Kari Patana

Avoin viestijonoprotokolla Korpin verkkomaksuissa

Tietotekniikan kandidaatintutkielma

8. helmikuuta 2014

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Kari Patana

Yhteystiedot: kari.e.m.patana@jyu.fi

Työn nimi: Avoin viestijonoprotokolla Korpin verkkomaksuissa

Title in English: Open Message Queue Protocol in Korppi Web Payments

Työ: Kandidaatintutkielma

Sivumäärä: 21+0

Tiivistelmä: Viestijonoilla voidaan ratkaista tilanteita, joissa tiedon tuottajia ja tiedon kuluttajia on useita ja niiden välinen tiedonsiirto pitää saada tehtyä järkevästi. Viestijonoon kuuluu tuottajia, jotka lähettävät viestejä, sekä kuluttajia, jotka tilaavat tietyt ehdot täyttävät viestit itselleen. Viestijonojen avulla voidaan jakaa järjestelmää pienempiin, riippumattomiin osiin tai jakaa kuormaa usean eri käsittelijän kesken. Advanced Message Queuing Protocol (AMQP) on avoin protokolla viestijonojärjestelmän tekemiseen ja RabbitMQ eräs sen 0-9-1 version toteutus. Jyväskylän yliopiston Korppi-järjestelmän ja Payments-verkkomaksujen yhteys on toteutettu RabbitMQ:lla.

Avainsanat: AMQP, RabbitMQ, viestijono, Korppi

Abstract: Using message queues it is possible to handle cases where there are many suppliers and consumers of information and there's a need to arrange the data transfer between them in a smart way. A message queue consists of publishers that hand out information and consumers that subscibe to information that matches certain criteria. Message queues help divide systems into smaller, individual components, and to divide labour to many handlers. Advanced Message Queuing Protocol (AMQP) is an open protocol for implementing message queues and RabbitMQ an implementation of the version 0-9-1. The connection between the Korppi system of the University of Jyväskylä and the Payments web shop has been implemented using RabbitMQ.

Keywords: AMQP, RabbitMQ, message queue, Korppi

Kuviot

Sisältö

1 Johdanto

On paljon tilanteita, joissa monista eri lähteistä on tarve saada tietoa koostettua. Esimerkki tällaisesta on pankkimaailma, jossa välittäjä tarvitsee tietoja osake- ja valuttakurssien muutoksista. Perinteinen tapa tietojen keräämiseen on tehdä kyselyjä aina silloin kun tietoa tarvitaan. Tällaisessa on kuitenkin ongelmia: Jos samalla tiedolla on monta tarvitsijaa, jokaisen pitää kysellä ja vastaanottaa tieto erikseen. Toisaalta jos tieto on tarve saada heti, kun se on tarjolla, on pakko jatkuvasti tehdä uusia kyselyitä.

Vaihtoehto tälle on tuottaja-kuluttaja-malli, jossa tuottaja laittaa tiedon tarjolle heti kun se on olemassa ja kuluttaja saa sen saman tien käyttöönsä. Tällainen voidaan toteuttaa viestiväylän avulla. Advanced Message Queuing Protocol (AMQP) on avoin standardi viestiväylän toteuttamiseen ja RabbitMQ yksi sen toteutus. Tutkielmassa esitellään kirjallisuuskatsauksena AMQP:n ja RabbitMQ:n kehitystä, ominaisuuksia ja käyttöä sekä esitellään käytännön kokemuksia RabbitMQ:n hyödyntämisestä Jyväskylän yliopiston Korppi-järjestelmän ja verkkokaupan integroimiseen.

Tutkielman luvussa 2 kuvataan viestijonojen periaatteet. Luvussa 3 esitellään AMQP-protokollaa ja sen kehitystä. Luvussa 4 tarkastellaan AMQP:n RabbitMQ-toteutusta. Luvussa 5 selvitetään RabbitMQ:n käyttöä Korpin ja verkkokaupan toiminnassa.

2 Viestijonot

Viestijonojen ideaa hyödynnettiin ensimmäisenä pankkialalla, kun Goldman Sachs alkoi käyttää The Information Bus (viestiväylä, TIB) -järjestelmää. Pörssivälittäjä tarvitsi työssään ajantasaisia tietoja monista eri lähteistä ja eri aiheista. Näitä varten oli useita erillisiä päätteitä, joista erilaisiin tietoihin oli pääsy. TIB:n avulla useat päätteet voitiin korvata yhdellä viestiväylään kytketyllä laitteella. (Videla, 2012)

TIB:n kehittäjä oli Vivek Ranadivén Tekneron-nimisessä yrityksessä. Hän sai idean tietokoneen emolevyn tapaisesta väylästä, johon laitteiden sijasta liittyvät ohjelmat. Järjestelmän toimijoina ovat tuottajat, kuluttajat sekä välitysjärjestelmä (broker). Tuottajat lähettävät viestejä välitysjärjestelmään. Kuluttajat puolestaan voivat liittyä tilaajiksi tietynlaisille viesteille. Viestijärjestelmä päättelee, mille kuluttajille viesti pitää välittää. (Videla, 2012)

Vastaavia järjestelmiä ottivat sittemmin käyttöön muutkin pankit. Koska kyseessä olivat lähinnä pankkien sisäiset järjestelmät, eivät eri toimijat sopineet yhteisistä standardeista. Eri pankeilla oli omia, kalliita ja suljettuja järjestelmiään. Niiden kehitystä hidasti myös se, ettei ohjelmistokehitys varsinaisesti ole pankkien toimialaa. Sen vuoksi aikaa myöten myös ohjelmistotalot ovat vastanneet tarpeeseen omilla järjestelmillään. Hallitseva, kaupallinen viestijonoalusta on nykyisin WebSphere HQ, jonka kehitys alkoi vuonna 1993 nimellä IBM MQSeries. Kaupalliset järjestelmät ovat perinteisesti suljettuja vailla yhteisiä standardeja. Tällöin on hyvin vaikeaa vaihtaa olemassa olevan tilalle toista viestijonojärjestelmää. Tämän myötä on ilmennyt tarve yleiselle standardille, joka mahdollistaisi tuottajan ja tilaajan vaihtamisen, kunhan viestien muoto ja välitysohjelmiston toiminta säilyvät ennallaan. (O'Hara, 2007)

Yksi yritys tuoda yhteinen rajapinta eri järjestelmien välille on Java Message Service (JMS). Se tarjoaa saman Java-sovellusrajapinnan useiden eri toimittajien viestijonototeutukisen käyttämiseen. Ajuri hoitaa viestin muuttamisen oikeaan muotoon sekä välittää viestin kohdejärjestelmään. Koska eri toteutusten rajapinnat saattavat

olla hyvinkin erilaisia, on niiden yhteensovittamisessa kuitenkin pahoja ongelmia. Sen vuoksi JMS ei kunnolla vastaa yhteisen rajapintastandardin tarpeeseen. (Videla, 2012)

3 AMQP

Luvussa luodaan katsaus AMQP:n syntyyn ja kehittymiseen johtaneisiin seikkoihin. Lisäksi esitellään tärkeimmät protokollaan liittyvät käsitteet sekä kuvataan protokollan toimintaa ja käyttöä.

3.1 Kehitys

Rajapintastandardin tarpeeseen ryhdyttiin vastaamaan JPMorgan-investointipankin sekä iMatrix-kehitysyrityksen projektissa, joka sai nimen Advanced Message Queuing Protocol. Jotta projekti voisi onnistua, O'Hara (2007) mainitsee seuraavat lähtökohdat: täysin määritelty, avoin, ilmainen ja patenteista sekä toteutuksesta riippumaton standardimäärittely. Lisäksi tarvittaisiin oikea, käytössä koeteltu toteutus tai mieluusti useitakin. Standardin pitää myös olla yhteistyön tulos sen sijaan että se olisi vain yhden tahon tarpeiden myötä syntynyt. Kun peruslähtökohdat oli saatu huomioitua, mukaan otettiin vapaasti muita tahoja. Tämä johti AMQP Working Group -työryhmän perustamiseen. (O'Hara, 2007)

AMQP:n kahitys alkoi vuonna 2004 ja ensimmäinen määritys tuli vuonna 2006. Protokollan kehityksessä merkittävä rajakohta oli versio 0-9-1 vuonna 2008. Siihen versioon saakka AMQP:hen kuului sekä liikennöintimalli, joka yritti standardisoida asiakkaiden sekä välitysjärjestelmän välisen liikenteen, että jonomalli, joka kuvasi viestijärjestelmän ominaisuudet siten, että eri asiakasohjelmien välinen toiminta voitiin varmistaa myös eri viestijärjestelmien kesken. Version 0-9-1 toteuttavat esimerkiksi RabbitMQ- ja OpenAMQ -viestijärjestelmät. (O'Hara, 2007) (Pivotal, 2013) (OpenAMQ, 2010)

Version 0-9-1 jälkeen protokollan kehitys siirtyi OASIS-yhteenliittymälle (Organization for the Advancement of Structured Information Standards). 31.10.2012 AMQP 1.0 hyväksyttiin OASIS-standardiksi. Versiossa ei kuitenkaan enää ole AMQP-jonomallia. Verkkosivun Pivotal (2013) mukaan AMQP 1.0 on täysin erilainen protokolla ja RabbitMQ säilyttää yhteensopivuuden versioon 0-9-1. Tutkielmassa keski-

tytään RabbitMQ:n käyttämään versioon 0-9-1. (OASIS, 2012)

3.2 Toimintaperiaate

AMQP:n liikennöintimalli on binääriprotokolla, eli se on tarkoitettu koneelle helpoksi käsitellä, ei niinkään ihmisen luettavaksi. Se myös helpottaa AMQP:n ottamista mukaan verkkolaitteiden erityisille integroiduille piireille. Lisäksi se tekee protokollasta nopean. Varsinainen siirtotie voidaan toteuttaa käyttäen Transmission Control Protocol/Internet Protocol (TCP/IP) - tai Stream Control Transmission Protocol (SCTP) -protokollaa ja tulevaisuudessa mahdollisesti myös User Datagram Protocol (UDP) -protokollaa ja InfiniBandia. (O'Hara, 2007)

Asiakasohjelmat eivät ota suoraa TCP-yhteyttä viestijärjestelmään. Sen sijaan ne käyttävät erityisiä AMQP-kanavia. Sama TCP-yhteys voi toimia yhteytenä mielivaltaiselle määrälle kanavia. Tämän vuoksi jokaista, erillistä yhteyttä varten ei tarvitse avata uutta TCP-yhteyttä, mikä nopeuttaa yhteyksien saantia ja säästää asiakaskoneen resursseja. (Videla, 2012)

AMQP:n jonomallin tärkeimmät käsitteet ovat vaihde (exchange), jono (queue) ja liitos (binding). Vaihde on paikka, johon tuottaja julkaisee viestin. Jono taas on paikka, jonne viesti välitetään kuluttajan saataville. Liitos kertoo, mihin jonoon tai jonoihin viesti vaihteesta välitetään. (O'Hara, 2007)

Viesti itsessään koostuu nimekkeestä (label) ja sisällöstä (payload). Nimekkeen perusteella viesti osataan ohjata oikeisiin jonoihin. Kuluttajalle asti välitetään pelkkä sisältö. Kuluttaja ei siis saa tietoonsa sitä, mistä viesti tuli, ellei sitten viestiosassa sitä kerrota. (Videla, 2012)

AMQP määrittelee komentoja, joita toteutuksissa pitää olla. Se ei kuitenkaan määrittele sovellusrajapintaa. Niinpä eri toteutuksissa esimerkiksi ohjelmoijalle näkyvät metodien nimet ja parametrit voivat olla poikkeavia, vaikka eri totetukset tarjoavatkin samat ominaisuudet. (O'Hara, 2007)

Aiemmista viestijonojärjestelmistä poiketen AMQP:ssa resursseja voidaan määrit-

tää protokollalla itsellään. Jonoja voivat luoda niin kuluttajat kuin tuottajatkin komennolla queue.declare. Jonolle voidaan antaa nimi tai jättää nimi antamatta, jolloin viestijärjestelmä nimeää jonon automaattisesti. Anonyymi jono voi olla hyödyllinen esimerkiksi Remote Procedure Call (RPC) -etäkutsujen toteuttamisessa. Mikäli jonoa yritetään luoda nimellä, joka on jo käytössä, otetaan käyttöön olemassa oleva jono. Monesti on toivottavaa, että sekä tuottaja että kuluttaja yrittävät luoda jonon. Tällä varmistetaan, että jono on olemassa. Mikäli tuottaja julkaisee viestin, jolle sopivaa jonoa ei löydetä, viesti tuhotaan. Julistamalla jonon voi tuottaja varmistaa, ettei viesti katoa. (Videla, 2012)

Vaihteita luodaan exchange_declare-komennolla. On olemassa eri tyyppisiä vaihteita. Yksinkertaisin niistä on suora (direct) vaihde. Tällaiseen vaihteeseen lähetetty viesti välitetään reititysavaimen perusteella vastaavaan jonoon. Viestijärjestelmän on tarjottava suora vaihde sekä myös nimetön oletusvaihde. Jonoa määriteltäessä se liitetään oletusvaihteeseen käyttämällä jonon nimeä reititysavaimena. Tällöin jonoon voidaan lähettää viesti komennolla basic_publish kertomalla pelkästään reititysavaimena toimivan jonon nimen sekä viestin. (Videla, 2012)

Toinen vaihdetyyppi on jakelu (fanout). Tällaiseen vaihteeseen lähetetty viesti välitetään jokaiseen vaihteeseen liittyvään jonoon. Tämä on yksi tapa saada sama viesti välitettyä usealle kuluttajalle. (Videla, 2012)

Kolmantena vaihde voi olla myös aihetyyppinen (topic). Perustapauksessaan se toimii samoin kuin suora vaihde. Tällöin voidaan jonolle ilmaista liitos komennolla queue_bind esimerkiksi siten, että vaihteen ”msg-inbox-logs” jonoon ”logs.exchange” liitetään reititysosoite ”exchange.msg-inbox”. (Videla, 2012)

Aihetyyppiselle vaihteelle voidaan liitoksia kertoa myös jokerimerkkien avulla. Tähdellä (*) voidaan sallia mikä tahansa reititysavain seuraavaan pisteeseen saakka. Risuaidalla (#) voidaan kelpuuttaa mikä tahansa reititysavain. Esimerkiksi voidaan luoda jono, johon ohjataan lokitietoa useista eri järjestelmistä. Se tehtäisiin komennolla queue_bind ilmoittamalla, että vaihteen ”msg-inbox-logs” jono ”logs.exchange” ottaa vastaan reitityavaimia, jotka vastaavat lauseketta ”*.msg-inbox”. Liitoksen te-

keminen vaatii, että jono on jo olemassa. (Videla, 2012)

3.3 Viestin lähettäminen ja vastaanottaminen

Kuluttajalla on kaksi tapaa saada viestejä. Ensimmäinen vaihtoehto on liittyä jonkin jonon tilaajaksi käyttämällä AMQP:n basic_consume-komentoa. Tällöin kuluttaja saa viestejä jatkuvasti sitä mukaa kuin niitä tulee tarjolle. Toinen vaihtoehto on pyytää vain yksi viesti käyttämällä basic_get-komentoa. (Videla, 2012)

Jos viesti saapuu jonoon, jossa ei ole tilaajia, se jää odottamaan. Kun jonoon tulee tilaaja, viesti välitetään tälle. Jos taas tilaajia on monta, viestit välitetään round robin -periaatteen mukaisesti vuoroittain jokaiselle. Yksi viesti päätyy siis vain yhdelle tilaajalle samassa jonossa. (Videla, 2012)

Protokollaan liittyy myös viestin kuittaus basic_ack-komennolla. Viestijärjelmä ei tuhoa viestiä ennen kuin se on kuitattu vastaanotetuksi. Mikäli välitettyyn viestiin ei ole saatu kuittausta ja kuluttaja, jolle viesti välitettiin, menettää yhteyden, palautuu viesti jonoon ja siirtyy seuraavan kuluttajan käsiteltäväksi. Mikäli kuluttaja ei pysty käsittelemään saamaansa viestiä, se voi käyttää basic_reject-komentoa, joka palauttaa viestin muiden saataville. (Videla, 2012)

4 RabbitMQ

Luvussa esitellään AMQP:n toteuttavan RabbitMQ-välitysjärjestelmän syntyä, kehitystä ja toteutustapaa. Lisäksi järjestelmän sovellusintegrointia havainnollistetaan Java-kielellä tehdyin esimerkein. Lopuksi esitellään RabbitMQ:n ominaisuuksia sekä luodaan katsaus sen hyötyihin ja haittoihin.

4.1 Lähtökohta

RabbitMQ on eräs AMQP-protokollan toteutus. Se syntyi CohesiveFT- ja LShift-yritysten yhteisistä tarpeista viestinvälitykseen hajautetussa ympäristössä. Koska valmista järjestelmää ei ollut, yritykset päätyivät kehittämään oman. Rabbit Technologies perustettiin vuonna 2006 samoihin aikoihin kuin AMQP:n ensimmäinen, julkinen versio esiteliin. RabbitMQ:n kehittäjät olivat seuranneet AMQP:n kehitystä ja päätyivät ottamaan protokollan viestijärjestelmän lähtökohdaksi. Nykyisin RabbitMQ toteuttaa AMQP:n version 0-9-1. Verkkosivun Pivotal (2013) mukaan siihen on tarjolla plugin, jolla saadaan yhteensopivuus versioon 1.0, vaikka RabbitMQ:n on muutoin suunniteltu jäävän AMQP:n aiempaan versioon. (Videla, 2012)

RabbitMQ:n toteutuskieleksi on valittu Erlang, joka on puhelinalan yrityksen, Ericssonin, kehittämä ohjelmointikieli. Puhelinvaihteiden ohjelmistoissa on suuret vaatimukset reaaliaikaisuudelle, virheensiedolle ja yhtäaikaisuudelle. Kielen tärkeimmät suunnitteluperiaatteet olivat prosessien eristäminen toisistaan, viestien välittäminen prosessien välillä jaetun muistiavaruuden sijaan sekä mahdollisuus huomata virheet ja niiden aiheuttajat. AMQP-määritys muistuttaa puhelinkytkimen arkkitehtuuria. Tämän vuoksi RabbitMQ päätettiin toteuttaa Erlang-kielellä. (Armstrong, 2010) (Videla, 2012)

Koska RabbitMQ noudattaa AMQP:n versiota 0-9-1, on siinä mukana AMQP-jonomalli, joka sisältää vaihteet, jonot ja liitokset. Tarjolla on kirjastoja eri ohjelmointikielille asiakasohjelmien tekoon. (Videla, 2012)

4.2 Java-esimerkkejä

Java-kirjastoa käyttäen yhteys voidaan luoda ConnectionFactoryn avulla, joka puolestaan voi lukea yhteysasetukset esimerkiksi tietokannasta:

ConnectionFactory factory = new ConnectionFactory();

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

Vaihteen ja jonon luominen, jonon liittäminen sekä viestin julkaiseminen tehdään kanavan avulla:

channel.exchangeDeclare(”logs”, ”fanout”);

channel.queueDeclare(”logs-queue”);

channel.queueBind(”logs-queue”, ”logs”, ””);

channel.basicPublish(”logs-queue”, ””, null, msq.toByteArray());

Metodin queueBind() kolmas parametri määrittäisi sen, millaisia viestejä jonon tulee. Koska vaihde on fanout-tyyppinen, tämä on tarpeeton. Samoin pasicPublish()-metodin toinen parametri kertoisi reititysavaimen, joka on tarpeeton. Kolmanteen parametriin voitaisiin kertoa viestiin liittyviä parameterja. Neljäntenä parametrina on itse viesti tavutaulukkona.

Kuluttaja puolestaan voi kuunnella luotua kanavaa näin:

channel.basicConsume(”logs-queue”, false, consumer);

Toinen parametri kertoo, että viestiä ei automaattisesti kuitata. Kolmantena parametrina on Consumer-luokan ilmentymä, joka varsinaisesti hoitaa viestin käsittelyn. Se voi vaikkapa kirjoittaa viestin tiedostoon ja sitten tehdä kuitata viestin channel.basicAck() -metodilla. Metodi basicConsume() jatkaa kanavan kuuntelua kunnes se keskeytetään.

4.3 Muita ominaisuuksia

AMQP:n version 0-9-1 määrittelyssä (Amqp, 2013) mainitaan mahdollisuus virtuaali-isäntien käyttöön, mutta niiden toiminta ja käyttäminen jätetään toteutuksen vastuulle. RabbitMQ:n mukana tulee sekä komentorivipohjaisia että WWW:ssä toimivia ylläpitotyökaluja. Niiden avulla voidaan luoda virtuaali-isäntiä sekä muutenkin säädellä järjestelmän asetuksia. Virtuaali-isäntä on yksi tieto, joka vaidaan asettaa ConnectionFactory-luokalle. Virtuaali-isännän avulla sama viestijärjestelmä voidaan jakaa useaan kokonaisuuteen, jotka eivät tiedä toisistaan mitään. Sen avulla sama viestijärjestelmä voi palvella useita erillisiä järjestelmiä. Virtuaali-isännän sisällä on omat vaihteet ja jonot. Lisäksi RabbitMQ tarjoaa mahdollisuuden luoda käyttäjiä ja jakaa näille oikeuksia siten, että tietty käyttäjä pääsee vain tiettyyn virtuaali-isäntään. Käyttäjään liittyy myös salasana, mikä on yksi tapa lisätä järjestelmän tietoturvaa. (Videla, 2012)

RabbitMQ tarjoaa myös keinoja vikasietoisuuden lisäämiseen. Normaalisti viestit ovat järjestelmän muistissa ja katoavat, jos järjestelmään tulee virhe. Viestit voidaan turvata asettamalla viestin lähetyksessä välitystavaksi pysyvä (persistent) ja määräämällä sekä vaihde että jono kestäväksi (durable). Tällöin viestit tallennetaan levylle ja järjestelmän käynnistyessä uudelleen voidaan vanha tilanne palauttaa. (Videla, 2012)

On myös mahdollista jakaa RabbitMQ-välitysjärjestelmä toimimaan useassa klusterissa. Näin voidaan tarvittaessa kuormaa jakaa usealle koneelle. Tämä mahdollistaa myös nopean vioista selviämisen, sillä mahdollisesti epäkuntoon menevän klusterin tilalle voidaan nopeasti pystyttää toinen. Klusterointi ei kuitenkaan suojaa viestejä: eri klusterit eivät jaa samoja tietoja, minkä vuoksi klusterin kadotessa myös sen sisältämät viestit saattavat kadota. (Videla, 2012)

4.4 Hyötyjä ja haittoja

Kirjassa Videla (2012) mainitaan yhtenä esimerkkinä RabbitMQ:n hyödyntämisestä järjestelmän paloittelu osiin. Sen sijaan että autentikointimoduuli tekisi autentikoin-

nin suoraan autentikointipalvelimella, voidaan autentikointipyyntö lähettää viestinä RabbitMQ:n avulla. Palvelin kuuntelee jonoa, johon pyynnöt saapuvat, ja vastaa pyyntöön. Jos järjestelmä on rakennettu näin, on helppoa lisätä ominaisuuksia, kuten autentikointipyyntöjen lokitus. Riittää, että lokia kirjoittava palvelu määrittää ja kuuntelee jonoa, joka on liitetty vastaanottamaan autentikointipyynnöistä kertovat viestit. Tällöin voidaan järjestelmää laajentaa muuttamatta olemassa olevia osia lainkaan.

Jyväskylän yliopistossa AMQP:tä hyödynnetään Moniviestin-nimisessä videopalvelussa. Järjestelmässä tarvitaan videoiden käsittelyä. Tämä on RabbitMQ:n viestijonojen avulla hajautettu yhteensä 48:lle eri prosessointiyksikölle. Järjestelmä on nopea ja skaalautuva sekä selviää hyvin verkkokatkoista. (Soukka, 2013)

Tutkimuksessa Smith (2011) RabbitMQ:ta kokeiltiin TeraGrid-järjestelmän korvaajana. Järjestelmä käsittelee PDF-tiedostoja, joiden koko on 200 tavusta kolmeen megatavuun. RabbitMQ:n Java-kirjaston avulla toteutettu yhteysohjelma kykeni välittämään pienimpiä viestejä 25000 kappaletta sekunnissa ja isoimpia kaksikymmentä. Tämä ylitti tarpeen moninkertaisesti.

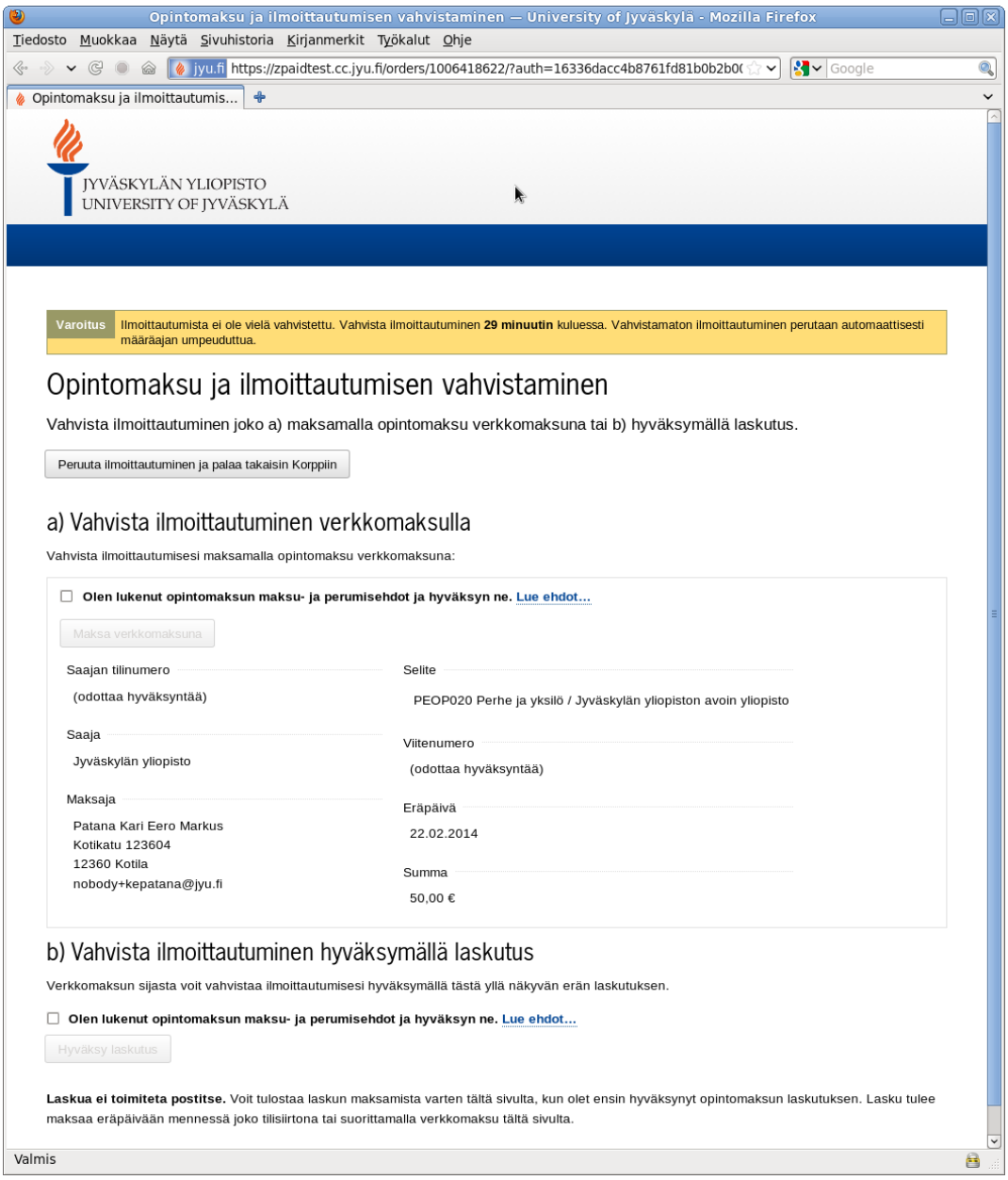
5 RabbitMQ Jyväskylän yliopiston Korppi-järjestelmän ja verkkomaksujen integroinnissa

Jyväskylän yliopiston Korppi-järjestelmän sekä Payments-verkkomaksujärjestelmän integrointi toteutettiin AMQP-protokollaa ja sen RabbitMQ-toteutusta hyödyntäen keväällä 2013 ja otettiin käyttöön alkukesästä. Tiedot järjestelmien välillä siirretään RabbitMQ:n avulla.

Toimintoketju alkaa, kun opiskelija etsii kurssin Korpissa. Hän saa näyttöönsä ilmoittautumispainikkeen sekä mahdollisia ilmoittautumiseen liittyviä valintoja kuvan 1 mukaisesti.

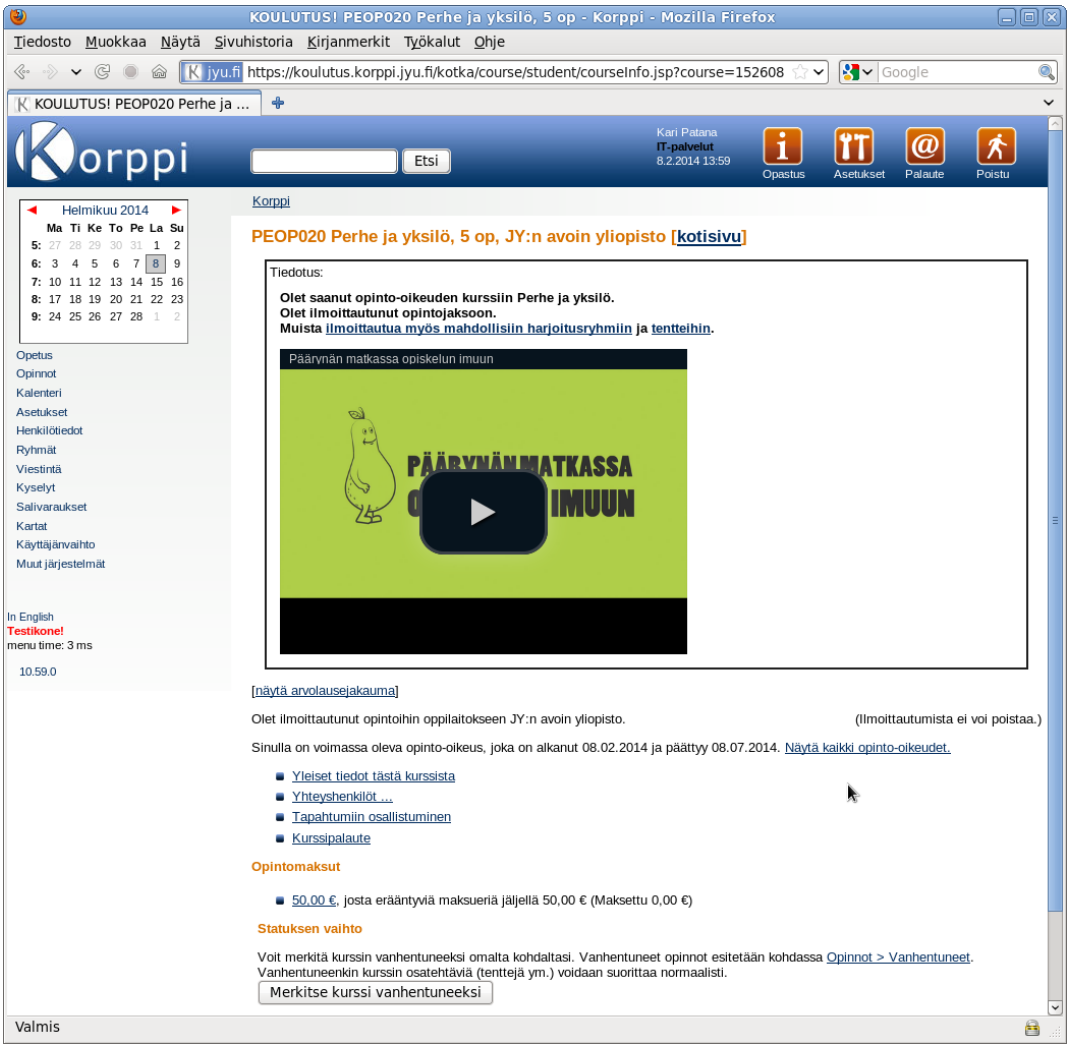
Kuvio 1. Korpin kurssisivu, kun ollaan ilmoittautumassa maksulliselle psykologian kurssille.

Ilmoittaudu-painikkeen painamisen jälkeen Korppi välittää tiedot maksuista RabbitMQ:n avulla Payments-verkkomaksuihin. Vastauksena saadaan osoite, johon asiakas ohjataan maksamaan. Payments tarjoaa opiskelijalle mahdollisuuden valita maksutavan tai perua maksun ja kurssi-ilmoittautumisen kuvan 2 mukaisesti.



Kuvio 2. Payments-verkkomaksujen sivu lasukutuksen hyväksymiseen ja maksutavan valintaan.

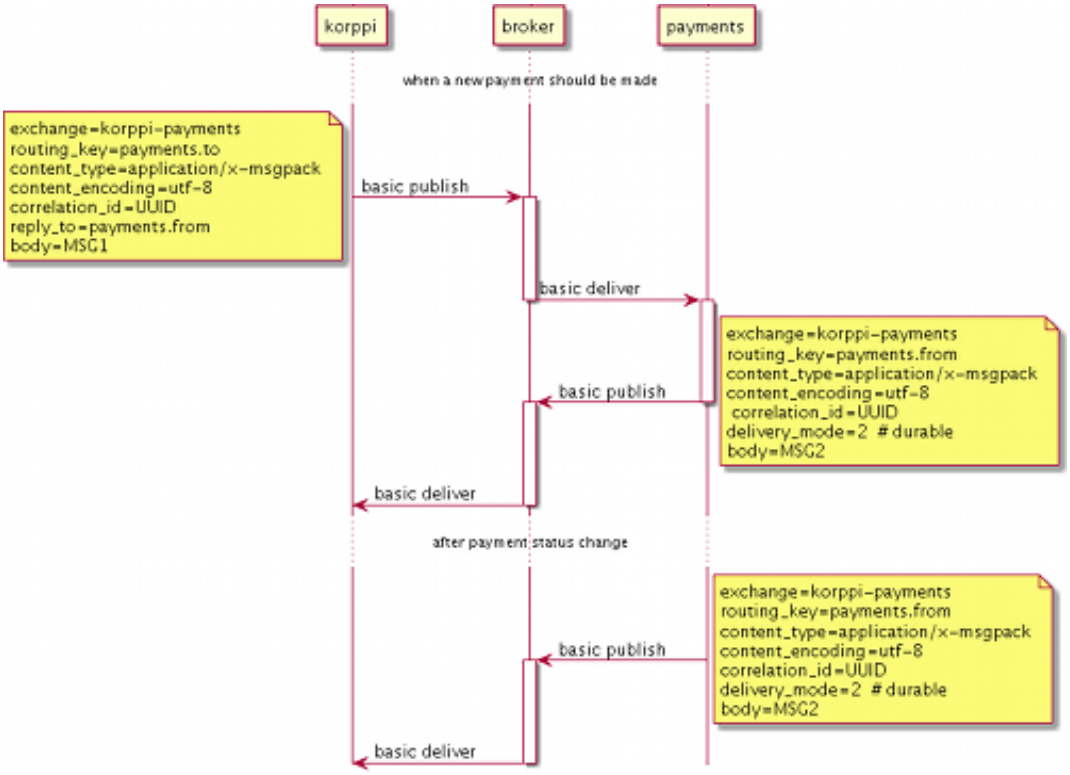
Kun opiskelija on hyväksynyt tai maksanut maksut, ohjataan hänet takaisin Korppi-_järjestelmän kurssisivulle. Kuvassa 3 opiskelija on valinnut maksavansa maksut myöhemmin verkkomaksuna tai tilisiirrolla. Korppi näyttää tiedot saadusta opinto-oikeudesta sekä maksutilanteen.



Kuvio 3. Korpin kurssisivu maksujen hyväksymisen jälkeen.

Sekvenssikaaviossa 4 kuvataan ilmoittautumisen ja verkkomaksuihin siirtymisen välissä tapahtuvaa toimintaa. Korppi tarkoittaa Korppi-järjestelmää, jossa opiskelija ilmoittautuu maksulliselle kurssille. Korppi välittää tarpeelliset tiedot, kuten kurssin ja opiskelijan nimet sekä maksun suuruuden, viestissä MSG1 RabbitMQ-välitysjärjestelmän (broker) kautta verkkomaksupalveluun (payments). Jokaista tapahtumaa varten muodostetaan correlationid-kentässä välittyvä yksilöllinen Universally

Unique Identifier (UUID) -tunniste. Paymentsin lähettämään paluuviestiin MSG2 tulee sama UUID, jonka perusteella Korppi tietää, mitä tapahtumaa viesti koskee. Paluuviestin sisältönä on osoite Payments-järjestelmään, johon käyttäjän Internet-selain ohjataan maksamista varten. Viestit on koodattu käyttäen MessagePack-protokollaa.



Kuvio 4. Sekvenssikaavio Korpin ja Paymentsin yhteydestä maksulliselle kurssille ilmoitatuduttaessa.

Toteutuksen hyvänä puolena on se, että Korpin ei juuri tarvitse tietää verkkomaksujärjestelmän olemassa olosta. Tilalle voitaisiin helposti vaihtaa toisaalta eri järjestelmä hoitamaan verkkomaksuja ja toisaalta myös toinen viestijärjestelmä. Käytön kasvaessa olisi myös mahdollista hajauttaa maksaminen usealle eri verkkomaksujärjestelmälle. Huonona puolena ilmi tuli koko järjestelmän hankala testattavuus.

6 Yhteenveto

Tutkielmassa esiteltiin syitä viestijonojen tarpeelle, viestijonojärjestelmien kehitystä, AMQP-protokollaa sekä sen version 0-9-1 toteuttavaa RabbitMQ-järjestelmää. Viestijonojen avulla voidaan rakentaa järjestelmä, jossa tiedon tarjoajan ja tarvitsijan ei tarvitse tietää toisistaan. Sen avulla voidaan jakaa vastuuta erillisille komponenteille, jotka voivat toimia toisistaan riippumatta. Se myös tarjoaa keinoja kuorman tasaamiseen ja yhtäaikaisuuden hallintaan. AMQP on avoin protokolla, joka pyrkii luomaan yhtenäisen pohjan viestijonojärjestelmille. Saman AMQP-version toteuttavia järjestelmiä ja asiakasohjelmia on helppo vaihtaa keskenään, jolloin vältytään tiettyyn ratkaisuun sitoutumiselta. RabbitMQ puolestaan on eräs AMQP version 0-9-1 toteutus. Se on tehty Erlang-kielellä, mikä tekee siitä suorituskykyisen viestijonojärjestelmän. RabbitMQ:n avulla voidaan helposti jakaa kuorma usealle eri käsittelijälle. RabbitMQ:ta hyödynnettiin Jyväskylän yliopiston Korppi-järjestelmän ja Payments-verkkomaksujen integroinnissa. Projektin perusteella RabbitMQ:n avulla saa vastuun jaettua hyvin, mutta koko järjetelmän testaaminen on hankalaa.

Kirjallisuutta

Videla, A & Williams, J. 2012. *RabbitMQ in Action*. ISBN: 9781935182979

O'Hara, J 2007. *Toward a Commodity Enterprise Middleware*. JPMorgan Queue - API Design, Volume 5 Issue 4, May-June 2007, s. 48–55.

Pivotal Software 2013. *RabbitMQ Specification*. Saatavilla WWW-muodossa <URL: http://www.rabbitmq.com/specification.html>. Viitattu 27.11.2013.

OpenAMQ 2010. *OpenAMQ Release Notes*. Saatavilla WWW-muodossa <URL: http://www.openamq.org/main:release-notes>. Viitattu 27.11.2013.

OASIS 2012. *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0*. Saatavilla PDF-muodossa <URL: http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pd
Viitattu 27.11.2013.

Armstrong, J. 2010. *Erlang*. Communications of the ACM, Volume 53 Issue 9, September 2010, s. 68–75.

AMQP Working Group 2013. *AMQP Working Group 0-9-1*. Saatavilla PDF-muodossa <URL: http://www.amqp.org/sites/amqp.org/files/amqp0-9-1.zip>. Viitattu 27.11.2013.

Soukka, A. & Ojaniemi, J. 2012. *Plone, RabbitMQ and messaging that just work*. PloneConf 2012, Arnhem, Hollanti, saatavilla WWW-muodossa <URL: https://www.slideshare.net/datakurre/plone-rabbit-mq-and-messaging-t
Viitattu 27.11.2013.

Smith, W. 2011. *An Information Architecture Based on Publish/Subscribe Messaging*. Texas Advanced Computing Center, University of Texas at Austin, TG '11 Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery, Article No. 27