääville sovelluksille tarvitaan lisää väliohjelmia.

Väliohjelmien toteuttamista varten on kehitetty erilaisia malleja, jotka kuvaavat sovellusten välistä viestintää abstraktioina. Abstraktiot auttavat ymmärtämään kuinka sovellukset toimivat keskenään ilman, että tarvitsee ymmärtää järjestelmän yksityiskohtia. Mallit eivät ota kantaa toteutukseen, jolloin useimmat samaa mallia käyttävät väliohjelmat ovat harvoin yhteensopivia keskenään. Mallit eivät myöskään takaa viestiprotokollien avoimuutta. Useimmat kaupalliset väliohjelmat paljastavat vain rajapintoja niiden käyttöön, joka hankaloittaa esimerkiksi oman yhteensopivan toteutuksen luomista. Lisäksi kaupalliset väliohjelmat ovat usein hyvin vahvasti lisenssoituja, jolloin yhteensopivien ohjelmien levitys voi olla rajoitettu, tai

jopa kokonaan estetty, lisenssiehdoissa.

On myös olemassa tarjolla valmiita avoimia työkaluja, joiden avulla on mahdollista kehittää viestintäkeinoja sovellusten välille. Advanced Message Queueing Protocol (AMQP) on eräs avoin sovellusprotokolla, jota on ehdotettu yleiseksi väliohjelmaratkaisuksi. Sen uutuudesta huolimatta useat yritykset ovat jo soveltaneet AMQP:aa omiin tarkoituksiinsa, minkä vuoksi siitä on myös vähän tutkimusta saatavilla.

AMQP on valmis avoin sovellusprotokolla, jonka vuoksi sillä on valmiudet menestyä teollisena standardina sovelluskehityksessä. Jotta AMQP:aa kyettäisiin soveltamaan kunnolla, on tiedättävä ensin mihin se kykenee ja ei kykene. Näin voidaan myös tutkia voidaanko toteuttaa sellaisia ratkaisuja, jotka soveltuvat mobiileihin hajautettuihin järjestelmiin.

Tutkielman luvussa 2 tarkastellaan mitä väliohjelma yleisesti tarkoittaa, ja millaisia ominaisuuksia ne yleensä sisältävät. Luvussa 3 tutustutaan erilaisiin väliohjelmamalleihin. Luku 4 käsittelee AMQP:aa yleisesti. Luvussa 5 tarkastellaan mobiilien hajautettujen järjestelmien ominaisuuksia, ja kuinka AMQP:aa voidaan soveltaa niissä.

2 Väliohjelman määritelmä ja ominaisuudet

Väliohjelma tarkoittaa ympäristöstä riippumatonta sovellusta tai sovelluskerrosta, joka tarjoaa muille sovelluksille tapoja toimia keskenään. Tämä tapahtuu abstrahoimalla toiminnan suorittamiseen liittyvät monimutkaisuudet väliohjelman hoidettavaksi (Li & Zhou, 2010, s. 83). Väliohjelma on siis ohjelma, joka niin sanotusti sitoo ohjelmat toisiinsa. Esimerkiksi käyttöjärjestelmä voitaisiin ymmärtää eräänlaisena väliohjelmana sovellusten ja laitteen välillä. Toisaalta WWW-sovelluksen verkkorajapintaa ei voida tulkita väliohjelmaksi, koska se on riippuvainen sovelluksesta johon se on kytketty.

Tässä tutkielmassa väliohjelmalla tarkoitetaan ohjelmaa, jolla yhdenmukaistetaan sovellusten välinen toiminta hajautettujen järjestelmien laitteiden välillä (Issarny ym., 2007). Yhdenmukaistaminen tapahtuu abstrahoimalla järjestelmälle keskeiset ominaisuudet taakseen (Li & Zhou, 2010, s. 83). Käytännössä tällainen väliohjelma tarjoaa muille sovelluksille tapoja toimia keskenään tyypillisesti tietoverkkojen välityksellä.

Capra ym. (2001) ovat tarkastelleet väliohjelmien kolmea ominaisuutta: lasken-

nallista kuormaa, kommunikaatiotapaa ja kontekstin esittämistä. Heidän mukaansa jokainen ominaisuus voidaan yhä jakaa siihen millä tavalla ominaisuus on toteutettu väliohjelmassa. Tässä luvussa tarkastellaan näitä kolmea ominaisuutta ja lisäksi transaktioita.

2.1 Laskennallinen kuorma

Laskennallisella kuormalla tarkoitetaan sitä, kuinka paljon suoritustehoa ja muita resursseja väliohjelma vaatii pääteohjelmilta. Mobiileissa laitteissa suoritusteho on tyypillisesti muihin laitteisiin verrattuna rajallisempi, minkä vuoksi voidaan joutua alentamaan resursseihin liittyviä vaatimuksia (Capra ym., 2001, s. 10). Resurssivaatimukset voivat sisältää esimerkiksi vaatimuksia muistin määrälle, prosessorin tehokkuudelle tai verkkoyhteyden nopeudelle.

Vaatimukset voivat myös liittyä toiminnallisuuksiin kuten datan synkronisointiin (Capra ym., 2001). Esimerkiksi järjestelmässä voi olla vaatimus, että kaikki palveluun liittyvään dataan tehdyt muutokset tulee synkronoida heti takaisin palveluun. Tällöin siirtonopeuksien on oltava tarpeeksi nopeita, jotta ohjelman käyttäminen ei hidastu liikaa. Toinen vaihtoehto on sallia useita paikallisia kopioita datasta, ja varmistaa, että data synkronoidaan jossain vaiheessa.

2.2 Kommunikaatiotavat

Kommunikaatiotavalla tarkoitetaan sitä, millaisia rajoitteita väliohjelma asettaa ohjelman kulkemiselle, eli toimivatko ohjelmat keskenään synkronisesti vai asynkronisesti. Synkronisessa kommunikaatiossa sekä palvelunpyytäjän että tuottajan on oltava jatkuvassa yhteydessä toisiinsa (joko suoraan tai väliohjelman kautta) toiminnon ajan. Asynkronisessa viestinnässä ohjelmat ohjelmien ei tarvitse olla jatkuvassa yhteydessä (Capra ym., 2001).

Synkroninen tapa kommunikoida on samankaltainen imperatiivisen ohjelmointiparadigman kanssa. Toiminnon aloittavan ohjelman on odotettava toiminnon käsittely loppuun asti, ennen kuin se voi jatkaa muun ohjelman suorittamista (Geihs, 2001, s. 27). Samalla tavalla imperatiivisissa ohjelmissa yksittäinen käsky on suoritettava loppuun saakka, ennen kuin voidaan siirtyä seuraavaan käskyyn.

Toisaalta samalla myös kommunikoinnin rinnakkaistaminen muun ohjelman kanssa synkronisessa kommunikoinnissa on tehtävä samoilla menetelmillä kuin muun ohjelman rinnakkaistaminen. Näitä menetelmiä ovat muun muassa säikeet ja rin-

nakkaiset prosessit (Geihs, 2001, s. 27).

Asynkronisessa kommunikoinnissa ohjelma voi tarvittaessa jatkaa ohjelman suoritusta toiminnon aloituksen jälkeen. Ohjelma voi siten käsitellä toiminnosta saadut tulokset heti, kun ne saapuvat takaisin tai ne ovat jossain luettavissa. Se millä tapaa ohjelma saa tulokset riippuu väliohjelmasta. Ohjelman ei siis tarvitse itse hoitaa kommunikaation rinnakkaistamista.

2.3 Konteksti ja sen esittäminen väliohjelmilla

Kontekstitietoisuudella väliohjelmissa tarkoitetaan väliohjelman kykyä välittää vallitsevasta kontekstista riippuvaa dataa väliohjelmalle (Bellavista ym., 2003). Suorituskontekstilla tarkoitetaan mitä tahansa elementtiä, mikä vaikuttaa sovelluksen käyttäytymiseen sovelluksen suorituksen aikana (Capra ym., 2001, s. 5). Konteksti voi sisältää tietoa muun muassa pääteohjelman tietoliikenneyhteyden ja laitteen tyypistä tai lokaalisaatiosta.

Capra ym. (2001) esittävät kaksi tapaa, joilla väliohjelmat voivat esittää ohjelmien konteksteja toisille ohjelmille. Väliohjelmat voivat kerätä tietoa pääteohjelmien suorituskonteksteista, ja käyttää tätä tietoa hyväksi väliohjelman sisällä. Tällöin tyypillisesti kontekstia ei välitetä enää pääteohjelmille, eli ohjelmien kontekstit ovat läpinäkyviä toisilleen.

Toinen tapa esittää konteksteja on välittää kaikki kontekstidata pääteohjelmille. Tällöin väliohjelmat tyypillisesti eivät itse käytä suorituskonteksteja, vaan siirtävät vastuun pääteohjelmille (Capra ym., 2001).

2.4 Transaktiot

Väliohjelmat voidaan vielä jakaa niihin jotka pystyvät transaktioihin ja niihin jotka eivät pysty. Transaktiot ovat useista operaatioista koostuvia atomisia operaatioita. Niillä voidaan varmistaa, että ohjelmien välillä tapahtuva tilamuutos tapahtuu täysin yhtenäisesti (Issarny ym., 2007).

Transaktioita voidaan hyödyntää tilanteissa, joissa ohjelman tapahtuma sisältää useita muutoksia joiden kaikkien on suoriuduttava onnistuneesti. Tällöin tapahtumassa kaikki muutokset onnistuvat tai yhtäkään muutosta ei tehdä. Transaktiokeskeiset väliohjelmat voivat olla joko synkronisia tai asynkronisia riippuen väliohjelman muusta toteutuksesta (Li & Zhou, 2010, s. 84).

3 Väliohjelmamallit

Väliohjelmien toimintaa voidaan hahmotella erilaisilla malleilla. Mallit kertovat abstraktilla tasolla millä tavoilla interaktio ohjelmien välillä tapahtuu väliohjelman kautta. Lajittelemalla väliohjelmat malleihin voidaan arvioida jokaisen mallin soveltuvuutta ongelman ratkaisemiseen ilman, että tarvitsee erikseen tarkastella jokaisen väliohjelmatoteutuksen ominaisuuksia.

Tässä luvussa tarkastellaan muutamia yleisiä väliohjelmamalleja. Ensimmäiseksi esitellään etäkutsut ja hajautetut oliomallit. Sen jälkeen tarkastellaan väliohjelmamallia, jossa kommunikaatio ohjelmien välillä tapahtuu muistia jakamalla. Luvussa tarkastellaan lisäksi viesti- ja palvelukeskeisiä väliohjelmia.

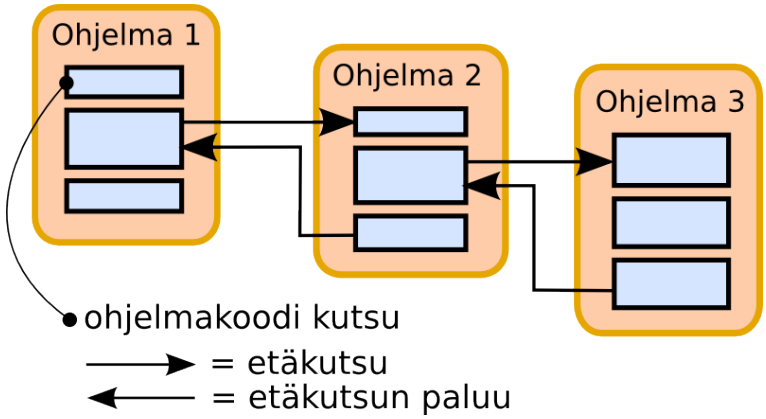
3.1 Etäkutsut ja hajautetut oliomallit

Etäkutsut (Remote Procedure Call, RPC) on väliohjelmamalli, joka tarjoaa ohjelmille mahdollisuuden kutsua toistensa aliohjelmia (Issarny ym., 2007). RPC:ssä ohjelmat avaavat itselleen rajapinnan, johon väliohjelmat voivat välittää saapuvia etäkutsuja. Kutsut välitetään aliohjelmille, ja aliohjelmista saadut tulokset palautetaan väliohjelmalle. Väliohjelma välittää saadun tuloksen takaisin kutsun aloittaneelle ohjelmalle. Vaikka etäkutsuja varten on olemassa useita väliohjelmia, jotka hallinnoivat sovellusten välistä viestintää, etäkutsut voidaan toteuttaa myös suoraan päästä päähän ilman väliohjelmaa.

Hajautetussa oliomallissa (Distributed Object Middleware, DOM) ohjelmat luovat olioita, joita muut ohjelmat pystyvät kutsumaan kuin ne olisivat oman ohjelman olioita (Li & Zhou, 2010, s. 84). Issarny ym. (2007) mukaan oliokeskeisissä väliohjelmissa jatketaan etäkutsumallia tuomalla mukaan olio-ohjelmoinnille ominaisia piirteitä. DOM voidaan siis käsittää etäkutsujen laajennuksena.

Etäkutsumallia mukailevat väliohjelmat ovat tyypillisesti synkronisia (Li & Zhou, 2010, s. 83), eli väliohjelmaa käyttävät ohjelmat kommunikoivat keskenään synkronisesti. Koska etäkutsuissa kutsuvan ja kutsuttavan ohjelman täytyy olla yhteydessä etäkutsun suorituksen ajan, voidaan ohjelmat ajatella olevan toisistaan riippuvaisia. Kuvan 1 esimerkissä ensimmäinen ohjelma etäkutsuu toista ohjelmaa, joka edelleen kutsuu kolmatta ohjelmaa. Jokaisen ohjelman on odotettava oman etäkutsunsa kohdalla vastauksen palaamista, ennen kuin ne pystyvät jatkamaan ohjelman suoritusta.

Etäkutsuja sovelletaan pääasiassa tilanteissa, joissa järjestelmät on hajautettu pääte–



Kuva 1: Etäkutsujen tapahtuminen ketjussa ohjelmalta ohjelmalle.

palvelin-mallin mukaisesti (Geihs, 2001, s. 26–27), eli pääteohjelman on tehtävä etäkutsu palvelinohjelmalle. Näissä tilanteissa palvelinohjelma on käynnissä jatkuvasti, jolloin voidaan hyväksyä pääteohjelman riippuvaisuus palvelinohjelmasta.

3.2 Muistin jakaminen väliohjelmilla

Eräs tapa välittää dataa ohjelman sisällä säikeeltä säikeelle on jakaa muistia. Tätä mallia voidaan soveltaa myös sovellusten välillä tuomalla mukaan väliohjelma, joka ylläpitää jaettavaa muistia ja siihen pääsyä. Tällaisia väliohjelmia ovat muun muassa monikkoavaruuspohjaiset (tuple space) väliohjelmat, jotka jakavat monikkopohjaista tietorakennetta asiakasohjelmien kesken.

Monikkoavaruutta käyttävät ohjelmat ovat toisistaan täysin riippumattomia, minkä vuoksi ohjelmien ei tarvitse toimia samaan aikaan (Issarny ym., 2007). Esimerkiksi eräs ohjelma voi syöttää dataa monikkoavaruuteen säilöön, jota muut ohjelmat voivat käydä lukemassa tai muokkaamassa myöhemmin.

Tällaiset väliohjelmat ovat siis luonteeltaan asynkronisia. Koska avaruuteen tehdyt muutokset eivät vaikuta suoraan asiakasohjelmiin (Geihs, 2001, s. 28), ohjelmat saavat tietää uusista muutoksista vasta tarkastellessaan avaruutta. Jotta ohjelmat kykenisivät paremmin reagoimaan muutoksiin, väliohjelmaan tarvitaan jonkinlainen mekanismi välittämään ilmoituksia (Geihs, 2001, s. 28). Monikkoavaruuden vapaamuotoisuuden vuoksi on vaikea päättää mistä muutoksista pitäisi ilmoittaa ja mille ohjelmille.

Muita monikkoavaruuspohjaisten väliohjelmien johdannaisia ovat muun muassa oliokeskeiset avaruudet, joissa data talletetaan väliohjelmassa olioina. Myös olio-

ja relaatiopohjaisia tietokantoja voidaan soveltaa muistin jakamiseen, mikäli niihin on olemassa yhteiset rajapinnat ohjelmia varten.

Monikkoavaaruutta voidaan hyödyntää esimerkiksi ohjelmien työnjakamisessa. Ohjelmat voivat tallettaa avaruuteen uusia töitä (esim. laskentaa vaativia töitä) avaruuteen esimerkiksi sovitulla työavainsanalla.

Työohjelmat (workers) voivat hakea avaruudesta työavainsanan perusteella suoritettavia töitä. Kun uusi työ löytyy, työohjelma poistaa työn avaruudesta, suorittaa työn ja palauttaa tuloksen avaruuteen esimerkiksi sovitulla tulosavainsanalla. Näin työt jakautuvat jokaiselle työohjelmalle tasaisesti siten, että jokainen työohjelma suorittaa kerrallaan yhtä työtä.

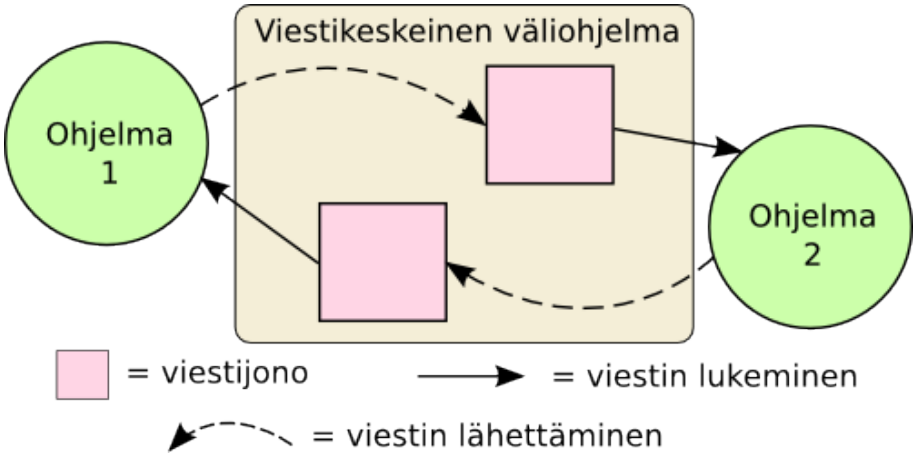
3.3 Viestikeskeinen väliohjelma

Li & Zhou (2010) määrittelevät viestikeskeiset väliohjelmat (Message-oriented Middleware, MOM) ohjelmiksi, joiden avulla asiakasohjelmat voivat kommunikoida lähettämällä viestejä. Vastaanottaja voi tarvittaessa lähtettää tuloksia takaisin lähettäjälle viesteinä. Koska ohjelman ei tarvitse jäädä odottamaan vastausta, voidaan viestikeskeiset väliohjelmat lajitella asynkronisiin väliohjelmiin.

Viestikeskeiset väliohjelmat sisältävät viestijonoja, joista ohjelmat pääsevät lukemaan viestejä. Viestit siis säilyvät väliohjelman muistissa siihen asti kunnes jokin ohjelma on ne käsitellyt, ja siten ohjelman ei tarvitse olla jatkuvassa yhteydessä väliohjelmaan vastaanottaakseen viestejä. Kuvan 2 esimerkissä kaksi ohjelmaa lähettävät viestinsä toistensa viestijonoihin, joista ohjelmat pääsevät tarvittaessa lukemaan viestit.

Väliohjelmat voidaan tulkita monikkoavaruusväliohjelmiksi, joissa avaruuden monikot koostuvat viestijonon kaltaisista tietorakenteista (Issarny ym., 2007). Koska tietorakenteet ovat selvästi rajoittuneempia tavalliseen monikkoavaruuteen verrattuna, on myös helpompi rakentaa viestikeskeiseen väliohjelmaan tapoja välittää ilmoituksia ohjelmille uusien viestien saapumisesta.

Viestikeskeiset väliohjelmat voidaan lajitella viestinvalintamekanismin perusteella jonopohjaisiin ja julkaisu–tilaus-väliohjelmiin (Issarny ym., 2007). Jonopohjaisissa väliohjelmissa viestit välitetään itse viestijonojen perusteella. Tällaisissa tapauksissa lähettäjä lähettää viestin tietylle viestijonolle. Julkaisu–tilaus-väliohjelmassa viestit välitetään kaikille jotka ovat asettaneet tilauksen tietyn tyyppisille viesteille (Geihs, 2001, s. 27). Esimerkiksi joukko ohjelmia voi olla kiinnostuneita viesteistä, joiden avainsana on “aiheX”. Tällöin avainsanalla “aiheX” väliohjelmalle



Kuva 2: Kahden ohjelman välinen kommunikointi viestikeskeisen väliohjelman kautta.

saapuneet viestit välitetään jokaiselle edellisen joukon ohjelmalle.

3.4 Palvelukeskeinen väliohjelma

Palvelukeskeinen arkkitehtuuri (Service Oriented Architecture, SOA) koostuu useista standardeihin pohjautuvista, itsenäisistä sovelluspalveluista. SOA:ssa palvelut on tarkoitus rakentaa siten, että niitä käytettäessä ei tarvitse yhtään kontekstia (Li & Zhou, 2010), eli palvelut eivät säilytä pääteohjelmien puolesta tilaa. Kontekstittomuus mahdollistaa palveluiden käyttämisen silloin kuin se on tarpeen sen sijaan, että integraatiota varten valmistetaan oma ratkaisu (Al-Jaroodi ym., 2010). Muun muassa WWW-palvelut ovat voidaan käsittää SOA palveluina. Niissä käytetään standardeja menetelmiä palvelun käyttämiseen (URL, HTTP), ja vastaus on standerdeja mukaileva (esim. HTML, XML). Lisäksi jokainen WWW-palveluun kohdistuva kysely voidaan tehdä täysin riippumatta edellisistä kyselyistä, eli palvelu on riippumaton kontekstista.

SOA muodostuu kolmesta osasta: palveluntarjoajasta (service provider), palvelunvälittäjästä (service broker) ja palvelunkäyttäjästä (service requester). Tarjoaja tuottaa käytettävän palvelun. Välittäjä rekisteröi tuotetut palvelut palvelurekisteriin, josta käyttäjät löytävät tarvitsemansa palvelut (Al-Jaroodi ym., 2010). Jos WWW-palvelua voidaan käsitellä palvelun tarjoajana, niin tällöin esimerkiksi hakupalvelut voidaan ymmärtää eräänlaiseksi rekisteriksi. Kun käyttäjän löytää rekisteristä etsimänsä palvelun, se yhdistää suoraan palveluun (Al-Jaroodi ym., 2010). Samalla tapaa WWW-selaimen ei tarvitse enää ylläpitää yhteyttä hakukoneeseen löytäessään

hakemansa palvelun.

Palvelukeskeinen väliohjelma (Service Oriented Middleware, SOM) on palvelukeskeiseen arkkitehtuuriin pohjautuva väliohjelmien malli. SOM:lla on tarkoitus huolehtia SOA-palveluiden käyttöönotosta ja palveluiden koordinoimisesta palvelun tarjoajan, käyttäjän ja välittäjän välillä (Issarny ym., 2007).

SOA ja SOM eivät ota kantaa siihen millä menetelmillä varsinainen viestintä arkkitehtuurin osien välillä tapahtuu. Esimerkiksi WWW-pohjaisissa palveluissa voidaan käyttää WSDL:ää (Web Services Description Language) ilmoittamaan palveluista välittäjille, ja SOAP:a (Simple Object Access Protocol) palvelun ja käyttäjän väliseen kommunikointiin (Li & Zhou, 2010). Muita mahdollisia menetelmiä palvelun ja käyttäjän väliseen kommunikointiin ovat muun muassa REST, RPC, DCOM ja CORBA (Al-Jaroodi ym., 2010).

Luetelluista menetelmistä REST ja SOAP pohjautuvat HTTP-protokollan päälle, eli kommunikointi sovellusten välillä tapahtuu suoraan päästä päähän. Näin kommunikointi palvelun ja käyttäjän välillä voi tapahtua itsenäisesti ilman väliohjelmaa, mutta palvelupyynnöt täytyy tehdä etäkutsujen tapaan synkronisesti.

Huhns & Singh (2005) ovat tutkineet tapausta, jossa WWW-pohjaisen palvelupyynnön on kuljettava niin sanotusti ketjussa usean palvelun kautta, ennen kuin vastaus päätyy takaisin käyttäjälle. Heidän mukaansa transaktioilla voidaan varmistaa tilamuutosten tapahtuminen palveluiden välillä yhtenäisesti, mutta niitä ei voida soveltaa palvelun ja käyttäjän välisissä HTTP-kutsuissa. Ratkaisuiksi ongelmaan on ehdotettu muun muassa tapoja korjata rikkinäinen yhteys tai hoitamalla kyselyt asynkronisesti joko erillisillä palveluilla (esim. sähköpostilla) tai integroimalla kyselyt viestipohjaiseen väliohjelmaan (Huhns & Singh, 2005).

4 AMQP

AMQP (Advanced Message Queuing Protocol) on sovelluskerroksen protokolla, joka soveltaa viestikeskeisen väliohjelman mallia. Protokollan suunnittelussa ovat mukana muutamat finanssi- ja teknologiayritykset. Yritysten päämääränä on kehittää avoin standardi skaalautuvalle asynkroniselle viestintäprotokollalle (Vinoski, 2006, s. 87). Protokollan kehitystä on motivoinut finanssiyritysten tarve saada tehokas viestintäkeino sovellusten välille (Vinoski, 2006, s. 88), joihin WWW-palvelut eivät ole kykeneet vastaamaan (O'Hara, 2007, s. 48).

Uusimman saatavilla olevan protokollan spesifikaation versio on 1.0, mutta tut-

kielmassa tarkasteltavana versiona käytetään versiota 0-9-1. Tässä luvussa perehdytään työkaluihin joilla viestintäratkaisut toteutetaan, ja siihen kuinka työkaluja voidaan soveltaa käytännössä. Luvun lopussa tutustutaan lyhyesti AMQP:n versioon 1.0.

4.1 Työkalut

AMQP:ssa määritetään semantiikat, joita AMQP-toteutusten on noudatettava toimiakseen muiden järjestelmien kanssa (Vinoski, 2006, s. 88). Protokollan semantiikkaa voidaan tarkastella joukkona työkaluja, joita yhdistämällä voidaan rakentaa viestintäratkaisuja. Työkalut koostuvat välittäjästä, viestijonoista, vaihteista, viesteistä ja siteistä.

Muiden viestikeskeisten väliohjelmien tapaan myös AMQP sisältää oman toteutuksensa viestijonoista (message queue) ja viesteistä (message). Viestijonot säilyttävät viestejä väliohjelman muistissa tai kiintolevyllä siihen asti kunnes jokin ohjelma lukee viestit (Vinoski, 2006, s. 88). Jokaista viestijonoa kohden on uniikki nimi, jonka perusteella ohjelmat pystyvät lukemaan oikeaa jonoa. Asiakasohjelmat voivat itse luoda omia viestijonojaan tai lukea olemassa olevia jonoja. Viestit sisältävät otsikkotietueita viestin reitittämistä varten sekä vapaamuotoista dataa (O'Hara, 2007, s. 52).

Viestit välitetään AMQP:ssa vaihteiden (exchange) avulla. Jokainen vaihteen vastaanottama viesti välitetään asianmukaisille viestijonoille määrättyjen sääntöjen perusteella, jonka jälkeen viesti poistuu kokonaan vaihteesta (Vinoski, 2006, s. 88). Asiakasohjelmat eivät siis suoraan toimita viestejään toisillensa vaan vaihteille.

Jokaista vaihdetta varten on algoritmi, jonka perusteella viestit jaetaan viestijonoille. Algoritmityypit voidaan karkeasti jakaa suoravaihtoon (direct exhange) ja aihepohjaiseen vaihtoon (topic exhange). Suoravaihdossa viestit välitetään jonoille avainsanojen perusteella, ja aihepohjaisessa kuvioiden perusteella. Välittäminen tapahtuu vertaamalla viestin sisältämää avainta avainsanaan tai kuvioon. (AMQP, 2008)

Kuviot voidaan ilmaista säännöllisten lauseiden tapaan muodossa "*.sana.#", jossa "*" vastaa yhtä sanaa ja "#" vastaa yhtä tai useampaa sanaa. Esimerkiksi kuvion "sana.*" kanssa täsmäävat avaimet "sana.yksi" ja "sana.kaksi", mutta avaimet "joku.yksi" tai "sana.yksi.kaksi" eivät täsmää.

Jotta viestit voidaan välittää vaihteilta viestijonoille, on oltava jokin tapa linkittää jonot ja vaihteet. Siteillä (binding) määrätään säännöt, joiden perusteella vaihteelle

saapuneet viestit voidaan viedä jonolle (Vinoski, 2006, s. 88). Esimerkiksi jono voidaan sitoa suoravaihdetyyppiseen vaihteeseen avainsanalla "XXX", jolloin kaikki samalla avainsanalla vaihteelle saapuneet viestit päätyvät kyseiselle jonolle. Ainepohjaisissa vaihteissa avainsanojen tilalla käytetään sitoessa kuvioita.

Siteitä voidaan asettaa jonon ja vaihteen välille useita, jolloin viesti voi päätyä jonolle useamman säännön perusteella. Aihepohjaisessa vaihdossa tyypillisesti riittää vain yksi side, koska siinä siteellä voidaan ilmaista useampi vastaanotettava aihe.

Viestejä lukuunottamatta jokainen työkalu on osa AMQP-palvelinsovellusta, jota kutsujaan välittäjäksi (AMQP, 2008). Pääteohjelmien tarvitsee siis ainoastaan kyetä muodostamaan oikeata muotoa olevia viestejä. Palvelinsovellusta kutsutaan välittäjäksi (broker).

4.2 Esimerkkejä

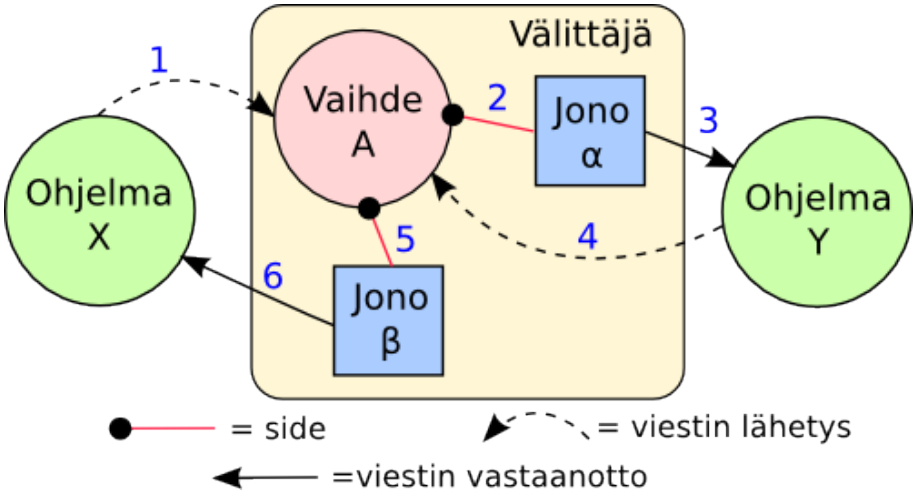
Yksinkertaisimmillaan viestintä AMQP:ssa koostuu kahden ohjelman välisestä viestinnästä. Kuvan 3 esimerkissä esitetään kuinka ohjelma X voi välittää viestin ohjelmalle Y, ja saada takaisin vastauksen. Vaihde A on tässä esimerkissä suoravaihtotyyppinen.

Viestin kulkeminen on kuvattu vaiheilla 1–3, ja paluuviestin kulku vaiheilla 4–6. Jos oletetaan, että vaihteen A ja jonon α välinen side on kytketty avainsanalla "YYY", ja vaihteen A ja jonon β välinen side on kytketty avainsanalla "XXX", niin tällöin vaiheet ovat seuraavat:

1. Ohjelma X lähettää vaihteeseen A viestin avainsanalla "YYY".
2. Vaihde A välittää siteen ja avainsanan perusteella viestin jonolle α.
3. Ohjelma Y lukee saapuneen viestin jonosta α.
4. Ohjelma Y lähettää viestin vaiheeseen A avainsanalla "XXX".
5. Vaihde A välittää siteen ja avainsanan perusteella viestin jonolle β.
6. Ohjelma X lukee saapuneen viestin jonosta β.

Kuvan 3 viestijonot voivat olla vaihtoehtoisesti sidoksissa eri vaihteisiin, jolloin myös ohjelmat lähettävät viestinsä eri vaihteisiin.

Lähettäjän tarvitsee siis tietää vaihteen nimi, vastaanottajan avain ja jonon nimi, josta vastaus luetaan. Samoin myös vastaanottajan on tiedettävä lähettäjän avain ja



Kuva 3: Kahden ohjelman välinen viestintä suoravaihdetyyppisen vaihteen välityksellä.

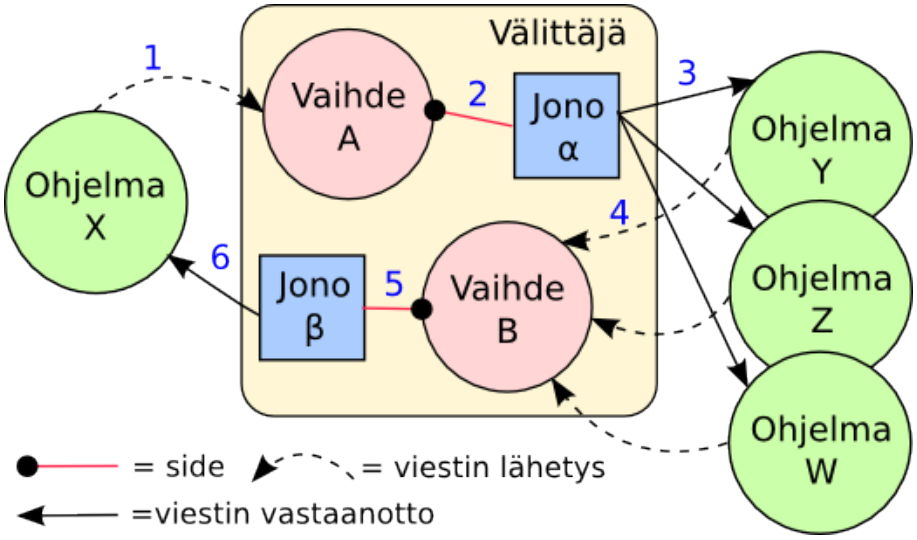
vaihde, jotta se voi vastata viestiin. Jos lähettäjiä on useampia, ja jokaisella lähettäjällä on oma vastausjononsa, niin kuinka vastaanottaja tietää mihin vastausviesti lähetetään? Eräs tapa erottaa viestit toisistaan on lähettää viestin otsikkotietojen mukana tieto siitä mihin vastaus tulisi lähettää. Tyypillisesti tämä tapahtuu luomalla vastauksia varten väliaikainen viestijono, ja asettamalla viestijonon reititystiedot “reply-to” -otsikkotietueeseen (AMQP, 2008).

Koska lähettäjän tarvitsee tietää ainoastaan vaihde ja avain, voi vaihde–avain -pariin olla liitettynä erilaisia asetelmia jonoja ja ohjelmia, ja asetelma säilyy lähettäjän näkökulmasta aina samanlaisena. Tätä on hyödynnetty kuvan 4 esimerkissä, jossa tilanne säilyy ohjelman X kannalta samana. Esimerkissä viestin lukee ohjelmista joko Y, Z tai W. Tässä esimerkissä paluuviestit reititetään tällä kertaa vaihteen B kautta.

Kuvan 4 esimerkkiä voidaan hyödyntää monikkoavaruuden tapaan (ks. luku 2.2) muun muassa työnjakamisessa. Tällöin ohjelma Z toimii työnjakajana, ja ohjelmat Y, Z ja W työohjelmina.

Jos esimerkeissä vaihteeseen A olisi liitetty useampi viestijono samalla avaimella, viesti kopioituisi jokaiselle jonolle (AMQP, 2008). Tällä tavalla voidaan toteuttaa julkaisu–tilaus-tyylisiä (ks. luku 2.3) ratkaisuja.

Samoin myös yhdellä jonolla voi olla kerrallaan useampi side liitettynä vaihteisiin (AMQP, 2008), jolloin viestejä voidaan kerätä useammasta lähteestä tai eri avainsanoilla. Tätä voidaan yhä ehostaa vaihtamalla vaihteen tyyppi aihepohjaiseksi, jolloin voidaan vastaanottaa viestejä kuvioiden perusteella.



Kuva 4: Viestien jakaminen ohjelmalta useammalle ohjelmalle.

4.3 AMQP 1.0

AMQP versio 1.0 on AMQP:n ensimmäinen standardi. Versiossa 1.0 työkalut koostuvat viestien lisäksi kahdenlaisista osista, solmuista ja linkeistä. Solmut pystyvät luomaan, tuhoamaan ja välittämään viestejä. Linkit ovat yhdensuuntaisia solmujen välisiä yhteyksiä, joilla pystytään ilmaisemaan mistä viestit tulevat ja minne ne lähtevät. Linkkeihin voidaan luoda sääntöjä viestien kulkemiselle linkin kautta seuraaville solmuille. Viestit siirtyvät linkissä vasta silloin kun linkille määrätty sääntö toteutuu. (OASIS, 2012)

Voidaan siis ajatella, että solmut korvaavat edellisen version jonot, ja linkit korvaavat siteet. Edellisen version vaihteet voidaan korvata yhdistämällä solmuja ja linkkejä. Viestit välitetään solmuille, mutta solmujen sijaan linkit hoitavat oikeiden viestien välittämisen eteenpäin.

Solmut ja linkit kykenevät myös edellisiä työkaluja monipuolisempiin ratkaisuihin. Edellisessä versiossa viestit kulkevat aina lähettäjältä vaihteelle, vaihteelta viestijonolle ja viestijonolta vastaanottajalle. Solmuja ja linkkejä voidaan sen sijaan yhdistää peräkkäin useita. Voidaan muodostaa esimerkiksi ratkaisuja, joissa viestit kulkevatkin useamman niin sanotun vaihteen läpi.

5 AMQP mobiileissa hajautetuissa järjestelmissä

Hajautetulla järjestelmällä tarkoitetaan eri tietokoneilla sijaitsevien laskennallisten komponenttien joukkoa, jotka ovat toimivat keskenään tietoverkon välityksellä (Capra ym., 2001, s. 3). Mobiililla hajautetulla järjestelmällä tarkoitetaan hajautettua järjestelmää, jonka laitteet koostuvat ainakin osittain mobiileista laitteista (Li ym., 2008). Se ei siis välttämättä täysin koostu mobiileista laitteista, mutta siinä on erityisesti otettava huomioon mobiilien laitteiden erityispiirteet.

Tässä luvussa tarkastellaan ensimmäisenä mobiilien hajautettujen järjestelmien ominaisuuksia. Tämän jälkeen luvussa tarkastellaan mobiilien hajautettujen järjestelmien haasteita, jotka väliohjelmien tulisi ottaa huomioon, ja kuinka AMQP soveltuu näihin haasteisiin. Näitä haasteita ovat väliohjelmien keveys ja asynkronisuus, irralliset liitokset, palvelujen löydettävyys ja ohjelmien kontekstien käsittely.

5.1 Mobiilien hajautettujen järjestelmien ominaisuuksia

Hajautettujen järjestelmien eroja voidaan tutkia niiden ominaisuuksien avulla. Ominaisuudet voidaan jakaa laitetyyppiin, verkkoyhteyden tyyppiin ja suorituskontekstiin (Capra ym., 2001, s. 4).

Perinteisten hajautettujen järjestelmien laitteet ja niiden verkkoyhteydet ovat tyypillisesti kiinteitä, sekä niiden kontekstit säilyvät yleensä staattisena (Capra ym., 2001, s. 4–5). Esimerkiksi työasemakone on kyseistä tyyppiä edustava laite. Työasemia on tarkoitettu käytettäväksi pitkiä aikoja yhdessä paikkaa, niiden verkkoyhteydet ovat tyypillisesti luotettavia, ja konteksti säilyy suurinpiirtein samana.

Mobiilien hajautettujen järjestelmien laitteita on tarkoitus pystyä siirtämään jopa käytön aikana. Lisäksi mobiilien laitteiden verkkoyhteydet ovat yleensä katkonaisia, mikä tekee laitteen suorituskontekstista dynaamisen (Capra ym., 2001, s. 5). Matkapuhelin on tyypillinen esimerkki mobiilista laitteesta. Matkapuhelin saattaa joutua tilapäisesti katvealueelle langattoman lähiverkon ulottumattomiin (yhteys katkonainen), ja samalla saattaa pyrkiä jatkamaan tiedonsiirtoa esimerkiksi 3G-verkon välityksellä (konteksti muuttuu).

Katkonaisten yhteyksien ja laitteiden siirtymisen voidaan myös ajatella kannustavan ohjelmien riippumattomuuteen. Jos ohjelmat ovat riippuvaisia toisistaan, ohjelmien täytyy olla suorituksen ajan yhteydessä väliohjelmaan, jotta ohjelmat toimisivat normaalisti. Yhteyksien katkonaisuuden ja laitteiden siirtymisen vuoksi se on kuitenkin hankala saavuttaa.

5.2 Väliohjelman keveys

Mobiilisovelluksia käytetään tyypillisesti laitteissa, joissa ei ole paljoa suorituskykyä tai muita resursseja. Siispä raskaiden väliohjelmien ajaminen näissä laitteissa ei ole kannattavaa (Li ym., 2008, s. 335). Väliohjelmien aiheuttaman laskennallisen kuorman pitäisi siis olla mahdollisimman pieni.

Jos hajautettu järjestelmä koostuu pelkästään mobiileista laitteista, väliohjelmien pitäisi olla tarpeeksi kevyitä. Toisaalta järjestelmään voidaan sijoittaa myös tehokkaampia laitteita, joissa voidaan suorittaa väliohjelmien raskaampia osia. Esimerkiksi väliohjelma, joka säilyttää dataa muiden ohjelmien puolesta, voidaan suorittaa laitteessa, johon tietoliikenneyhteydet ovat tavallista nopeampia.

AMQP:n raskainta osaa eli välittäjää on tarkoitus ajaa palvelimella. Koska työkalut viestejä lukuunottamatta ovat osa välittäjää, väliohjelman logiikkaa suoritetaan lähes täysin palvelimella. Pääteohjelman tarvitsee siis ainoastaan pystyä kommunikoimaan protokollan mukaan toimiakseen välittäjän kanssa. Toisin sanoen päätelaite ei tarvitse erillistä ohjelmaa.

Koska viestien sisältämä data on vapaamuotoista ja ohjelmat voivat valita ajankohdan viestien lukemiselle, AMQP ei varsinaisesti aseta vaatimuksia myöskään tietoliikenneyhteyksien nopeudelle. Kun luodaan viestintäratkaisua yhdistelemällä AMQP:n työkaluja, on erikseen otettava huomioon, kuinka suuren rasitteen ratkaisu tuo päätelaitteiden tietoliikenneyhteyksille.

5.3 Asynkronisuus ja irralliset liitokset

Mobiilien laitteiden yhteydet ovat katkonaisia, joten kommunikaatiotavan tulisi olla asynkroninen (Li ym., 2008, s. 335). Mobiilin laitteen ei voida aina olettaa pystyvän olemaan yhteydessä toiseen ohjelmaan koko toiminnon ajan. Väliohjelmat eivät siis voi noudattaa esimerkiksi etäkutsumallia tai hajautettua oliomallia, joissa kommunikaatio tapahtuu synkronisesti.

Katkonaiset yhteydet kannustavat myös toimintoihin, jotka ovat irrallisesti liitettyjä (loose coupling). Tällöin ohjelmat pystyvät jatkamaan kommunikointia keskenään vaikka yhteydet välillä katkeaisivatkin (Li ym., 2008, s. 336). Asynkroniseen kommunikointitapaan liittyy jonkinlainen tapa kerätä tuloksia talteen myöhempää lukemista varten. Esimerkiksi ohjelma voi lukea tulokset monikkoavaruudesta tai viestijonosta. Näiden keinojen avulla ohjelma voi katkaista yhteytensä lähetettyään suoritettavat toiminnot kohteelle (Li ym., 2008). Voidaan siis ajatella, että irralli-

sissa liitoksissa pääteohjelman tarvitsee olla yhteydessä väliohjelmaan ainoastaan lähetys- ja vastaanottovaiheissa.

Irrallisia liitoksia voidaan toteuttaa myös varastoimalla osa palveluilta saadusta datasta talteen laitteen muistiin. Tällöin yhteyden katketessa voidaan pyrkiä turvautumaan varastoituun dataan (Li ym., 2008, s. 336). Yhteyden palautuessa pääteohjelma voi tarvittaessa synkronoida varastoituun dataan tehdyt muutokset palvelimelle.

Kommunikaatio AMQP:ssa tapahtuu asynkronisesti. Viestin saavuttua vaihteelle, viesti reititetään oikealle viestijonolle, josta toinen ohjelma voi lukea viestin valitsemallaan ajankohdalla. Viesti siis säilyy käsiteltävänä vaikka lähettäjä sulkisi yhteyden lähetyksen jälkeen. Myös liitoksien irrallisuus onnistuu AMQP:lla. Pääteohjelma voi sulkea yhteyden väliohjelmaan viestin lähetettyään viestin, ja kerätä vastauksen viestijonosta myöhemmällä yhteydellä.

5.4 Kontekstit mobiileissa ympäristöissä

Kontekstitietoisuutta voidaan hyödyntää sekä perinteisissä että mobiileissa hajautetuissa järjestelmissä. Mobiilien pääteohjelmien jatkuvasti muuttuvasta ympäristöstä johtuen voidaan kuitenkin ajatella, että suurin hyöty kontekstitietoisuudesta saadaan mobiileissa hajautetuissa järjestelmissä.

Koska kontekstit ovat mobiileissa laitteissa dynaamisia, ja laitteiden sijainti verkossa vaihtelee jatkuvasti, ei ole kannattavaa ottaa huomioon jokaista mahdollista kontekstia väliohjelmaa toteuttaessa. Siispä on järkevämpää välittää kontekstidata pääteohjelmille kuin käsitellä sitä väliohjelman sisällä (Capra ym., 2001; Li ym., 2008).

AMQP:ssa ei suoraan ole omaa tukea konteksteille, mutta sen ominaisuuksia voidaan hyödyntää kontekstitietoisuuden saavuttamiseksi. Kontekstidataa voidaan sijoittaa viestien otsikkotietueisiin, jolloin vastaanottaja voi tehdä päätöksiä kontekstidatan perusteella lukematta koko viestiä.

Kun vaihteena käytetään aihepohjaista vaihdetta, voidaan sijoittaa rajallinen määrä kontekstidataa viestin avaimeen. Tällöin konteksteihin voidaan reagoida jo reititysvaiheessa. Esimerkiksi mobiilissa ympäristössä ohjelma voi lisätä avaimeen “aihe1” päätteen “.mobi”, jolloin kaikki samasta aiheesta ja samasta kontekstista kiinnostuneet ohjelmat voivat ilmaista sidoksen kuviolla “aihe1.mobi”. Tällainen kontekstien ilmaisu on kuitenkin hyvin rajallinen, koska jokainen mahdollinen ilmaistava konteksti tulee ottaa huomioon kuviossa.

5.5 Palvelujen löydettävyys

Koska palvelujen saavutettavuus vaihtuu helposti, tarvitaan keinoja korvaavien palvelujen löytämiseen. Lisäksi on otettava huomioon esimerkiksi kuinka palveluita voidaan julkaista, miten toimitaan palveluiden mitätöityessä, ja kuinka lähellä olevia palveluita etsitään (Li ym., 2008, s. 337).

Tietoa palveluista voidaan säilyttää joko keskitetyssä globaalissa palvelussa tai hajautetussa palvelussa. Hajautetussa palvelussa jokainen laite tarkkailee lähellä olevia laitteita (Li ym., 2008, s. 337).

Keskitetty palvelu voi toimia esimerkiksi palvelukeskeisen arkkitehtuurin avulla. Tällöin jokainen laite voi toimia sekä palveluntarjoajana että palvelunkäyttäjänä. Kun laite liikkuu verkkoympäristössä, laite voi ylläpitää palvelun tietoja ajan tasalla päivittämällä tiedot rekisteriin.

AMQP ei sisällä tukea palvelujen löytämiselle. Tällainen ominaisuus voidaan kuitenkin rakentaa esimerkiksi erillisellä ohjelmalla. Tällöin väliohjelmaan kytketään erillinen palvelinohjelma, joka pitää yllä rekisteriä palveluista. Muut ohjelmat voivat hakea palveluita kommunikoimalla ohjelman kanssa AMQP:n välityksellä. Tällöin myös jokainen palvelu on erikseen rekisteröitävä ohjelman ylläpitämään rekisteriin. Ohjelma toimii siis kuten palvelukeskeisen arkkitehtuurin palvelunvälittäjä.

Palvelujen etsiminen ei välttämättä ole AMQP:ssa paras ratkaisu palvelujen saavuttamiseen. Koska viestin lähettäjä on aina yhteydessä vaihteeseen eikä vastaanottajaan, vastaanottajien määrä ja asetelma voi vaihdella vapaasti. Siispä yhteys palveluun säilyy aina samana samalla, kun palvelun rakenne muuttuu. Jos siis palvelujen rajapinnat, eli vaihteet, on ennalta sovittu, ohjelmien ei tarvitse missään vaiheessa hakea korvaavaa palvelua.

6 Yhteenveto

Tutkielman tavoitteena oli tutkia AMQP:n soveltuvuutta mobiileissa hajautetuissa järjestelmissä niiden ominaisuuksien perusteella. Lisäksi tutkielmassa tutustuttiin väliohjelmien ominaisuuksiin ja erilaisiin väliohjelmamalleihin.

Tutkielmassa tarkasteltiin väliohjelmien ominaisuuksista laskennallista kuormaa, kommunikointitapoja, konteksteja ja transaktioita. Lähdetietojen perusteella selvisi, että mobiileissa hajautetuissa järjestelmissä väliohjelmien laskennallisen kuorman

on oltava kevyt, kommunikointitavan oltava asynkroninen ja pääteohjelmien tulisi itse kyetä tekemään päätöksiä kontekstien perusteella. Lisäksi väliohjelmien tulisi mobiileissa ympäristöissä jollakin tapaa tukea irrallisia liitoksia ja tarjota toimintoja palvelujen löytämiseen.

Tutkielmassa arvioitiin AMQP:n soveltuvuutta mobiileissa hajautetuissa järjestelmissä tarkastelemalla kuinka kevyt AMQP:n laskennallinen kuorma on, kuinka hyvin se noudattaa asynkronista viestintätapaa ja irrallisia liitoksia, ja kuinka hyvin AMQP tukee konteksteja ja palvelujen löydettävyyttä. Tutkielmassa selvisi, että AMQP:n laskennallinen kuorma on minimaalinen. Se on myös täysin asynkroninen ja tukee irrallisia liitoksia. AMQP tukee kontekstidatan välittämistä muun datan mukana, mutta tarjoaa minimaalisen tuen kontekstidataan reagoimiseen reititysvaiheessa. Vaikka AMQP ei sisällä omaa tukea palvelujen löytämiselle, se kuitenkin tarjoaa mahdollisuuden kiinteään rajapintaan.

Tutkielmassa tarkasteltavana oli AMQP:n versio 0-9-1, joka on monella tapaa erilainen AMQP:n versioon 1.0 nähden. Jatkossa voidaan siis tutkia voidaanko versiossa 1.0 toteuttaa ratkaisuja, joissa kontekstit ja palvelujen löydettävyys on huomioitu paremmin.

Eräs tapa huolehtia palvelujen löydettävyydestä on noudattaa palvelukeskeistä arkkitehtuuria. Jatkossa voidaan tutkia voidaanko AMQP:aa käyttää suoraan palvelukeskeisenä väliohjelmana tai voidaanko toteuttaa palvelukeskeinen väliohjelma AMQP:n rinnalle.

Viitteet

Al-Jaroodi, J., Mohamed, N. & Aziz, J. Service Oriented Middleware: Trends and Challenges. Teoksessa *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, sivut 974 –979. 2010. doi:10.1109/ITNG.2010.55.

AMQP. Advanced Message Queuing Protocol Specification v0-9-1. 2008.

Bellavista, P., Corradi, A., Montanari, R. & Stefanelli, C. Context-aware middleware for resource management in the wireless Internet. *Software Engineering, IEEE Transactions on*, 29(12):1086 – 1099, 2003. doi:10.1109/TSE.2003.1265523.

Capra, L., Emmerich, W. & Mascolo, C. Middleware for Mobile Computing. *UCL Research Note RN/30/01*, 2001.

Geihs, K. Middleware challenges ahead. *Computer*, 34(6):24 –31, 2001. doi:10.1109/2.928618.

Huhns, M.N. & Singh, M.P. Service-oriented computing: key concepts and principles. *Internet Computing, IEEE*, 9(1):75 – 81, 2005. doi:10.1109/MIC.2005.21.

Issarny, V., Caporuscio, M. & Georgantas, N. A Perspective on the Future of Middleware-based Software Engineering. Teoksessa *2007 Future of Software Engineering*, FOSE '07, sivut 244–258. IEEE Computer Society, Washington, DC, USA, 2007. ISBN 0-7695-2829-5. doi:10.1109/FOSE.2007.2.

Li, Q., Zhen, W., Wang, M., Zhou, M. & He, J. Researches on Key Issues of Mobile Middleware Technology. Teoksessa *Embedded Software and Systems Symposia, 2008. ICESS Symposia '08. International Conference on*, sivut 333 –338. 2008. doi:10.1109/ICESS.Symposia.2008.51.

Li, Q. & Zhou, M. The State of the Art in Middleware. Teoksessa *Information Technology and Applications (IFITA), 2010 International Forum on*, osa 1, sivut 83 –85. 2010. doi:10.1109/IFITA.2010.118.

OASIS. OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. 2012.

O'Hara, J. Toward a Commodity Enterprise Middleware. *Queue*, 5(4):48–55, 2007. doi:10.1145/1255421.1255424.

Vinoski, S. Advanced Message Queuing Protocol. *Internet Computing, IEEE*, 10(6):87 –89, 2006. doi:10.1109/MIC.2006.116.