PALVELUN SKAALAUS MONIASIAKASYMPÄRISTÖSSÄ

Marko Virmalainen

Opinnäytetyö
Marraskuu 2012

Ohjelmistotekniikan koulutusohjelma

Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) VIRMALAINEN, Marko	Julkaisun laji Opinnäytetyö	Päivämäärä 15.11.2012
	Sivumäärä 29	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
PALVELUN SKAALAUS MONIASIAKASYMPÄRISTÖSSÄ

Koulutusohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
PELTOMÄKI, Juha

Toimeksiantaja(t)
Codecenter Oy

Tiivistelmä

Opinnäytetyössä toteutettiin jyväskyläläiselle ohjelmistoalalalla toimivalle Codecenter Oy:lle Java-ohjelmointikielellä skaalautuva muunnospalvelu moniasiakasympäristöön, jolla muunnettiin XML-tiedostoja eri muotoihin. Palvelun toteutuksessa hyödynnettiin VMwaren vFabric-alustaa sekä avoimen lähdekoodin ratkaisuja, joista tärkein oli Spring Framework. Toteutettu palvelu koostui erillisestä hallintasovelluksesta ja viestikäsittelijästä.

Työssä vertailtiin vertikaalista ja horisontaalista skaalausta keskenään teoriatasolla. Muunnospalvelu toteutettiin noudattaen horisontaalisen skaalauksen periaatteita. Työssä käsiteltiin muistinvaraisten tietokantojen ja viestijonojen käyttämistä osana skaalausratkaisuja. Muistinvaraisena tietokantana käytettiin vFabric GemFirea, joka mahdollisti useita eri lähestysmistapoja muistinvaraisen kannan muodostamiseen. Viestijonototeutuksena taas käytettiin RabbitMQ:ta.

Muunnosten tekeminen suoritettiin Apache ServiceMix -alustalle toteutettua käsittelijäkomponenttia käyttäen. Tällöin mahdollistettiin muunnospalvelun joustava integroiminen muiden järjestelmien kanssa. ServiceMix ja sen kanssa käytetty Apache Camel tarjosivat yhdessä mittavan joukon erilaisia työkaluja integraatioita varten.

Tutkimustuloksena saatiin tietoa horisontaalisen skaalauksen soveltamisesta oikean sovelluksen toteutuksessa sekä tietoa vFabric-alustan käytöstä. Horisontaalisen skaalauksen keskeisimpänä periaatteena esitelty useisiin osiin jaottelu osoittautui käytännössä hyvin toimivaksi ratkaisuksi. vFabric-alusta tarjosi hyvät työkalut skaalautuvan palvelun toteuttamiseen.

Avainsanat (asiasanat)
GemFire, Java, RabbitMQ, skaalaus, Spring Framework, XSLT

Muut tiedot

Author(s)
VIRMALAINEN, Marko

Type of publication
Bachelor´s Thesis

Date
15112012

Pages
29

Language
Finnish

Permission for web publication
(X)

Title
SCALABLE SERVICE AIMED AT A MULTIUSER ENVIRONMENT

Degree Programme
Software Engineering

Tutor(s)
PELTOMÄKI, Juha

Assigned by
Codecenter Oy

Abstract

This bachelor's thesis describes the implementation process of a scalable transformation service given as an assignment by an IT company called Codecenter Oy. The service was aimed at a multiuser environment. The transformation service was used for transforming XML files into other formats. VMware's vFabric platform was used for implementing scalability. Other open source tools were also used, including Spring Framework.

Vertical and horizontal scaling were compared; but the main focus, however, was on horizontal scaling. The use of an in-memory database and message queues were the two most important concepts. vFabric GemFire was used as an implementation for an in-memory database and RabbitMQ as a messaging queue implementation.

Apache ServiceMix and Apache Camel were used together for providing the best way of performing transformations. They also provided a number of tools for integrating the service with other systems.

The thesis demonstrates how to apply the theory of horizontal scaling in practice. The results were promising and prove how splitting was the best way to implement horizontal scaling. vFabric platform was the most important solutions provider for scalability.

Keywords
GemFire, Java, RabbitMQ, scaling, Spring Framework, XSLT

Miscellaneous

SISÄLTÖ

KÄSITTEET JA LYHENTEET / TERMISTÖ

AMQP

Avoin protokollastandardi valmistajariippumattomalle viestijonototeutukselle.

Apache Camel

Sovelluskehys integraatioreittien toteuttamiseen.

Apache Qpid

AMQP-standardin toteuttava viestijono.

Apache Tomcat

Avoimen lähdekoodin verkkosovelluspalvelin.

JAR

Pakettitiedosto, joka sisältää Java-luokat ja resurssitiedostot.

JavaScript

JavaScript on verkkosivuilla käytettävä skriptikieli, jolla voidaan toteuttaa esimerkiksi lomakkeiden dynaaminen tarkistustoiminnallisuus.

JPA

JPA eli Java Persistence API määrittelee miten tiedot muunnetaan yhteensopiviksi Java-luokkien ja tietokannan välillä.

P2P

Vertaisverkko, jonka kaikki palvelimet voivat jakaa ja ladata tiedostoja keskenään.

REST

REST määrittelee verkkoresurssin sijainnin sekä toiminnot, jotka resurssille voidaan suorittaa.

1 TYÖN LÄHTÖKOHDAT

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantajana toimi Codecenter Oy, joka on vuonna 2003 perustettu suomalainen ohjelmistoalan asiantuntijayritys. Yrityksen erityisosaamista ovat Java-teknologiat, ja se toteuttaa tyypillisesti verkkopohjaisia tietojärjestelmiä monille eri toimialoilla toimiville yrityksille. Codecenter Oy on myös erittäin kysytty konsultti ja kouluttaja muille ohjelmistoyrityksille. (Yritys 2011.)

1.2 Työn tavoitteet

Opinnäytetyön tavoitteena oli toteuttaa moniasiakasympäristössä toimiva skaalautuva XML-tiedostojen muunnospalvelu. Palvelun tarkoituksena oli muuntaa asiakkaan toimittama XML-tiedosto asiakkaan määrittelemään muotoon, esimerkiksi PDF-tiedostoksi. Koska palvelu oli tarkoitettu monen eri asiakkaan käyttöön, piti palvelu suunnitella alusta pitäen mahdollisimman skaalautuvaksi, jotta se kykeni käsittelemään joustavasti kasvavia käyttäjä- ja tietomääriä. Palvelu toteutettiin hyödyntäen VMwaren vFabric-tuoteperheen osia toimeksiantajan vaatimuksesta, koska tarkoituksena oli saada samalla tietoa myös näiden tuotteiden käytöstä ja niiden tarjoamista ratkaisuista skaalautuvuuden suhteen.

2 TEKNIIKAT

Muunnospalvelun toteutuksessa käytettiin VMwaren vFabric-tuoteperheen osia vFabric GemFire, RabbitMQ, vFabric tc Server ja vFabric Web Server. Sovellusten ja palvelinten valvontaan käytettiin tuotteita vFabric Hyperic ja Spring Insight. Sovelluksen toteutuksessa käytettiin eri Java-teknologioita, joista tärkeimpiä olivat Apache Maven, Spring Framework ja Vaadin.

2.1 vFabric GemFire

GemStonen kehittämä ja nykyisin VMwaren omistama ja ylläpitämä vFabric GemFire on muistinvarainen tietovarasto, joka tarjoaa 10-kertaisen luku- ja kirjoitusnopeuden verrattuna perinteisiin tietokantoihin. Tietovarasto muodostuu P2P-verkosta, jossa kaikilla verkon yksiköillä on joko kokonainen tai osittainen kopio kaikkien yksiköiden varastoimista tiedoista. Tietovarasto voidaan myös toteuttaa asiakas/palvelin -ratkaisuna, jossa palvelinpää on kuitenkin edelleen P2P-verkko. (Main Features of vFabric GemFire n.d.)

GemFire takaa korkean skaalautuvuuden ja suorituskyvyn tasaamalla automaattisesti kuorman verkon muodostavien osien kesken ja jakamalla tiedot muutoksista tausta-ajona eri osille. Käytetyimmistä tiedoista on mahdollista ottaa useampiakin kopioita muille osille hakuaikojen pienentämiseksi. Tiedot voidaan myös kirjoittaa talteen levylle tai tietokantaan tausta-ajona tietyin väliajoin tietojen säilyttämiseksi kaikkien verkon muodostavien osien mahdollisen sammumisen yhteydessä. (Main Features of vFabric GemFire n.d.)

2.2 RabbitMQ

RabbitMQ on skaalautuva viestijonototeutus, joka mahdollistaa kuormantasauksen usean eri käsittelijän kesken. RabbitMQ on protokollapohjainen toteutus, joten se on helppo yhdistää mitä erilaisimpiin ohjelmistokomponentteihin mikä tekee siitä erittäin hyvän valinnan myös pilvipalveluita ajatellen. (VMware RabbitMQ: Open Source Enterprise Messaging Middleware Optimized for the Cloud Using AMQP n.d.)

RabbitMQ tukee viestien kuittaamista, minkä ansiosta viestijono tietää milloin viestit voidaan poistaa jonosta turvallisesti. Jos viestikäsittelijälle tapahtuu jotain käsittelyn aikana, RabbitMQ lähettää viestin uudelleen toiselle käsittelijälle. Viesti säilyy tällä tavoin niin kauan jonossa, kunnes se käsitellään. (Getting started with RabbitMQ. n.d.)

2.3 vFabric tc Server

vFabric tc Server on sovelluspalvelin, joka on kehitetty Apache Tomcatin päälle. Sovelluspalvelimeen on tehty mittavia parannuksia verrattuna Tomcat-palvelimeen, joista merkittävin on tc Runtime -osa, joka helpottaa sovelluspalvelimen käyttöönottoa ja mahdollistaa palvelinten luomisen erilaisia pohjia käyttäen. Pohjat voivat sisältää toiminnallisuutta aina klusteroinnista erilaisiin valvontamenetelmiin asti. Pohjia on myös mahdollista luoda itse lisää. (Usability Enhancements to Apache Tomcat n.d.)

Tc Server sopii erittäin hyvin virtualisoituun ympäristöön, koska se on optimoitu käyttämään mahdollisimman vähän resursseja (VMware vFabric tc Server. n.d.). Palvelimella pyöriviä sovelluksia on mahdollista valvoa erillisen Spring Insight -lisäosan avulla, joka mahdollistaa sovelluksen suorituskyvyn kannalta kriittisimpien kohtien paikantamisen.

2.4 Spring Insight

Spring Insight Developer on SpringSourcen kehittämä erillinen valvontasovellus, jonka avulla voidaan tutkia sovelluksen suorittamia toimintoja ja niihin kuluneita suoritusaikoja. Työkalu on suunnattu sovelluskehittäjien käyttöön, ja se tulee tc Serverin kehittäjäversion mukana valmiina insight-nimisenä pohjana. Spring Insight Developerin avulla kehittäjät voivat helposti listata kaikki sovelluksen tekemät tietokantakyselyt, verkkopyynnöt yms. suoritusaikoineen. Suoritusajoista esitetään graafi, joka voidaan pilkkoa aina pienempiin osiin ongelman syyn selvittämiseksi. Sovelluskehittäjän on kuitenkin muistettava lisätä koodiin tarvittavat Java-annotaatiot, jotta Spring Insight kykenee keräämään tietoja sovelluksesta.

Spring Insight Operations on SpringSourcen kehittämä valvontasovellus, joka sisältää samat toiminnot kuin Spring Insight Developer, mutta on tarkoitettu tuotantoympäristöön. Sen avulla voidaan valvoa sovelluksien toimintaa ja suoritusaikoja koko palvelinklusterin laajuudelta. Normaalisti Spring Insight Deve-

loper ja valvottava sovellus pyörivät saman sovelluspalvelimen päällä, jolloin valvontasovellus aiheuttaa ylimääräistä kuormaa sovelluspalvelimelle. Spring Insight Operations taas voidaan asentaa erilliselle valvontakoneelle, jonne tuotantoympäristön tc Server -sovelluspalvelimille asennetut agenttisovellukset lähettävät tietoja sovelluksista. Agenttisovellukset ovat paljon kevyempiä, joten ne eivät kuormita sovelluspalvelinta läheskään niin paljon kuin Spring Insight Developerin käyttö. (Application Cluster Monitoring with Spring Insight Operations n.d.)

2.5 vFabric Hyperic

vFabric Hyperic on valvontasovellus, jonka avulla voidaan valvoa koneiden, palveluiden ja sovelluspalvelinten toimintaa. Sillä voidaan esimerkiksi asettaa valvonta kovalevyosioille, jolloin niiden käyttämän tilan ylittäessä määritellyn rajan lähetetään tapahtumasta hälytys määriteltyihin sähköpostiosoitteisiin. Tämä mahdollistaa nopean reagoinnin tuotantoympäristössä ilmeneviin ongelmiin parantaen tarjottujen palveluiden saatavuutta. (Introduction to Hyperic Monitoring n.d.)

Uusin versio sisältää toimintoja tc Server -sovelluspalvelinten hallintaan. Niiden avulla voidaan hallita sovelluspalvelimen asetuksia ja toimintaa. Sovelluspalvelimia voidaan myös ryhmitellä hallittaviksi kokonaisuuksiksi, joille voidaan lisätä uusia versioita sovelluksista ilman käyttökatkoja. (Manage tc Server Applications n.d.)

2.6 Spring Framework

Spring Frameworkin avulla voidaan kehittää alustariippumattomia Java-sovelluksia. Spring mahdollistaa esimerkiksi lähdekoodissa esiintyvien muuttujien alustamisen XML-tiedostojen tai erilaisten annotaatioiden avulla. Spring sisältää myös lukuisia eri työkaluja esim. MVC-mallia, tietoturvaa, tietokantakäsittelyä sekä yksikkö- ja integraatiotestejä varten, joilla yksinkertaistetaan sovelluskehitystä. (Spring Framework n.d.)

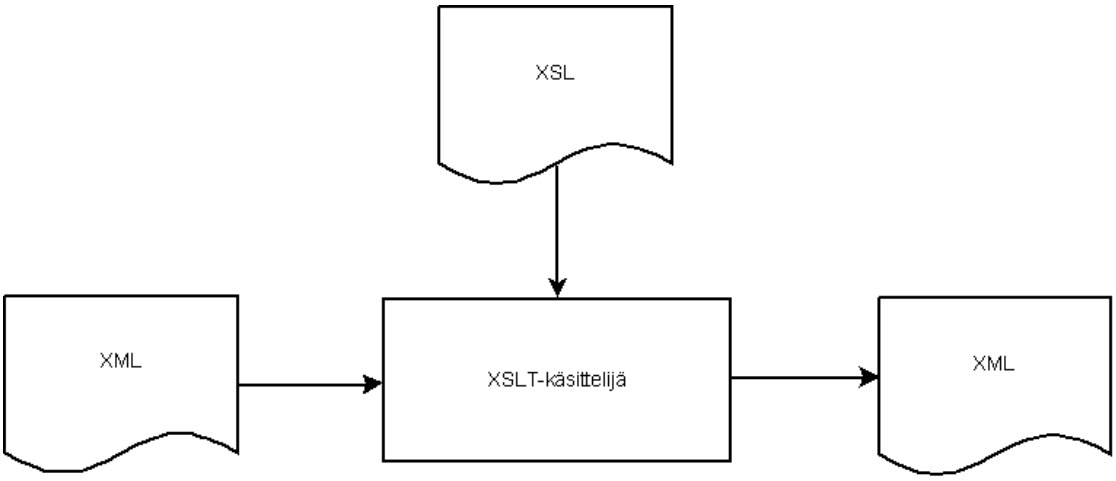
2.7 Vaadin

Vaadin on sovelluskehys, jolla voidaan kehittää käyttöliittymiä verkkosovelluksille helposti ja selainriippumattomasti. Vaadin pohjautuu GWT:hen ja JavaScriptiin, joten se ei edellytä selaimelta ylimääräisiä lisäosia. Sovelluskehys sisältää paljon valmiita käyttöliittymäkomponentteja. Käyttöliittymäohjelmoijan ei tarvitse huolehtia Java-koodissaan eri selainten pienistä eroista, vaan Vaadin pitää huolen siitä, että käyttöliittymä toimii eri selaimilla samalla tavalla. Lisäksi Vaadin sisältää valmiin toteutuksen asiakas- ja palvelinpään väliseen viestintään. (Vaadin n.d.)

GWT eli Google Web Toolkit on sovelluskehys selainpohjaisten sovellusten kehittämiseen, jota Vaadin käyttää Java-koodin muuttamisessa JavaScriptiksi, joka toimii eri selaimilla samalla tavalla. Pelkästään GWT:n avulla toteutetussa käyttöliittymässä myös sovelluslogiikka on asiakaspuolella, mikä tekee käyttöliittymän vasteajoista nopean. Vaadin taas siirtää sovelluslogiikan palvelinpäähän tehden sovelluksesta turvallisemman, koska käyttäjä ei pääse tutkimaan sovelluksen koodia selaimen avulla, mutta toisaalta myös vasteajat ovat hitaampia useiden eri palvelinkutsujen takia. (Knego 2010.)

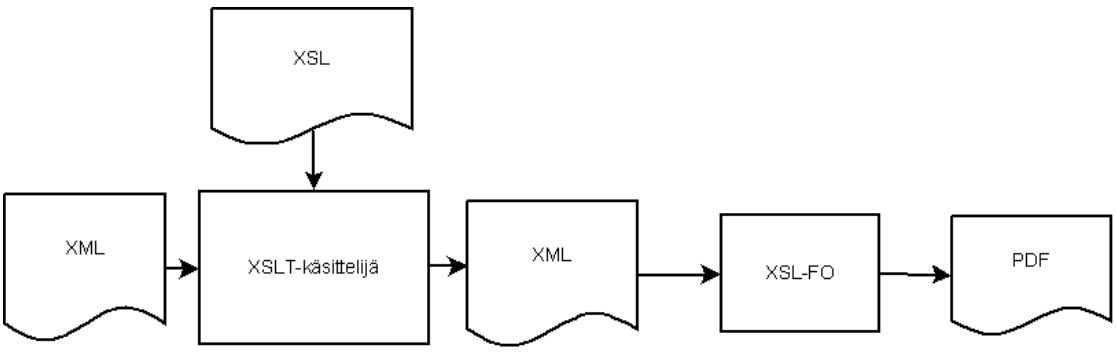
3 XML-MUUNNOSTYÖKALUT

XSLT:tä käytetään muuntamaan XML-dokumentteja toisenlaisiksi XML-dokumenteiksi. Muunnoksessa käytetään aina XSLT-tyylitiedostoa, joka määrittelee muunnossäännöt alkuperäisestä XML-dokumentista muunnettavalle uudelle XML-dokumentille. Muunnettavaa XML-dokumenttia sanotaan lähdepuuksi viitaten sen puurakennemaiseen muotoon, ja XSLT-tyylitiedostoa sanotaan tyylipuuksi. Molemmat tiedot annetaan erilliselle XSLT-käsittelijälle, joka tuottaa uuden XML-dokumentin. (Lenz n.d.)



KUVIO 1. XSLT-muunnos.

XML-tiedosto voidaan muuttaa myös PDF-dokumentiksi. Muunnosprosessi aloitetaan samalla tavalla kuin muunnettaessa XML-dokumentti toiseksi XML-dokumentiksi, mutta XSLT-muunnoksesta saatu XML-dokumentti annetaan vielä eteenpäin erilliselle XSL-FO -käsittelijälle, joka tuottaa lopullisen PDF-dokumentin. XSL-FO kattaa muunnoksen lisäksi myös muotoilun. Muunnoksessa käytettävässä dokumentissa pitää olla mukana myös PDF-dokumenttiin halutut muotoilut. (XSL-FO Introduction n.d.)



KUVIO 2. XML-dokumentin muunnos PDF-dokumentiksi.

XPath on oleellinen osa XSLT:tä. Sen avulla XML-dokumentin puurakenteesta voidaan valita eri osia mitä moninaisimmilla tavoilla. Yksinkertaisimmillaan do-

kumentista voidaan valita tietyn elementin nimen perusteella elementin sisältönä oleva teksti.

4 SKAALAUTUVUUS

4.1 Vertikaalinen ja horisontaalinen skaalaus

Sovelluksen skaalautuvuudella tarkoitetaan sovelluksen kykyä sopeutua kasvaviin käyttäjä- ja tietomääriin ja sen suorituskykyä kasvavan kuormituksen alla. Skaalautuvuus voidaan saavuttaa kahdella eri tavalla. Näistä kahdesta tavasta vertikaalinen skaalaus on aikaisemmin yleisesti käytetty, mutta sen rinnalle on noussut horisontaalinen skaalaus. (Terrill 2007.)

Vertikaalinen skaalaus tarkoittaa tehokkaampien palvelinten hankkimista tai palvelimen muistin lisäämistä suorituskyvyn parantamiseksi kuorman kasvaessa. Tällainen tapa on usein hyvin yksinkertaisesti hallittavissa laitteistotasolla eikä myöskään vaadi muutoksia sovelluksen lähdekoodiin. Huonoina puolina vertikaalisessa skaalauksessa ovat kuitenkin joustamattomuus ja hinta. Jos kuormaa ei aina olekaan yhtä paljon kuin tilanteessa, jolloin päädyttiin hankkimaan tehokkaampia laitteita, näiden laitteiden tehoa ei voi hyödyntää muuhun käyttöön. Myöskin kustannukset kasvavat, jos joudutaan hankkimaan koko ajan tehokkaampia laitteita. Jossain vaiheessa saavutetaan kuitenkin raja siinä, kuinka tehokkaita laitteita voidaan hankkia. (Terrill 2007.)

Horisontaalinen skaalaus tarkoittaa kuorman jakamista useammalle pienelle palvelimelle. Tällainen tapa on hyvin joustava, koska palvelimia voidaan lisätä, vähentää ja muokata kuorman vaihdellessa. Etuina ovat myös halpuus ja tehokkuus. Lisäksi saatavuus paranee tällaisella ratkaisulla, koska erilliset palvelimet ovat eristyksissä toisistaan, jolloin yhden palvelimen kaatuminen ei vaikuta muiden palvelinten toimintaan. Huonona puolena on kuitenkin hallittavuuden kasvava monimutkaistuminen useiden erillisten palvelinten takia. (Terrill 2007.)

4.2 Virtualisointi

Virtualisoinnin avulla voidaan parantaa skaalautuvuutta, koska virtuaalikoneita voidaan lisätä, vähentää ja muokata helposti. Virtualisointi myös parantaa tehokkuutta, koska palvelimet voivat toimia huipputeholla virtualisoidussa ympäristössä. Virtualisoinnin taloudellinen kannattavuus tulee pienemmästä fyysisten laitteiden määrästä, tila- ja jäähdytyskustannusten vähenemisestä sekä helposta hallittavuudesta. (Virtualization Use Case: Application Scaling using Virtualization 2008.)

Virtualisoinnin etuina ovat luotettavuus ja helppokäyttöisyys. Virtuaalikoneet ovat yleensä eristyksissä toisistaan, joten niiden toiminnallisuus ei vaarannu yhden virtuaalikoneen kaatuessa. Helppokäyttöisyys taas tarkoittaa virtuaalikoneiden keskitettyä hallintaa. (Virtualization Use Case: Application Scaling using Virtualization 2008.)

5 MUUNNOSPALVELU

Toteutetun palvelun vaatimuksina olivat mahdollisimman hyvä skaalautuvuus ja helppo integrointi muiden sovellusten kanssa. Joustava integrointi muiden järjestelmien kanssa mahdollistettiin siten, että sovellus tarjosi muunnostoimintonsa REST-palveluna. Kutsuttavan toiminnon tehtävänä oli välittää XML- ja XSLT-tiedostot sekä muunnoksen tuottaman tiedoston tyyppi (XML, PDF) ja kohdeosoite työjonokäsittelijälle, joka toteutettiin OSGi-komponenttina. OSGi-komponenttitoteutus oli paras tapa kehittää yksinkertainen käsittelijä, joka voitiin ajaa ServiceMixissä, ja jonka riippuvuuksia muihin komponentteihin pystyttiin hallitsemaan helposti ServiceMixin avulla. OSGi-komponentin vastuulla oli XSLT-muunnoksen suorittaminen ja valmiin tiedoston lähettäminen määriteltyyn kohdeosoitteeseen, joka saattoi olla esimerkiksi työjono, sähköposti- tai webservice-osoite.

5.1 Hallintasovellus

Toteutetun hallintasovelluksen avulla hallittiin käyttäjätietoja sekä käyttäjien muunnostietoja. Sovellus haluttiin pitää alkuvaiheessa mahdollisimman yksinkertaisena, joten käyttäjätietoihin sisältyi ainoastaan nimi. Muunnostietojen osalta taas tarvittiin vain viittaus käyttäjätiliin, muunnosoperaation tuottaman tiedoston tyyppi, muunnoksessa käytettävä XSLT-tiedosto sekä osoite, johon muunnos lähetetään. Molemmille tiedoille toteutettiin hallintaa varten toiminnot lukemiselle, lisäämiselle, muokkaamiselle ja poistamiselle. Näitä toimintoja kutsutaan lyhyesti CRUD-toiminnoiksi.

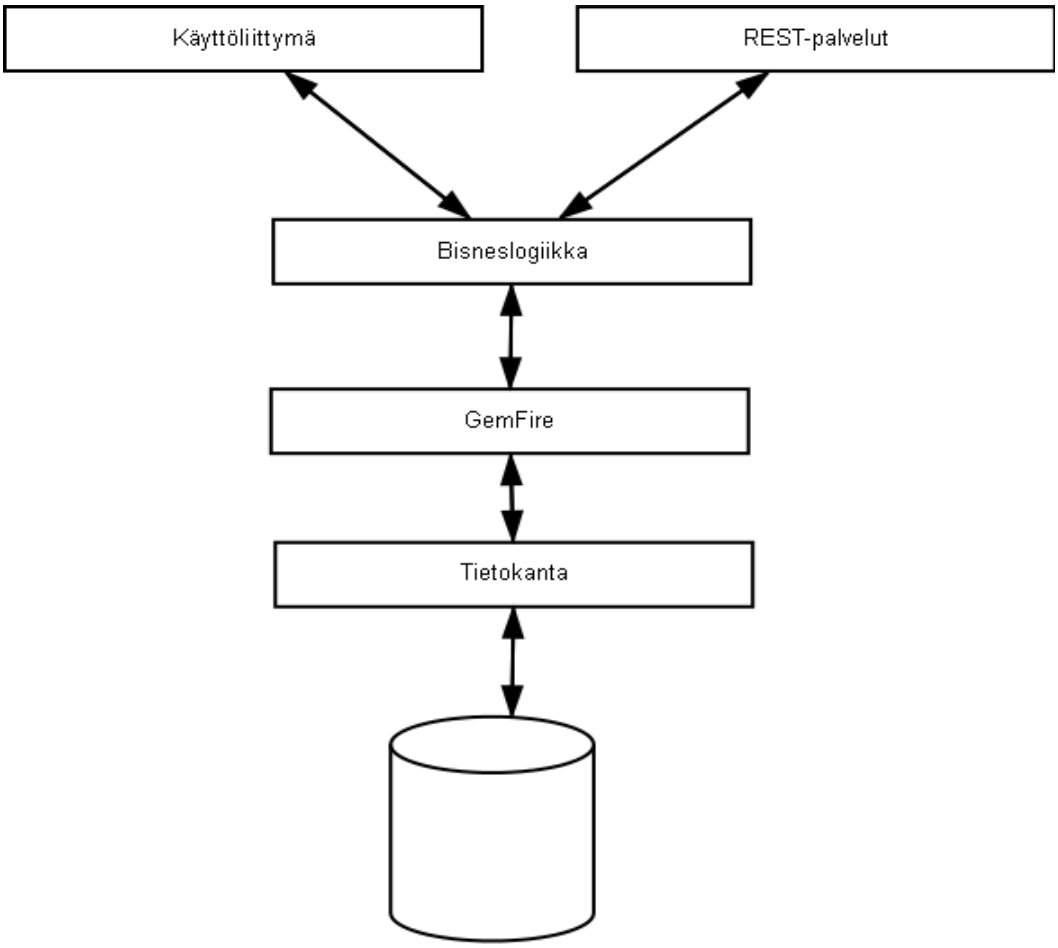
Sovellus jaettiin viiteen eri osaprojektiin:

- API (malliluokat, rajapinnat)
- CORE (sovelluslogiikka, asetustiedostot)
- SQL (tietokantaskriptit)
- WS (REST-palvelut)
- UI (käyttöliittymätoteutus).

API-projekti sisälsi sovelluksen malliluokat käyttäjätileille ja muunnoksille, ja rajapintamääritykset toiminnoille. CORE-projekti taas sisälsi API-projektissa määriteltyjen rajapintojen toteutukset sekä keskeiset Spring XML -asetustiedostot. SQL-projektiin laitettiin tarvittavat tietokantaskriptit sekä asetukset erityistä Flyway-lisäosaa varten, joka huolehti tietokantamuutosten tekemisestä hallitusti ja automaattisesti. WS-projekti sisälsi REST-palvelun määrittelyn ja toteutuksen, ja UI-projekti taas sisälsi hallintasovelluksen käyttöliittymätoteutuksen, joka toteutetttiin Vaadin-komponenteilla.

Hallintasovelluksen kerrosarkkitehtuuri koostui neljästä eri kerroksesta. Alimpana tasona oli tietokantakerros, joka viesti tietokannaksi valitun PostgreSQL-tietokannan kanssa. Tietokanta- ja bisneslogiikkakerroksen välille oli lisätty oma kerros GemFiren käyttöä varten, jolla GemFire haki tarvittaessa tietoja tietokantakerrokselta bisneslogiikkakerrokselle ja välitti bisneslogiikkakerrokselta tulleet tiedot tietokantakerrokselle tausta-ajona. Bisneslogiikkakerrok-

seen oli sijoitettu nimensä mukaisesti kaikki sovelluksen varsinainen toiminnallisuus liittyen käyttäjä- ja muunnostietojen hallintaan sekä viestimiseen viestijonon kanssa. Näiden kerrosten päälle oli vielä laitettu omat rinnakkaiset kerroksensa käyttöliittymälle ja REST-palveluille viestimään bisneslogiikkakerroksen kanssa. Kerrosarkkitehtuurin avulla varmistettiin, että eri kerrokset eivät olleet liian sidoksissa toisiinsa.



KUVIO 3. Hallintasovelluksen kerrosarkkitehtuuri.

5.2 Käsittelijä

Käsittelijän tehtävänä oli lukea työjonosta tulevia muunnostehtäviä ja käsitellä ne sekä lähettää tiedot eteenpäin käyttäjän muunnostiedoissa määriteltyyn

kohdeosoitteeseen. Käsittelijä toteutettiin omana komponenttinaan, jotta käsittelijöitä voitiin tarvittaessa käynnistää useampia. Toteutus olisi voinut olla pelkkä komentoriviltä käynnistettävä JAR-paketti, mutta PDF-tiedoston tuottaminen itsessään vaati jo oman riippuvuutensa kolmannen osapuolen komponenttiin, joten JAR-paketin käynnistäminen komentoriviltä ei olisi ollut enää niin suoraviivaista. Tästä syystä käsittelijä päätettiin toteuttaa OSGi-komponenttina, jolloin se voitiin ajaa ServiceMixissä ja sen riippuvuuksia voitiin hallita ServiceMixin Karaf-komponentin avulla, joka tarjosi tarvittavan OSGi-toiminnallisuuden. (Raehalme 2012.)

OSGi-järjestelmä on erillisistä moduuleista koostuva järjestelmä, jonka yksittäinen moduuli paljastaa vain erikseen määritellyt toimintonsa muille moduuleille. Toiset järjestelmän moduulit voivat määritellä erikseen tarvitsemansa toisen moduulin toiminnot erillisellä import-määrittelyllä. OSGi-järjestelmä on dynaaminen ympäristö, jonka moduulit sopeutuvat saatavilla oleviin moduuleihin, jolloin yksittäinen moduuli voi toimia ilman kaikkia tarvittavia toimintoja. Moduulin palvelut rekisteröidään erilliseen rekisteriin, jonka avulla muut moduulit etsivät tarvitsemansa palvelut. Tärkeimpänä OSGi-järjestelmän ominaisuutena on mahdollisuus asentaa, poistaa, pysäyttää ja päivittää moduuleita sammuttamatta koko järjestelmää. (The OSGi Architecture n.d.)

Toteutuksessa hyödynnettiin Apache Camel -sovelluskehystä, jonka avulla voitiin määritellä reittejä, joita pitkin käsiteltävä tieto kulki. (Ks. kuvio 4.) Camel tarjosi myös omat luokkansa XSLT-muunnosten tekemiseen sekä dynaamisen lähetyksen eri kohteisiin käyttäen Recipient List -mallia (Dynamic Recipient List n.d.). Versiosta 2.9.0 lähtien Camel tukee dynaamisia XSLT-muunnoksia CamelXsltResourceUri-otsakkeen avulla, johon voidaan asettaa käytettävä XSLT-tiedosto (Dynamic stylesheets n.d.).

```
<camelContext id="routes" xmlns="http://camel.apache.org/schema/spring">
    <route id="transformations">
        <from uri="spring-amqp:direct:transformationQueue:*?type=direct&amp;autodelete=false&amp;durable=true&amp;prefetchCount=1" />
        <to uri="consumer" />
        <to uri="xslt:placeholder.xsl?contentCache=false" />
        <choice>
            <when>
                <xpath>$type = 'PDF'</xpath>
                <setProperty propertyName="headerProperty">
                    <header>destination</header>
                </setProperty>
                <setProperty propertyName="typeProperty">
                    <header>type</header>
                </setProperty>
                <to uri="fop:application/pdf" />
                <setHeader headerName="destination">
                    <property>headerProperty</property>
                </setHeader>
                <setHeader headerName="type">
                    <property>typeProperty</property>
                </setHeader>
            </when>
            <otherwise />
        </choice>
        <recipientList onPrepareRef="destinationProcessor">
            <header>destination</header>
        </recipientList>
    </route>
</camelContext>
```

KUVIO 4. Camelin reittimääritys.

Camel tukee AMQP-protokollaa, mutta toteutus on tehty käyttäen Apache Qpidia. Spring tarjoaa kuitenkin oman tukensa AMQP:lle ja RabbitMQ:lle. Ainoaksi selvitettäväksi asiaksi jäi RabbitMQ:n ja Camelin välinen viestiminen, mutta tätä tarkoitusta varten oli jo olemassa Bluelockin camel-spring-amqp -lisäosa, joka ratkaisi tämän ongelman. (Camel and RabbitMQ n.d.)

6 GEMFIRE OSANA SOVELLUSKEHITYSTÄ

Osana hallintasovellusta käytetty muistinvarainen kanta koostui GemFiren välimuistipalvelimien (cacheserver) muodostamasta P2P-verkosta, joka on kaikkien GemFire-järjestelmien perusmalli. Tästä perusmallista oltaisiin voitu laajentaa järjestelmä sovellustasolla vielä tarvittaessa erillisiin asiakas- ja palvelinkerroksiin GemFiren osalta, jos oltaisiin haluttu esimerkiksi rajata asiakaspuolen käyttämiä tietoja (Storage and Distribution Options n.d.). Rajaukselle ei kuitenkaan ollut vielä tarvetta, joten se jätettiin tekemättä. Tämä ratkaisu myös yksinkertaisti tarvittavia GemFiren asetuksia.

Järjestelmässä olevat palvelimet olisivat voineet myös sijaita fyysisesti eri paikoissa muodostaen WAN-järjestelmän, joka tukee hyvin horisontaalista skaalautuvuutta, sillä fyysisesti eri paikoissa sijaitsevien järjestelmien sidonnaisuus toisiinsa on melko löyhä (Topology Types n.d.). Tutkimukseen valittiin kuitenkin yksinkertaisuuden vuoksi toteutusmalliksi samalla palvelimella sijaitsevat välimuistipalvelimet.

Järjestelmän osat voivat muodostaa yhteyden toisiinsa joko UDP/IP-ryhmälähetyksen tai GemFiren oman paikantimen (locator) avulla. Osat viestivät keskenään itsenäisesti kun yhteys on saatu muodostettua. Paikantimia voi olla useita, ja ne ovat suositeltu yhteydenhallintatapa. Paikantimet valittiin osaksi toteutusta edellä mainituista syistä. (How Member Discovery Works n.d.)

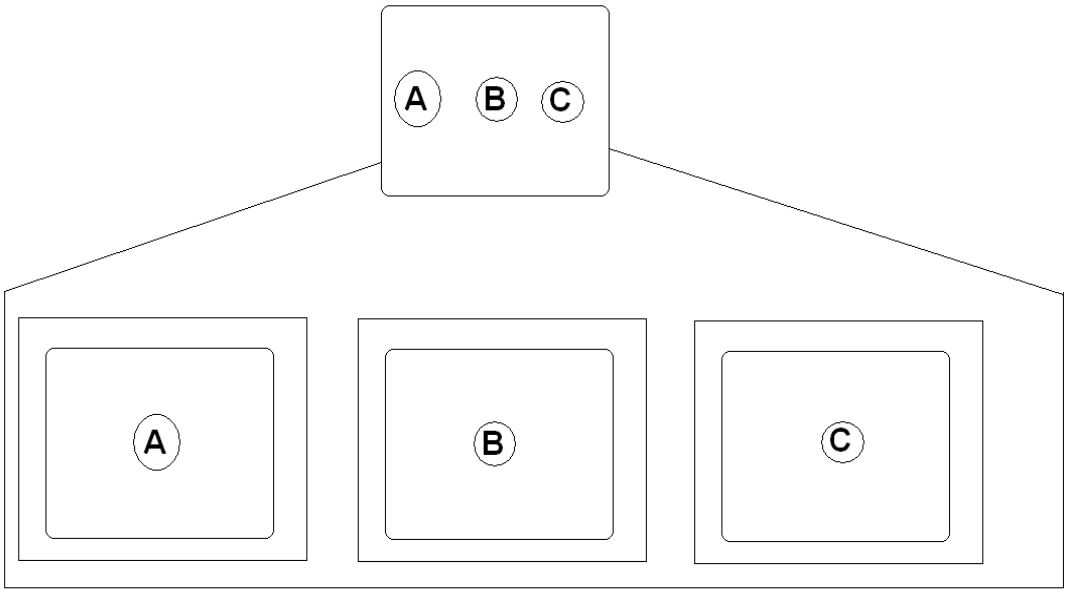
6.1 Alueet

GemFiressa tieto varastoidaan alueisiin (region), joiden tyyppi on joko kopioitu (replicated) tai ositettu (partitioned). Kopioitu alue sisältää kaikkien useammalle palvelimelle jaetun järjestelmän osien tiedot yhdellä palvelimella erillisenä kopiona. Tiedot kopioidaan myös järjestelmän kaikille muille palvelimille. Kopioitu alue soveltuu parhaiten pienille tietomäärille, sillä yhden välimuistipalvelimen tulee kyetä tallentamaan koko muistinvaraisen kannan tiedot. Kopioidun alueen etuina ovat pienin viive tietojenkäsittelyssä verrattuna muihin aluetyyppeihin sekä tietojen jakaminen tausta-ajona muille välimuistipalvelimille. (Region Types n.d.)

Ositetun alueen varastoima tieto on jaettu useiden eri välimuistipalvelimien välillä, jotka muodostavat jaetun järjestelmän. Tieto varastoidaan ns. ämpäriin (bucket), jota käytetään kuvaamaan alueen varastointiyksikköä. Ämpärit jaetaan alueen muodostavien välimuistipalvelinten kesken. Tällöin kaiken tiedon ei tarvitse mahtua yhden jäsenen muistiin, joten ositettu alue soveltuu parhaiten suurille tietomäärille. Etuna on myös tiedon korkea saatavuus, koska yksittäisen jäsenen varastoimista tiedoista voidaan ottaa haluttu määrä kopioita, jotka siirretään toisten jäsenien hallittaviksi. Jos yksi välimuistipalvelin kaatuu,

niin tiedot ovat tallessa toisen jäsenen muistissa. Ositettu alue myös skaalautuu suuriin tietomääriin sekä takaa hyvän kirjoitusnopeuden, koska kirjoitusoperaation aikana siirrettävän tiedon määrä ei kasva jäsenmäärän mukaan kuten kopioituja alueita käytettäessä tapahtuu. (Region Types n.d.)

Hallintasovelluksessa käytettäväksi aluetyypiksi valittiin ositettu alue, jotta voitiin varautua kasvaviin tietomääriin parhaiten. Ositetulle alueelle voidaan suorittaa kuormantasaus välimuistipalvelinten kesken, jolloin ämpäreitä siirrellään eri välimuistipalvelimien välillä. Kuormantasauksessa pitää kuitenkin huomioida transaktiot, sillä jos jokin jäsen on kuormantasauksen aikana osallisena transaktiossa, niin transaktio epäonnistuu (Rebalancing Partitioned Region Data n.d.). Koska sovelluksessa käytettiin transaktioita, niin kuormantasausta ei kannattanut suorittaa käsin. Öisin suoritettava ajastettu Cron-ajo olisi voinut olla hyvä vaihtoehto, mutta silloin käyttäjien olisi pitänyt olla samalla aikavyöhykkeellä sekä käyttää järjestelmää vain päiväsaikaan.



KUVIO 5. Ositettu alue.

6.2 Spring Data GemFire

Hallintasovelluksen kehityksessä käytettiin SpringSourcen Spring Data GemFirea GemFiren asetustenhallinnassa. Spring Data GemFire helpottaa GemFiren käyttöönottoa tarjoamalla valmiit CRUD-toimintojen toteutukset, oletusasetukset GemFiren hallintaan sekä helpon tavan asetusten muuttamiselle hyödyntäen Spring-sovelluskehyksen keskeisimpiä ominaisuuksia, kuten IoC-säiliötä ja property placeholder -toiminnallisuutta. (Ks. kuvio 6.) Spring-sovelluskehyksen käyttö on nykyään suositeltu tapa GemFiren asetusten säätämiseen. Käyttö vaatii ainoastaan joko uuden Cache-olion, joka voidaan luoda asetustiedostossa cache-elementin avulla, tai yhdistämällä olemassaolevaan Cache-olioon. (Gierke, Leau & Turanski n.d.)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gf="http://www.springframework.org/schema/gemfire"
    xmlns:dgf="http://www.springframework.org/schema/data/gemfire"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/gemfire http://www.springframework.org/schema/gemfire/spring-gemfire.xsd
        http://www.springframework.org/schema/data/gemfire http://www.springframework.org/schema/data/gemfire/spring-data-gemfire.xsd">
    <gf:cache properties-ref="props" />

    <bean id="props" class="org.springframework.beans.factory.config.PropertiesFactoryBean">
        <property name="location">
            <value>${cache.location}</value>
        </property>
    </bean>

    <gf:transaction-manager id="tx-manager" />

    <gf:partitioned-region id="accounts" recovery-delay="10000" total-buckets="3" statistics="true">
        <gf:cache-loader ref="cacheLoader" />
        <gf:cache-writer ref="cacheWriter" />
    </gf:partitioned-region>

    <gf:partitioned-region id="transformations" recovery-delay="10000" total-buckets="3" statistics="true">
        <gf:cache-loader ref="cacheLoader" />
        <gf:cache-writer ref="cacheWriter" />
    </gf:partitioned-region>

    <bean id="cacheLoader" class="fi.codecenter.xmlconverter.cache.JpaCacheLoader" />
    <bean id="cacheWriter" class="fi.codecenter.xmlconverter.cache.JpaCacheWriter" />

    <dgf:repositories base-package="fi.codecenter.xmlconverter.dao" />

    <bean id="jpaLoader" class="fi.codecenter.xmlconverter.cache.JpaLoader">
        <property name="cache" ref="gemfireCache" />
    </bean>

</beans>
```

KUVIO 6. Spring Data GemFiren asetukset.

Spring Data GemFire tarjoaa versiosta 1.2.0 lähtien mahdollisuuden merkitä alueisiin varastoitavat entiteetit @Region-annotaation avulla sekä tuen reposi-

tory-toiminnallisuudelle. Repository-luokat toteuttavat perinteiset CRUD-toiminnot sekä mahdollistavat omien kyselyiden lisäämisen. Yksinkertaisimmallaan toimintoja voi käyttää koodissa luomalla rajapinnan, joka peritään GemfireRepository-rajapinnnasta. Tämän lisäksi voidaan vielä lisätä erilaisia findBy-toimintoja, jotka muodostavat kyselyn automaattisesti nimessä määriteltyjen olion kenttien perusteella. Näiden lisäksi voidaan vielä lisätä toiminnot monimutkaisemmille kyselyille @Query-annotaation avulla, jolloin kysely voidaan kirjoittaa itse. (Ks. kuvio 7.)

```
/*
 * $Id$
 * Copyright (c) Codecenter Oy. All rights reserved.
 *
 * This software is the proprietary information of Codecenter Oy.
 * Use is subject to license terms.
 */
package fi.codecenter.xmlconverter.dao;

import org.springframework.data.gemfire.repository.GemfireRepository;
import org.springframework.stereotype.Repository;

import fi.codecenter.xmlconverter.model.Account;

/**
 * @author <a href="mailto:marko.virmalainen@codecenter.fi">Marko
 *         Virmalainen</a>
 */
@Repository
public interface AccountDAO extends GemfireRepository<Account, Long> {
    Account findByName(String name);
}
```

KUVIO 7. Käyttäjätilin Repository-rajapinta.

GemFiren muistiin tallennettu tieto oli mahdollista kirjoittaa pysyvämpään varastointipaikkaan tausta-ajona GemFiren oman CacheWriterAdapter-luokan avulla, joka mahdollistaa tietojen tallentamisen mihin tahansa muotoon. Tutkimuksessa tiedot tallennettiin yksinkertaisuuden vuoksi avoimen lähdekoodin PostgreSQL-tietokantaan käyttäen Spring Data JPA:n luokkia, jotka tarjoavat samantyyppiset toiminnot kuin Spring Data GemFirenkin luokat. Tiedot ladattiin tallennusjärjestelmästä CacheLoader-rajapinnan toteuttavan luokan avulla. Tiedot olisi myös voitu tallentaa esimerkiksi suoraan levylle, koska tiedostojär-

jestelmä on kaikkein halvin ja joustavin vaihtoehto skaalaukseen (Abbot & Fisher 2011, 55-56). Esteenä tälle oli entiteettien väliset riippuvuudet toisistaan.

6.3 GemFiren vaihtoehdot

Vaihtoehtona GemFiren käytölle olisi ollut tietojen tallentaminen johonkin NoSQL-kantaan. Tällöin sovellus ei olisi enää tarvinnut erillistä kerrosta muistinvaraiselle kannalle. Luku- ja kirjoitusnopeus olisi ollut todennäköisesti hiukan nopeampi kuin GemFirella saavutettu. Koska sovelluksen taulujen väliset suhteet eivät olleet monimutkaisia, niin NoSQL-kannan käyttäminen olisi ollut suositeltavaa (Abbot & Fisher 2011, 61). NoSQL-kannan käyttäminen tallennusratkaisuna ei olisi aiheuttanut ongelmia sovelluskehityksen näkökulmasta, koska sovellus tehtiin kokonaan uutena toteutuksena alusta asti. Valmiiseen sovellukseen kantaratkaisun muuttaminen ei olisi toisaalta ollut yhtä helppoa kuin esimerkiksi GemFiren lisääminen, joka olisi vaatinut vain yhden kerroksen lisäämistä bisneslogiikka- ja tietokantakerroksen väliin kerrosarkkitehtuurissa.

NoSQL-kannat on suunniteltu skaalautumaan horisontaalisesti ja toimimaan halvemmillakin laitteilla. Niiden tietomalli on myös hyvin joustava, jolloin erilaisten tietomallimuutosten tekeminen ei aiheuta suurta lisätyötä. Aikaisemmin ongelmana on kuitenkin ollut teknologian nuoruus sekä teknisen tuen puuttuminen, koska monet NoSQL-kannat ovat avoimen lähdekoodin projekteja. (Harrison, G 2010)

Koska sovelluksessa käytettiin JPA:n toteutuksena Hibernatea, niin toinen vaihtoehto GemFirelle olisi tällöin ollut Hibernaten oman 2. tason välimuistin käyttäminen. Hibernatessa on kaksi eri välimuistitasoa, joista ensimmäinen istunto-olioon liittyvä taso on oletuksena aina päällä. Toinen taso käsittää koko istunto-olioita luovan luokan, joka lataa esimerkiksi tietokannasta haetut tiedot välimuistiin välttääkseen ylimääräiset tietokantakutsut samaan tapaan kuin GemFire. Toteutuksena tälle 2. tason välimuistille olisi voitu käyttää esimerkiksi Terracottan EhCachea. (Cheung 2008.)

7 RABBITMQ OSANA SOVELLUSKEHITYSTÄ

AMQP:n määrittelemässä mallissa tuottajalta (producer) tulevat viestit menevät aina välittäjän (exchange) kautta, joka välittää tiedot eteenpäin siihen reititysavaimen (route key) perusteella liitettyihin jonoihin. Välittäjille on määritelty neljä eri tyyppiä, joista direct on oletusvälittäjätyyppi. Sen tarkoitus on välittää viesti aina suoraan nimettyyn tai satunnaisesti nimettyyn viestijonoon. Viestijonolle annetaan satunnainen nimi, jos se luodaan viestin lähettämisen aikana ilman erillistä nimimääritystä. Toinen tyyppi on fanout, joka jakaa viestit kaikille mahdollisille jonoille. Kolmas tyyppi on topic, joka jakaa viestit määritellyn aiheen perusteella jonoihin. Tällöin sama viestijono voi vastaanottaa viestejä useilta eri välittäjiltä. Neljäs tyyppi, header, käyttää otsakkeita ja on toiminta-periaatteeltaan sama kuin direct, mutta paljon hitaampi. (Videla & Williams 2012, 20-23.)

AMQP määrittelee myös, että viestijonopalvelimeen avataan vain yksi TCP-yhteys. Tämän yhden yhteyden sisällä voidaan muodostaa useita kanavia viesteille, jolloin viestijono kuluttaa vähemmän järjestelmäresursseja luodessaan vain yhden TCP-yhteyden riippumatta säiemäärästä. Mallin ansiosta myös viestien välitys on nopeaa useiden eri kanavien ansiosta. Kanavalla tarkoitetaan yhtä TCP-yhteyden sisällä olevaa säiettä. (Videla ym. 2012, 14-16.)

Virtuaalipalvelimet (virtual host) kuuluvat osana AMQP-protokollaan. Virtuaalipalvelimella tarkoitetaan yhden viestijonopalvelimen sisällä olevia virtuaalisia palvelimia, jotka on eristetty toisistaan. Virtuaalipalvelimelle voidaan rajata omat käyttäjänsä, käyttöoikeudet näille käyttäjille, välittäjät sekä jonot. Tällöin sama viestijonopalvelin voi hallita useamman sovelluksen viestijonoja pitäen ne kuitenkin erillään toisistaan. Tämä helpottaa myös sovellusten viestijonotoiminnallisuuksien siirtämistä erillisille viestijonopalvelimille. (Videla ym. 2012, 24-25.)

RabbitMQ on toteutettu käyttäen Erlang-ohjelmointikieltä, minkä ansiosta RabbitMQ on saatu optimoitua tehokkaaksi AMQP-protokollan toteuttavaksi viestijonopalvelimeksi (Videla ym. 2012, 7). RabbitMQ -palvelin käyttää Mnesia-tie-

tokantaa, joka on NoSQL-kanta, metatietojen tallentamiseen. Asiakassovelluksille on tehty toteutuksia useilla eri ohjelmointikielillä, esim. C++, Java, PHP ja Python.

Palvelinsovelluksessa RabbitMQ otettiin käyttöön hyödyntäen Spring AMQP -moduulia, joka tarjosi käyttövalmiit luokat oletusasetuksilla AMQP-protokollan määrittelyn toteuttavan jakelijan käyttöä varten. Spring AMQP tarjosi myös hyvät mahdollisuudet omien muutosten tekemiseen tarjoamalla samat asetusmahdollisuudet, jotka olisi saatu myös käyttämällä suoraan RabbitMQ:n omia Java-luokkia. Palvelinpäässä määriteltiin ainoastaan käytettävät tunnukset, palvelin, porttinumero sekä virtuaalipalvelin. Spring AMQP -moduulin toteutukseen on käytetty vain RabbitMQ:ta, jonka toteuttaman AMQP-protokollan versio on 0.9.1 (Fisher 2012). Tästä johtuen Spring AMQP ei toimi vielä Apache Qpidin kanssa, koska Apache Qpid toteuttaa jo version 1.0 AMQP-protokollasta.

Vaihtoehtoinen tapa AMQP-protokollan mukaiselle työjonototeutukselle olisi ollut Apache Qpidin käyttö, jolla myös Camelin oma AMQP-komponentti on toteutettu. Sovelluksessa päädyttiin kuitenkin käyttämään RabbitMQ:ta toimeksiantajan vaatimuksesta. Apache Qpidin käyttö olisi ainakin vähentänyt tarvittavia muutoksia käsittelijän toteutuksen yhteydessä.

8 KLUSTEROINTI

Hallintasovellus asennettiin samalla CentOS-palvelimella oleville kahdelle tc Server -sovelluspalvelimelle, joista muodostettiin oma pieni sovelluspalvelinklusteri klusteroinnin testaamista varten. vFabric Web Server toimi klusterin kuormantasaajana jakaen käyttäjät tasaisesti eri sovelluspalvelimille. Web Serverin asetustiedostoihin tehtiin tarvittavat muutokset kuormantasausta ja klusterointia varten määrittelemällä käytettävä kuormantasausalgoritmi, sovelluspalvelimet sekä tiedon siitä ovatko istunnot sidoksissa sovelluspalvelimeen. tc Server -palvelimet taas luotiin erityistä klusterointipohjaa käyttäen, jolloin

tarvittavat klusterointiasetukset saatiin tehtyä automaattisesti. Klusterointi ja istuntokopiointi sovelluspalvelinten välillä takasivat sen, että jonkin klusteriin kuuluvan sovelluspalvelimen sammuessa tai kaatuessa käyttäjien istunnot ohjattiin automaattisesti muille klusterin sovelluspalvelimille. Ideaalitilanteessa käyttäjän ei olisi pitänyt huomata minkäänlaista katkosta, mutta testauksessa ilmeni aivan muutaman sekunnin käyttäjille näkyviä käyttökatkoja. Testauksen aikana ei kuitenkaan päästy selvyyteen siitä johtuiko tämä hitaista vasteajoista vai johonkin itse hallintasovellukseen liittyvästä seikasta.

GemFiren klusteroinnissa paikannin (locator) toimi kuormantasaajana välimuistipalvelimille. Välimuistipalvelimia otettiin useampi käyttöön, jotta yhden välimuistipalvelimen sisältämä tietomäärä ei olisi ollut liian suuri. Samaten käytettiin myös useampia paikantimia, jotta yhden paikantimen käsittelemä pyyntömäärä ei olisi ollut niin suuri. Suuri tietomäärä otettiin huomioon jo sovelluskehityksen aikana, jolloin päätettiin hyödyntää GemFiren ositustekniikkaa, minkä ansiosta eri välimuistipalvelimille ositettiin vain pieni osa koko välimuistin hallinnoimasta tietomäärästä. Toinen vaihtoehto olisi ollut kopioida kaikki tiedot kaikille välimuistipalvelimille, mutta tällainen ei ollut paras lähestymistapa suurien tietomäärien tapauksessa. Osituksia voitiin siirrellä myös uudelleen kuorman tasaamiseksi eri välimuistinpalvelinten välillä.

Jokaiselle paikantimille ja välimuistipalvelimelle piti aina käynnistettäessä muistaa asettaa omat porttinumeronsa ja kansionsa, jos ne olivat samalla palvelimella. Tämä vaati myös omien käynnistysskriptien lisäämistä, jotta koko CentOS-palvelimen uudelleenkäynnistys ei olisi vaatinut ylläpitäjän puuttumista GemFiren paikantimien ja välimuistipalvelinten käynnistelyihin.

RabbitMQ:n klusterointia hallittiin komentorivityökalun ja asetustiedostojen avulla. Klusteroinnissa vaatimuksena oli, että kaikilla klusteriin osallistuvilla RabbitMQ-palvelimilla oli sama Erlang-eväste. Klusterin palvelimet olivat tyypiltään muistipalvelimia ja levypalvelimia. Muistipalvelin tallensi tietonsa nimensä mukaisesti muistiin kun taas levypalvelin kirjoitti tiedot levylle. Klusterissa piti olla vähintään yksi levypalvelin, jotta viestijonojen tiedot eivät olisi hävinneet katkosten aikana. Yhden levypalvelimen piti olla aina käynnissä ennen

kuin muut palvelimet pystyttiin uudelleenkäynnistämään. Tällä tavalla varmistettiin tallennettujen tietojen yhtenäisyys. (Videla ym. 2012, 92-94.)

Viestijonopalvelimen jonot ja viestit kopioitiin muille klusterin viestijonopalvelimille erityisillä peilausasetuksilla. Tässä mallissa yksi palvelin oli isäntäpalvelin ja muut palvelijoita, jotka saivat kopionsa viesteistä ja viestijonoista isäntäpalvelimelta. Jos isäntäpalvelin sammui, niin yhdestä palvelijasta tuli uusi isäntäpalvelin. (Videla ym. 2012, 101-105.)

9 ONGELMAT

Sovelluskehityksen aikana ilmeni ongelmia siinä kuinka välimuistissa olevalle tallennetulle oliolle saatiin asetettua tausta-ajona tapahtuvassa tallennuksessa tietokannan taulun pääavainkentässä käytettävä arvo. Ratkaisuna oli pyytää tietokannalta aina ennen tallennusta seuraava vapaa pääavain. Tämä ei kuitenkaan ollut paras ratkaisu, koska tallennus oli tarkoitus suorittaa aina tausta-ajona. Tällöin erilliset kyselyt tietokannan vapaasta pääavaimesta sekä sen asettamisesta tallennetun olion tietoihin hidastivat sovelluksen kirjoitustoimintaa turhaan sekä tekivät siitä pahimmassa tapauksessa lähes synkronisen tietokantatallennuksen kanssa. Toisaalta sovelluksen pääasiallinen tarkoitus oli tarjota tietoja luettaviksi, joten tallennuksia ei ollut niin useita. Käyttäjämäärän kasvaessa tämä olisi kuitenkin hyvä ratkaista jotenkin toisin, koska suuri käyttäjämäärä tarkoittaa suurta määrää tallennettavia tietoja.

Koska Camelin oma AMQP-komponentti käytti oletustoteutuksena Apache Qpidia, niin RabbitMQ-tuen selvittämisessä meni myös oma aikansa. Tämä kuitenkin saatiin ratkaistua aiemmin mainitulla camel-amqp-spring -komponentilla. Ongelmana oli kuitenkin käsittelijän osalta yleisestikin, että monet toteutuksessa käytetyt ohjelmakirjastot eivät ole ottaneet huomioon OSGi-tukea tehdessään riippuvuusmäärityksiä muihin kirjastoihin. Tällöin jouduttiin itse erottelemaan tarpeettomat riippuvuudet käytetyistä kirjastoista sekä korvaamaan joitakin riippuvuuksia OSGia varten räätälöidyillä paketeilla. Tällä me-

nettelyllä saatiin rajattua ServiceMixiin asennettavat komponentit ainoastaan sovelluksen kannalta oikeasti tarpeellisiin komponentteihin.

Käsittelijään haluttiin tuki dynaamisten XSLT-tiedostojen käytölle. Camel tukee XSLT-muunnoksia varhaisista versioista asti, mutta tuki dynaamisille XSLT-tiedostoille tuli mukaan vasta versiossa 2.9.0. ServiceMix taas tuki oletuksena Camelin versiota 2.8.5. Tästä syystä Camel päätettiin päivittää suoraan versioon 2.10.0, koska jo Camelin päivittäminen versioon 2.9.0 olisi vaatinut Karafin päivittämistä samaan versioon kuin mitä versio 2.10.0 vaati. Karaf-komponentin päivitykseen ei kuitenkaan ollut mitään selkeää menetelmää, ja päivitys olisi vaatinut useiden eri riippuvuuksien selvittämistä ja korvaamista toisilla riippuvuuksilla. Riippuvuuksien sisäisissä riippuvuuksissakin oli omat ongelmansa, joten lopulta Karafin uuden version asennus ajettiin ServiceMixin avulla pakolla loppuun asti, vaikka kaikkia kirjastoja ei saatukaan enää käynnistettyä uudelleen. ServiceMix ei normaalisti aja asennusta loppuun, jos se ei saa kaikkia asennuksen tarvitsemia kirjastoja käynnistettyä. Asennus on kuitenkin mahdollistaa ajaa loppuun käyttäjän niin halutessa.

Pakotettu Karaf-komponentin päivitys teki ServiceMixistä melko epävakaan alustan, varsinkin asennettaessa uusia ServiceMixin komponentteja, mutta sovelluskehityksen ja testauksen aikana se toimi riittävän hyvin eikä toteutetun käsittelijän tarvitsemissa komponenteissa huomattu mitään toimintaongelmia. ServiceMix piti tosin välillä asentaa uudelleen alusta asti, koska uuden komponentin asennuksen jälkeen ServiceMix ei välttämättä tunnistanut enää aiemmin asennettua ja toiminutta toista komponenttia.

10 TULOKSET JA HAVAINNOT

Skaalautuvuus saavutettiin parhaiten jakamalla asiat useisiin osiin. Tämä sääntö koski niin palvelimia kuin sovelluksen tehtäviäkin. Mitä pienempiin yksiköihin asioita saatiin jaettua sitä helpompi niitä oli tarvittaessa skaalata käyttötarpeen mukaisesti. Muistinvaraisen kannan käyttämisellä saatiin selkeästi ly-

hyemmät vasteajat tietoja luettaessa välimuistista verrattuna tietojen lukemiseen joka kerta tietokannasta asti.

Sovelluksesta olisi vielä voitu erottaa XSLT-tiedostoihin liittyvä hallinta omaksi kokonaisuudekseen, koska tietokantaan tallennetut dokumentit eivät ole pitkällä aikavälillä paras ratkaisu kun tallennettujen tiedostojen määrä kasvaa. Tulevaisuutta ajatellen dokumenttienhallinta voitaisiin siirtää tällaiseen tarkoitetun valmiin ratkaisun vastuulle. Dokumenttienhallintaan on useita ilmaisia ratkaisuja, joista mainittakoon esimerkkinä Alfresco Community Edition.

Jatkokehityksenä tulisi kehittää jonkinlainen klusterointiratkaisu ServiceMixille, koska yksi ServiceMix muodostaa todellisen pullonkaulan järjestelmään. Yksi toimiva vaihtoehto olisi voinut olla vain laittaa ServiceMixejä useille eri palvelimille vastaanottamaan viestejä samalta viestijonolta, mutta tällöin pitäisi myös varmistua jotenkin siitä, että samoja viestejä ei käsitellä useampaan kertaan. Viestien kuittaaminen voisi auttaa tässä jonkin verran, mutta yleensä kuittaus lähetetään vasta kun viesti on jo käsitelty. Käsittelyaika taas riippuu muunnoksen koosta.

Klusterointi oli yleisestikin toimiva ratkaisu moniasiakasympäristön vaatimassa korkean saatavuuden takaamisessa, ja se oli tuettu lähes kaikissa käytetyissä komponenteissa. Toisaalta klusterointi myös lisäsi ylläpitotyötä, koska hallittavia komponentteja oli moninkertainen määrä. Hallintaa voitiin kuitenkin helpottaa hiukan Hyperic-valvontapalvelimen käytöllä, jolla oli mahdollista myös hallita useimpia komponentteja. Ratkaisematta jäi kuitenkin miten kaikille sovelluspalvelinklusterin palvelimille olisi saatu päivitettyä uusi versio hallintasovelluksesta ilman käyttökatkoja. Sovelluspalvelinten yleisenä ongelmana on ollut Java-maailmassa, että jossain vaiheessa sovelluksia päivitettäessä ilman sovelluspalvelimen uudelleenkäynnistystä muisti loppuu ja palvelin on pakko sammuttaa ja käynnistää uudelleen. Palvelinklusterissa tämä ei olisi varsinaisesti ongelma, koska käyttäjien istunnot ohjattaisiin muille palvelimille, mutta jossain vaiheessa kaikki istunnot saattaisivat päätyä yhdelle ja samalle sovelluspalvelimelle, jossa on sovelluksen vanha versio. Tällöin palvelimen kuorma kasvaa liikaa aiheuttaen mahdollisia käyttökatkoja ja viivettä. Tämä olisi tieten-

kin riippuvainen käyttäjäistuntojen kestosta, joka on suoraan verrannollinen siihen kauanko sovelluksen vanhan version pitää olla saatavilla.

vFabric-alustan tarjoamat skaalautuvuusratkaisut olivat tehokkaita ja hyvin dokumentoituja. Toisaalta alustan käyttäminen myös sitoi käytetyt ratkaisut yhteen valmistajaan. Kaikille käytetyille komponenteille on kuitenkin olemassa muitakin vastineita aina maksullisista avoimen lähdekoodin ratkaisuihin, joten mahdollinen siirtyminen toisen tai useamman valmistajan tarjoamiin ratkaisuihin pitäisi olla mahdollista.

Yleisesti ottaen horisontaaliseen skaalaukseen on useita maksullisia sekä avoimen lähdekoodin ratkaisuja. Vaikka käytetyt teknologiat ovatkin melko nuoria vielä, niin ne kehittyvät koko ajan, ja useat isot yritykset VMwaren tapaan tarjoavat omia ratkaisujaan sekä asiantuntemustaan asian suhteen. Skaalautuvuus tulee kuitenkin huomioida alusta asti, koska jälkikäteen sen lisääminen voi olla haastavaa ellei peräti mahdotonta joissakin tapauksissa.

Sovelluskehityksen ja testauksen aikana saatiin paljon tietoa vFabric-alustan tarjoamista mahdollisuuksista sekä sen tuotteiden käytöstä. Jatkossa osataan myös ottaa paremmin huomioon erilaiset skaalaukseen liittyvät seikat ohjelmistokehityksen alusta asti suunniteltaessa ja toteutettaessa uusia järjestelmiä. Havainnot avoimen lähdekoodin työkalujen tarjoamista skaalausratkaisuista ovat myös tärkeitä.

LÄHTEET

Abbot, M. & Fisher, M. 2011. Scalability Rules. Crawfordsville: R.R. Donnelley.

Application Cluster Monitoring with Spring Insight Operations. n.d. vFabric Documentation Center. Viitattu 1.4.2012.
Http://pubs.vmware.com/vfabric5/topic/com.vmware.vfabric.tc-server.2.6/getting-started/intro-insight-enterprise.html.

Camel and RabbitMQ. n.d. Arthur Gonigberg. Viitattu 8.9.2012.
Http://arthur.gonigberg.com/2012/02/11/camel-rabbitmq/.

Cheung, D. 2008. What is First and Second Level caching in Hibernate?. Stack Overflow 3.12.2008. Viitattu 16.10.2012.
Http://stackoverflow.com/questions/337072/what-is-first-and-second-level-caching-in-hibernate.

Dynamic Recipient List. n.d. Apache Camel: Recipient List. Viitattu 9.9.2012.
Http://camel.apache.org/recipient-list.html.

Dynamic stylesheets. n.d. Apache Camel: XSLT. Viitattu 9.9.2012.
Http://camel.apache.org/xslt.html.

Fisher, M. 2012. When will you supper AMQP 1.0? Spring Community Forums 20.1.2012. Viitattu 7.11.2012.
Http://forum.springsource.org/showthread.php?121724-When-will-you-support-AMQP-1-0.

Getting started with RabbitMQ. n.d. SpringSource. Viitattu 20.2.2012.
Http://www.rabbitmq.com/getstarted.html.

Gierke, O., Leau, C. & Turanski, D. n.d. Spring Data Gemfire Reference Guide. Viitattu 15.9.2012.
Http://static.springsource.org/spring-data/gemfire/docs/1.2.0.RC1/reference/html/index.html.

Harrison, G. 2010. 10 things you should know about NoSQL databases. TechRepublic 26.8.2010. Viitattu 15.2.2012.
Http://www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772.

How Member Discovery Works. n.d. Vfabric Documentation Center. Viitattu 15.9.2012.
Http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.gemfire.6.6/topologies_and_comm/topology_concepts/how_member_discovery_works.html.

Introduction to Hyperic Monitoring. n.d. vFabric Documentation Center. Viitattu 1.9.2012. Http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.hyperic.4.6/Introduction_to_Hyperic_Monitoring.html.

Knego, P. 2010. Differences between GTW and Vaadin. Stack Overflow 11.11.2010. Viitattu 3.11.2012. Http://stackoverflow.com/questions/4155783/differences-between-gwt-and-vaadin.

Lenz, E. n.d. How XSLT Works. Lenz Consulting Group, Inc. Viitattu 28.10.2012. Http://lenzconsulting.com/how-xslt-works.

Main Features of vFabric GemFire. n.d. vFabric Documentation Center. Viitattu 1.9.2012. Http://pubs.vmware.com/vfabric5/topic/com.vmware.vfabric.gemfire.6.6/getting_started/product_intro.html.

Manage tc Server Applications. n.d. vFabric Documentation Center. Viitattu 1.9.2012. Http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.hyperic.4.6/tc_Server.html?path=7_6_14_1#79037464_tcServer-ManagetcServerApplications.

Raehalme, T. 2012. Opinnäytetyö 6.4.2012. Sähköpostiviesti. Vastaanottaja M. Virmalainen.

Rebalancing Partitioned Region Data. n.d. vFabric Documentation Center. Viitattu 15.9.2012. Http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.gemfire.6.6/developing/partitioned_regions/rebalancing_pr_data.html.

Region Types. n.d. vFabric Documentation Center. Viitattu 15.9.2012. Http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.gemfire.6.6/developing/region_options/region_types.html.

Spring Framework. n.d. SpringSource.org. Viitattu 1.4.2012. Http://www.springsource.org/spring-framework#documentation.

Storage and Distribution Options. n.d. vFabric Documentation Center. Viitattu 15.9.2012. Http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.gemfire.6.6/developing/region_options/storage_distribution_options.html.

Terrill, G. 2007. Think you know what scalability is? InfoQ 1.10.2007. Viitattu 1.9.2012. Http://www.infoq.com/news/2007/10/whatisscalability.

The OSGi Architecture. n.d. OSGi Alliance. Viitattu 15.10.2012. Http://www.osgi.org/Technology/WhatIsOSGi.

Topology Types. n.d. vFabric Documentation Center. Viitattu 15.9.2012. Http://pubs.vmware.com/vfabric51/topic/com.vmware.vfabric.gemfire.6.6/topologies_and_comm/topology_concepts/topology_types.html.

Usability Enhancements to Apache Tomcat. n.d. vFabric Documentation Center. Viitattu 16.3.2012. Http://pubs.vmware.com/vfabric5/topic/com.vmware.vfabric.tc-server.2.6/getting-started/intro.html?path=2_0_1_0#intro-tomcat-enhancements.

Vaadin. n.d. Vaadin Ltd. Viitattu 1.4.2012. Https://vaadin.com/learn.

Videla, A. & Williams, J. 2012. RabbitMQ in Action. Shelter Island: Manning.

Virtualization Use Case: Application Scaling using Virtualization. 2008. Intel Software Network. Viitattu 1.4.2012. Http://software.intel.com/en-us/articles/virtualization-use-case-application-scaling-using-virtualization/.

VMware vFabric tc Server. n.d. VMware. Viitattu 16.3.2012. Http://www.vmware.com/products/application-platform/vfabric-tcserver/overview.html.

VMware RabbitMQ: Open Source Enterprise Messaging Middleware Optimized for the Cloud Using AMQP. n.d. VMware. Viitattu 16.3.2012. Http://www.vmware.com/products/application-platform/vfabric-rabbitmq/overview.html.

XSL-FO Introduction. n.d. W3Schools Online Web Tutorials. Viitattu 7.11.2012. Http://www.w3schools.com/xslfo/xslfo_intro.asp.

Yritys. 2011. Codecenter Oy:n verkkosivut. Viitattu 31.3.2012. Http://www.codecenter.fi/yritys/.