Opinnäytetyö (AMK / YAMK)

Tietojenkäsittely

Tietojärjestelmät

2016

Jani Ullakonoja

RAJAPINTATOTEUTUS SERVICEMIX-INTEGRAATIOALUSTAA KÄYTTÄEN

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK / YAMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely |Tietojärjestelmät

2016 | 32

Anne Jumppanen

Jani Ullakonoja

RAJAPINTATOTEUTUS SERVICEMIX-INTEGRAATIOALUSTAA KÄYTTÄEN

Työn tavoitteena on antaa yleiskuva integraatiosta ja integraatioalustasta integraatiotyökaluna ja peilata tehtyä toimeksiannon mukaista integraatio- ja rajapintaratkaisua kerättyyn tietoon, arvioiden minkälainen integraatioratkaisu toteutus on.

Työssä käytetään tapaustutkimusta hyödyntäen tehtyä rajapintaratkaisua ja sen implementointia integrointialustaan heijastaen sitä tutkimukseen. Tutkimuksessa käytetään sekä kirjallisia lähteitä että oman kokemuksen kautta saatuja tuloksia.

Integraatioratkaisut ovat monimutkainen kokonaisuus ja ne ovat harvemmin keskenään verrattavia. Jokainen integraatioprojekti on erilainen riippuen integroitavista järjestelmistä ja integraation laajuudesta. Erilaisia integraatiotapoja on monta ja niillä saadaan aikaan hyvin paljon erilaisia toteutuksia. Toimeksianto toteutettiin integraatioalustaa käyttäen ja se pyrittiin rakentamaan mahdollisimman skaalautuvaksi. Integraatioalusta tarjoaa hyvän alun ratkaisuun, mutta valitun ServiceMix-integraatioalustan yksinkertaisuuden takia siihen pitää lisätä ominaisuuksia, jotta se on asiakkaan toiveiden mukainen.

Integraatiolla saadaan aikaiseksi hyvin paljon, mutta toimivan kokonaisuuden rakentaminen vaatii paljon työtä ja aikaa. Integraatioalustan asettaminen muotoon, jossa sinne on helppo implementoida uusia ohjelmia, vaatii hyvän pohjatyön ja ohjeistuksen, jotta oikean integraatiomallin mukaista toimintaa seurataan.

ASIASANAT:

Integraatio, tietojärjestelmä, tietokanta, rajapinta

BACHELOR´S / MASTER'S THESIS THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

ICT services and solutions | Information Systems

2016| 32

Anne Jumppanen

Jani Ullakonoja

INTERFACE SOLUTION USING SERVICEMIX INTEGRATION PLATFORM

This thesis was carried out as part of a project. The objective of this thesis was to produce an integration solution for the client that was contracted by ATR Soft. Another objective of the thesis was to produce extra value as ATR Soft had not worked with the ServiceMix integration platform before and thus there was no prior knowledge as to how ServiceMix could and would perform.

The purpose of this thesis was to produce a good overall understanding of integration solutions and how they were performed and how integration platforms and ServiceMix specifically could be used as part of such projects. This was then used to access how good the produced solution was and how it could be improved in the future.

The study was conducted by acquiring information about how integrations are usually implemented and what kind of procedures are used when implementing integration solutions. Because there are multiple different types of integrations solutions depending on the size and type of the systems and programs integrated, the scope of this thesis contains solutions that were practical when compared to the produced solution.

The integration platform was implemented to meet the current need, but might need further development if clustering is needed in the future. The full scope of the features offered by ServiceMix was not used and because of this, upgrading the produced interfaces inside of it is not as easy as it could be. The interfaces themselves are in a questionable state as it has not been decided how they will be developed in the future.

The implementation of an integration platform should make it easier to implement new interfaces or programs within it in the future. Current problems are that the message routes are barely used and the interfaces work individually instead of functioning as parts of a solution. As a solution in the current scope, the implementation is sufficient but if more systems are integrated in the architecture of how the programs are communicating and working now should be remodeled to use more of the functionality provided by the integration platform.

KEYWORDS:

Integration, information system, database, interface

SISÄLTÖ

LIITTEET

KAAVAT

KUVAT

KUVIOT

TAULUKOT

KÄYTETYT LYHENTEET TAI SANASTO

Lyhenne	Lyhenteen selitys (Lähdeviite)
CRM	Customer Relationship Management, Asiakkuudenhallinta
ESB	Enterprise Service Bus
ERP	Enterprise Resource Planning, Resurssinhallinta
MDS	Master Data Service. Palvelu ydintiedon ylläpitämiseksi
OSGi	Java-pohjainen modulaarinen dynaaminen säiliö (https://www.osgi.org/developer/architecture/)
Proof of concept	Proof of conceptilla tarkoitetaan yksinkertaista demoa, jonka ainoana tarkoituksena on näyttää ratkaisun toteutettavuus

1 JOHDANTO

Tässä opinnäytetyössä pyritään antamaan yleinen kuva integraatiosta, integraatioalustoista työkaluna integraation toteutukseen sekä tarkastelemaan näitä tehdyn rajapintaratkaisun ympäristössä. Integraatiot ovat nykypäivänä laajeneva ilmiö vanhojen järjestelmien jäädessä kehityksessä jälkeen versioissa sekä ympäröivissä järjestelmissä ja valmisratkaisujen tarjoamat mahdollisuudet eivät täytä enää tarvittavia viitekehyksiä. Integraatioilla on tarkoitus helpottaa ja yksinkertaistaa yrityksen järjestelmien välistä toimintaa ja ohjelmien hallintaa.

Integraatioratkaisuja on mahdoton yleistää, sillä integraatioratkaisut ovat tarveriippuvaisia. Integraatioiden luonteen takia niitä ei pysty hankkimaan valmiina ja useimmiten integraatiot kehittyvät laajoiksikin projekteiksi joissa aikaa kuluu toteutuksen lisäksi runsaasti myös selvitys- ja suunnittelutyöhön. Näiden syiden takia integraatio on laaja käsite, jonka alle voidaan laittaa sekä kahden ohjelman toiminnollisuuksien integrointi yhteen ohjelmaan, tai kokonaisen yrityksen tietojärjestelmien integrointi toimivaksi kokonaisuudeksi, sisältäen ohjelmat, toimintalogiikan, datan ja alustat. Tässä työssä pyritään antamaan yleinen kuva integraation toteutuksesta ja tarkentaa sitten toimeksiannossa tehtyihin ratkaisuihin antaen kontekstia teorialle.

Työn tavoitteena oli toteuttaa rajapintaratkaisu ja asentaa se integraatioalustalle. Ratkaisun ollessa näin yksinkertaisemmillaan on tarkoitus saada integraatioalusta asennettua siten että siihen on mahdollista tuoda tulevaisuudessa muita ratkaisuja mahdollisimman yksinkertaisesti. Tämän kautta pyritään saamaan kuva siitä miten tehty rajapintaratkaisu voisi toimia osana suurempaa integraatiota ja miten suuremman integraatioratkaisun tulisi toimia, jotta se toimisi suuremman ohjelmakokonaisuuden kanssa.

ServiceMixin ja sen komponenttien tutkimustyö on hyödyllistä toimeksiantajalle ATR softille, koska uuden tiedon tuominen yritykseen mahdollistaa kyseisen ratkaisun käytön muissakin projekteissa. Varsinkin räätälöityjä ratkaisuja tekevä yrityksellä on tarvetta saada mahdollisimman laaja käsitys monesta eri teknologiasta. ServiceMixin kaltainen muokattava perusratkaisu voi toimia pohjana monessa erilaisessa ratkaisussa.

Työssä otetaan myös kantaa tiedon ja ydintiedon integroimiseen, sillä se oli osana toimeksiannon projektia ja yhtenäistettyä tietoa käytettiin koko tietokantaratkaisussa. Tiedon kerääminen ydintiedoksi ja sen käyttäminen koko integraatioratkaisun läpi on yksi

merkittävimmistä haasteista suurien integraatioiden yhteydessä tietoon liittyvän luotettavuuden ja tämän luotettavuuden haavoittumisen takia.

2 TOIMEKSIANTO JA TUTKIMUSMENETELMÄT

Toimeksiantajana tälle työlle toimi ATR soft, joka on valmisratkaisuja ja projekteja tarjoava ohjelmointitalo. ATR soft on perustettu vuonna 2000 ja sen tärkeimpänä arvona on luoda mahdollisimman sopiva ratkaisu asiakkaan tarpeisiin. Tämän takia ATR soft tekee useita erilaisia ratkaisuja ja tarjoaa paljon erilaista osaamista. Tehty toimeksianto oli ATR Softilla yksi ensimmäisistä kosketuksista ServiceMix–alustan kanssa, joten teoreettisen tiedon haku opinnäytetyöhön toi siihen lisäarvoa tarjoten näkemyksen siihen, mitä alustalla on mahdollista tehdä.

Projektin tavoitteena oli rakentaa rajapintaratkaisu integraatioalustan sisään ja ottaa integraatioalusta käyttöön rajapintojen kanssa. Normaalisti näin pieneen toteutukseen ei kannata integraatioratkaisua harkita. Koska asiakkaalla oli tarve saada integraatioalusta, jonne tuoda muita käytössä olevia ohjelmia, oli tässä tapauksessa kannattavaa lähteä rakentamaan ratkaisua integraatioalustan päälle ja tehdä integraatioalustasta yleinen ja helposti skaalautuva.

Rajapintojen määritykset tulivat valmiista jo olemassa olevista rajapinnoista. Rajapintojen tuli peilata näitä malleja ja toteuttaa niitä siten että palautuva sanoma ja annetut parametrit olisivat samoja. Rajapintojen suunnittelusta suurin osa meni selvitystyöhön ja sovittamiseen.

Käytetty tietomalli tehtiin ydintiedon ylläpitämiseksi ja sen tarkoituksena on tulla uudeksi asiakkaan ydintiedon säilytyspaikaksi. Tietomalliin on tarkoitus siirtää kaikki ydintieto asiakkaan tämänhetkisistä tietokannoista ja tämän jälkeen siitä on tarkoitus tulla uusi ydintiedon pääversio.

Vaikka toimeksiannossa käytetäänkin valmista integraatioalustaa, on oletettavaa, että se ei täytä kaikkia asiakkaan toiveita. On tärkeää kartoittaa puutteet ja tämän jälkeen tarkastella minkälaisia ratkaisuja näihin puutteisiin löytyy, kun niiden implementointi ja testaus ovat vielä yksinkertaisemmillaan.

Ratkaisusta pyritään tekemään paras mahdollinen, jolloin on hyvä ensin kartoittaa mahdollisuudet ja tämän jälkeen alkaa verrata niitä haluttuihin ja toteutuskelpoisiin ratkaisuihin. Tällä tavalla saadaan ainakin teoreettinen kehys siitä, mitä tällä hetkellä on mahdollista ja järkevää tehdä ja mitä mahdollisia jatkokehityskohteita tulevaisuudessa on.

Tapaustutkimuksena tehdyssä työssä toimeksiantona kehitettyä ratkaisua käytetään esimerkkinä teoreettisen tiedon vierellä ja tuodaan konkreetista näkökulmaa ratkaisun tekemiseen. Tällä tavoin saadaan kuva integraatioratkaisun käytännöllisestä toteutuksesta liikemaailmassa. Teoreettisella tasolla integraatioratkaisun koko sekä toiminnollisuus ovat rajattomat, mutta käytännöllisessä mielessä on mahdoton tehdä kaikkia tarvittavia muutoksia integraatioratkaisun sisäisiin osiin, jotta koko järjestelmä saadaan toimimaan tarvittavalla tasolla.

Työssä käytettyjen tietolähteiden on oltava luotettavia, jonka takia tuotteiden omilta sivuilta saatuun tietoon ei voida suoraan luottaa. ServiceMixiin ja muihin teknologioihin kohdistuvassa tiedonkeruussa luotettiin testauksessa todettuihin tuloksiin. Proof-of-concept toteutuksella voidaan kotisivujen subjektiivinen tieto todeta oikeaksi ja todistaa että teknologialla voidaan toteuttaa tarpeellinen toiminnollisuus.

Toimeksiannossa oli hyvin paljon selvitystyötä, minkä takia opinnäytetyö on tehty teoria ensin -periaatteella. Suurin osa teoriasta on haettu ennen empiirisen osuuden tekemistä. Tästä johtuen empiiristä osuutta lähinnä verrataan teoriaan ja työn paino pysyy mahdollisuuksien pohdinnassa.

3 INTEGRAATIO

3.1 Integraation tarve

Integraatioille on kasvava tarve nykyisessä teknologisesti kehittyvässä maailmassa – varsinkin nykypäivänä, kun vanhat järjestelmät ovat siirtymässä tilaan, jossa niiden päivittäminen ei enää ole kannattavaa ja niiden kehitys lakkaa, mutta osaa niiden toiminnollisuudesta halutaan silti säilyttää (Hohpe & Woolf 2004, 3). Vanhojen järjestelmien toiminnollisuuden kopioiminen uusin järjestelmiin voi olla hyvin kallista ja pahimmillaan vanhaa järjestelmää ei voida hyödyntää ollenkaan teknologioiden päivitettyjen versioiden eroavaisuuksien takia.

Toinen suuri integraation tarpeen luoja on järjestelmäratkaisujen koon jatkuva kasvaminen. Nykyään suurimmilla yrityksillä ja julkisilla tahoilla voi olla kerääntynyt vuosien saatossa useita eri järjestelmiä (Hohpe & Woolf 2004, 1). Yritykset ja julkiset tahot haluavat kuitenkin tehdä säästöjä ja jatkuva uusien järjestelmien implementointi ja kehitys ovat olla kalliita. Myös järjestelmien päällekkäisyydet voivat käydä kalliiksi, sillä tahot joutuvat maksamaan useasti samasta toiminnollisuudesta. Monesti uusia järjestelmiä on hankittu, jotta saataisiin nykyisistä vaihtoehdoista puuttuvia toiminnollisuuksia. Usein tämän kaltaiset ominaisuudet kuitenkin haluttaisiin mahdollisimman vähän järjestelmän sisälle, jotta käyttäjien koulutus ja mahdollinen jatkokehitys olisi helpompi järjestää.

Ennen oli helpompaa rakentaa ja luoda uusia ratkaisuja, jotka täyttivät monen eri järjestelmän tehtävät. Näitä olivat esimerkiksi ERP- ja CRM-järjestelmät, mutta ajan kuluessa yrityksille on kertynyt useita erilaisia ja erikoistuneimpia järjestelmiä, joita ei valmisratkaisuista löydy (Hohpe & Woolf 2004, 2).

3.2 Integraation rakentaminen

Integraation rakentaminen lähtee liikkeelle selvitys- ja suunnittelutyöstä. Selvitystyöllä pyritään kartoittamaan sekä nykyisen että halutun järjestelmän tila. Mitä vanhempiin ja suurempiin järjestelmiin mennään, sitä enemmän selvitystyötä on tehtävä. Nykyistä ja haluttua ratkaisua verrataan ja tämän perusteella rakennetaan malli, jonka perusteella

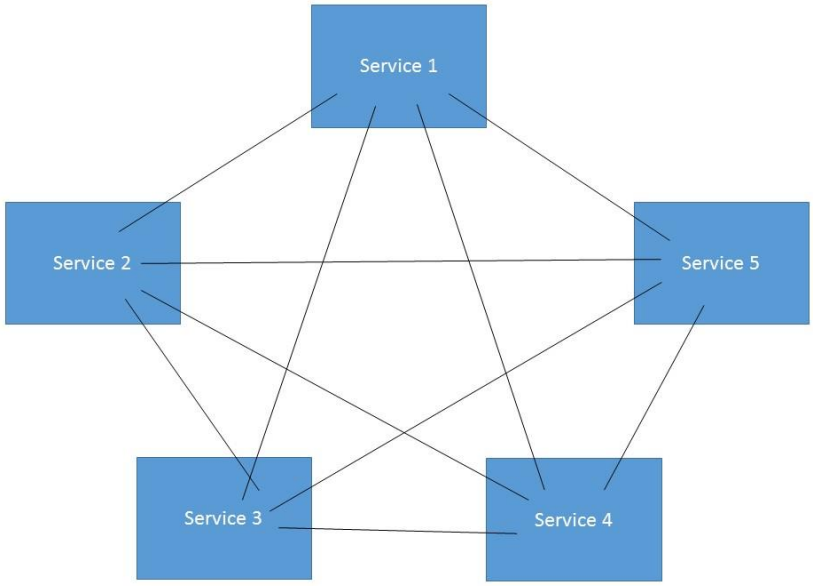
voidaan päätellä, miten erilaisten integroitavien osien tulee toimia keskenään, minkälaisia päätepisteitä pitää rakentaa ja mitä uutta pitää kehittää, jotta haluttu lopputulos saadaan aikaiseksi.

Tehdyn projektin tapauksessa vanhoja järjestelmiä ei ollut laisinkaan, jolloin olemassa olevan selvitystyö jäi kokonaan ydintiedon puolelle. Asennettuun integraatioalustaan on kuitenkin tarkoitus tuoda useita järjestelmiä tulevaisuudessa, minkä takia on tärkeää, että tässä vaiheessa rakennetaan mahdollisimman helposti laajennettava ratkaisu.

3.3 Point-to-Point

Point-to-point-menetelmä, myös tunnetaan app-to-app-menetelmänä, on integraatiomenetelmä, jossa integraatio tehdään luomalla sidos kahden osan välille, minkä kautta ne pystyvät jakamaan tietoa (Fenner 2003). Pienissä integraatioissa kyseinen menetelmä voi olla sopiva, mutta suuremmissa integraatioissa ohjelmien välisiä siteitä syntyy monia ja monessa tapauksessa kahden ohjelman sidonnaisuus on melko kestämätön muutoksille. Kuten kuvassa 1 olen esittänyt, viiden palvelun järjestelmässä sidoksia on jo kymmenen ja jokainen lisätty palvelu tekee sitä mukaa uusia sidoksia mitä enemmän sen pitää keskustella muun järjestelmän kanssa.

Point-to-point menetelmää käytettäessä työtaakka pohjautuu suoraan yhdistettävien osien määrään. Tämän takia sen suurimpia riskejä ovat liian monen pienen ohjelman integroiminen, jonka tuloksena sidoksia syntyy useita ja integraation kulut kasvavat suureksi (Carnegie Mellon University 2012). Jokainen sidos on tässä ratkaisussa tehty erikseen, jolloin yhden ohjelman muuttaminen voi aiheuttaa testaustaakkaa, joka ulottuu moneen eri ohjelmaan.



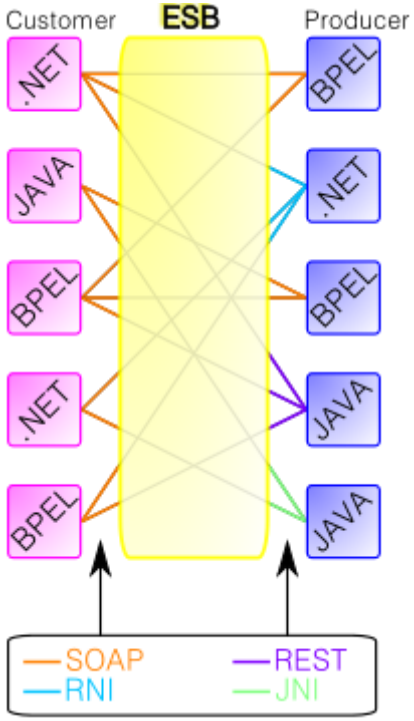
Kuva 1. Point-to-point-toimintamalli

3.4 Middleware toteutus

Middlewarella tehdyllä integraatiolla tarkoitetaan integraatiomallia, jonka osien väliin on rakennettu palvelu, jonka tehtävänä on reitittää ja kääntää kaikki integraatiossa kulkevat viestit (Fenner, 2003). Tämä tarkoittaa, että middlewaren on pystyttävä vastaanottamaan monta eri sanomaa ja tämän jälkeen kääntämään sanomat eri muotoon, jonka jälkeen se lähettää ne jälleen eteenpäin. Riippuen integraation koosta ja monimutkaisuudesta middleware voi olla hyvinkin vaikea rakentaa (Hohpe & Woolf 2004, 4).

Middlewarella tehdyssä integraatiossa tärkein ominaisuus on, että jokainen kutsu ohjelmasta tai palvelusta toiseen tehdään middlewaren kautta. Tällä tavalla varmistutaan siitä, että mahdollinen kirjanpito toimii ja että järjestelmän sisällä olevat osat pystyvät varmasti keskustelemaan keskenään. Middlewaren ympärille rakentuvaa ratkaisua kutsutaan horisontaaliksi, sillä siihen on helppo listä uusi ohjelma mihin vain päähän ja tämän takia se on hyvinkin skaalautuva. Uusien integraation tulevien palveluiden ja järjestelmien on keskusteltava vain middlewaren kanssa, joten niitä on helppo tuoda lisää nopeasti ja joustavasti.

Kuvan 2 Enterprise Service Bus eli ESB on yksi esimerkki middlewaresta. Se tarjoaa yksinkertaiset rajapinnat käyttäjälle ja järjestelmälle, joiden kautta integraation sisäinen sanomaliikenne kulkee (Chappell 2004, 1-3). Tällä tavoin sanomaliikenne pysyy yhtenäisenä ja järjestelmän toiminta saadaan varmemmaksi.



Kuva 2. Enterprise Service Bus -toimintamalli (Enterprise Service Bus 2015)

3.5 Viestireititin

Viestireititin on yksi ESB:n toteutusmalleista. Viestireitittimen tehtävänä on toteuttaa integraation läpi reittejä, joita pitkin kulkevat viestit johtavat jonkin toimintalogiikan toteutumiseen (Hohpe & Woolf 2004, 78). Koska itse toimintalogiikka otetaan ulos ohjelmista ja sen toteutus tehdään viestireitittimessä, johon pitää määritellä ainoastaan reitti ja viestien muoto, saadaan helposti muokattavissa oleva järjestelmä; integraation muuttuessa reitit voidaan siis nopeasti päivittää.

Viestireitittimet sisältävät myös mahdollisuuden muuttaa viestien sisältöä (Hohpe & Woolf 2004, 78). Tämän takia ne ovat hyvä työkalu sellaisten ohjelmien välille, jotka eivät jaa samoja viestimuotoja tai objekteja.

3.6 Tiedon integraatio

Ohjelmien lisäksi myös tieto tulee integroida. Kuten ohjelmienkin kanssa usean eri tietolähteen ylläpito on hankalaa ja voi olla ongelma, kun tuodaan uusia ohjelmia integraatioratkaisuun. Tämän takia myös data pyritään integroimaan siten että sitä pääsee käyttämään yhdestä paikasta.

Transaktiodataa pystytään siirtämään yleensä melko vapaasti suoraan data migraationa. Tämä tarkoittaa, että data pitää siirtää vanhasta tietomallista uuteen ja muokata se siten että se sopii uuteen tietomuotoon. Tämä tuo mukaan omia ongelmiaan, kuten esimerkiksi päivämäärien väärä muoto tai mahdolliset vanhassa tietokannassa olevat herätin, joille tehdään bisneslogiikkaa. (Javlin, 2015.)

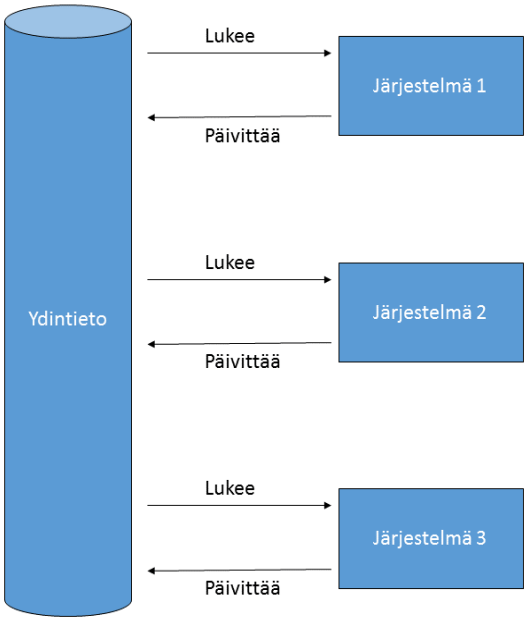
Suuremmissa integraatioissa voi tulla vastaan tilanne, jossa ydintietointegraatiota joudutaan tekemään. Ydintiedon integraatio eroaa transaktiotiedon integraatiosta merkittävästi ydintiedon luonteen takia. Ydintietoa ei saa missään tapauksessa olla useassa paikassa, jotta tiedon puhtaus ja luotettavuus säilyisivät. Duplikaattitieto voi johtaa valheellisten tietojen käyttöön järjestelmässä. (Wolter & Haselden, 2006.) Yleensä tämänlainen tilanne tulee vastaan kahden järjestelmän yhdistyessä.

Kuten ohjelmien integraatiossa, ydintiedon integraatiossa lähdetään liikkeelle selvitystyöstä. Ydintiedon sijainti, rakenne sekä sitä lukevat ja syöttävät tahot tulee tunnistaa ja kartoittaa. Integroitavien järjestelmien ydintiedon tunnistamisen jälkeen suunnitellaan ja rakennetaan tietomalli, jonne ydintieto tullaan syöttämään. Tämä tietomalli tulisi suunnitella ydintiedon muoto-ohjeita seuraten ja pyrkien säilyttämään mahdollisimman paljon sen alkuperäisestä muodosta, jotta järjestelmiä ei tarvitse muuttaa liikaa uuden tietomallin toimimiseksi. (Wolter & Haselenden, 2006.)

Tietomallin toteutuksen jälkeen ydintieto voidaan siirtää sinne järjestelmistä. Siirtämisen jälkeen ydintieto tulee käydä läpi ja varmistaa, että se on puhdasta eikä duplikaattidataa löydy. Duplikaattidatan löytyessä pitää päättää, mikä tiedoista säilytetään. (Wolter & Haselden, 2006.)

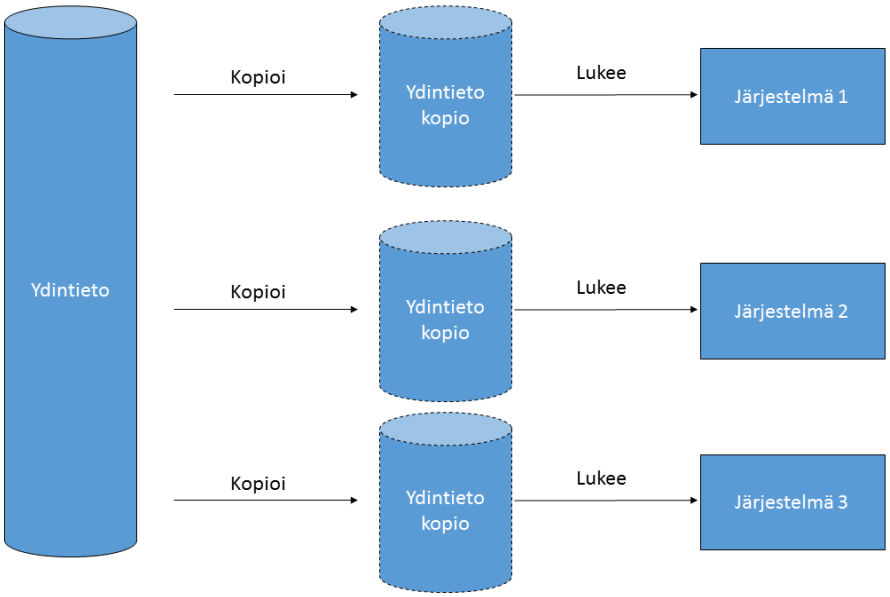
Ydintiedon siirron jälkeen järjestelmät on siirrettävä käyttämään uutta tietomallia. Tietomallia voidaan käyttää usealla eri tavalla. Kuten kuvassa 3 olen esittänyt, yksi vaihtoehto

on pitää se yhtenäisenä ratkaisuna, jota muut järjestelmät käyttävät. Tällöin on varmistuttava siitä, että järjestelmät ovat yhteensopivia uuden tietomallin kanssa, mikä suuremmissa tapauksissa voi olla hyvin raskas prosessi. (Wolter & Haselden, 2006.)



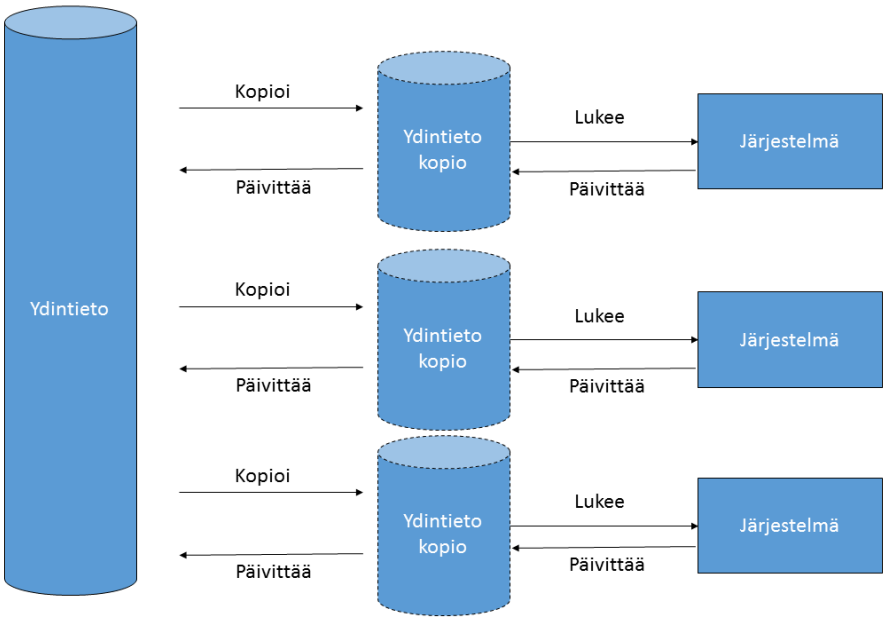
Kuva 3. Järjestelmillä suora yhteys ydintietoon

Vaihtoehtoisesti tietomallista voidaan tehdä kopio kuvan 4 tekemäni mallin mukaan, joka lähetetään järjestelmälle, jotta järjestelmä voi varastoida sen. Tällöin järjestelmä voisi syöttää tiedon vanhaan malliin ja muutoksia ei tarvitse tehdä itse. Tämä kuitenkin tarkoittaa, että käytössä oleva ydintieto ei ole oikea ydintieto, vaan kopio siitä, jolloin sen muokkaaminen ei onnistu, ja järjestelmistä voidaan joutua poistamaan ominaisuuksia, joilla ydintietoa ennen muokattiin. (Wolter & Haselden, 2006.)



Kuva 4. Järjestelmille luodaan kopio ydintiedosta

Kuvassa 5 esitetään kolmatta vaihtoehtoa, joka on tiedon jatkuva yhdistäminen. Malli muistuttaa kopiota siinä mielessä, että järjestelmillä on oma kopionsa ydintiedosta. Siihen tehdyt muutokset viedään kuitenkin oikeaan ydintietomalliin, jossa tieto päivitetään, ja päivitetystä ydintietomallista lähetetään uudet kopiot ohjelmille. Tämän mallin ongelma on järjestelmien tekemien muutosten hallinnointi. Lisäämiset ja muutokset voivat tapahtua samanaikaisesti ja huono hallinnointi voi johtaa muutosten tai lisäysten katoamiseen. (Wolter & Haselden, 2006.)



Kuva 5. Järjestelmille luodaan kopio ydintiedosta, joka voidaan lähettää takaisin

3.7 Hyödyt, haitat ja riskit

Integraation suurin hyöty on sen toteuttama kokonaisuus verrattuna useaan erinäiseen järjestelmään. Integraation tasosta riippuen tämä voi tarkoittaa keskitetympää tietovarastoa, parempaa toimivuutta ohjelmien välillä, vähempää ohjelmien ristikäyttöä tai monen ohjelman yhdistymistä yhdeksi kokonaisuudeksi. Ylläpidolliset toiminnot helpottuvat ja ylläpitokustannukset pienenevät. Työntekijöiden koulutustarve vähenee pienemmän ohjelmataakan takia. (Mike Briercliffe 2016)

Integraatio ei kuitenkaan aina ole halpaa tai hyödyllistä. Monen ohjelman yhdistäminen yhdeksi voi tulla kalliimmaksi kuin yhden uuden, vanhojen järjestelmien ominaisuudet sisältävän ratkaisun ostaminen. Tämän kaltaiset tilanteet syntyvät yleensä pienempiä integraatioita suunniteltaessa. Suuremmissa integraatioissa – kuten suuren yrityksen järjestelmän uusiminen tai kahden suuren yrityksen yhdistyessä tapahtuva yhtenäistäminen – on kuitenkin lähes mahdotonta yrittää rakentaa ratkaisua, joka säilyttäisi kaikki vanhojen järjestelmien toiminnot (Hohpe & Woolf 2004, 4).

Integraatioratkaisun käyttäminen vaatii tiukkaa kuria päivitysten ja lisäysten kanssa. Integraation sisältämillä ohjelmilla voi olla eri ylläpitäjiä ja kehittäjiä, kuten suurten yritysten eri alihankkijat. Tämän takia tiedostojen hallinnoinnin tulee olla kunnossa, jotta esimerkiksi toinen kehittäjä ei ylitä toisen tekemiä päivityksiä.

Integraation toteutuksessa dokumentoinnilla on suuri merkitys. Integraation muoto ja sen sisältämien osien välinen toiminta on oltava helposti saatavilla, koska integraatiota päivittäessä sen nykyinen toiminta täytyy tuntea mahdollisten virheiden välttämiseksi. Dokumentoinnin merkitys kasvaa sitä mukaa, mitä keskeisemmistä osista on kyse: mitä käytetympää osaa ollaan päivittämässä, sitä laajemmat vaikutukset päivityksellä on.

Integraatiot tuovat mukanaan myös uusia ongelmia ylläpitoon. Vaikka ylläpito voidaan teoriassa keskittää ja virheet ovat peräisin vain yhdestä paikasta, laajan integraatioratkaisun monitorointi on haastava tehtävä (Hohpe & Woolf 2004, 4). Integraatioissa virhe voi johtua useammista syistä kuin yksittäisissä ohjelmissa. Toimintalogiikka voi kulkea monen osan kautta, ja on vaikea selvittää, missä näistä virhe on. Myös integraatioon tehdyt korjaukset voivat olla ylläpidollisesti hankalia, koska testaustaakka on suurempi usean liittyvän ohjelman takia. Näistä syistä integraatiota käyttöönottaessa on hyvä varmistaa, että ylläpitohenkilöstö on koulutettu ja heille on annettu kaikki tarvittava tieto mahdollisimman tehokkaaseen valvontaan, jotta vikatilanteet saadaan korjattua mahdollisimman nopeasti.

4 SERVICEMIX

ServiceMixin ymmärtämiseksi on ymmärrettävä integraatioalustojen toimintamalli. Integraatioalustat ovat integraation työkaluja, joiden tarkoituksena on tarjota kehittäjille valmis pohja integraatioon. Integraatioalustat toteuttavat järjestelmäintegraation ohjeita, ja ne muistuttavat rakenteeltaan usein horisontaalia integraatiota, joka on tehty viestikanavien ja kääntäjien muodossa. Monimutkaisemmissa integraatioalustoissa on mukana myös versiohallintaa, mahdollisuus ajonaikaiseen päivitykseen, viestimahdollisuus ja konsoli, josta pystyy näkemään ylläpidolle tärkeitä tietoja, kuten kutsujen määriä tai viestikanavan lähettämien viestien sisältöä.

4.1 Historia

ServiceMix oli alun perin kokonaisprojekti, jossa sen kaikki osat, kuten viestikanava, ydin ja pakettisäiliö, kuuluivat ServiceMix-projektiin. ServiceMixin kehityksen aikana monet sen komponentit ovat kuitenkin siirtyneet omiksi projekteikseen, ja nykyään ServiceMix on ennemminkin säiliö, jonka tarkoituksena on koota integraatioalusta muista osista. Tämän seurauksena ServiceMixin kehitys on jäänyt tasolle, jossa sen sisäisiä osia päivitetään ja ne ohjelmoidaan toimimaan yhdessä. Tämä tarkoittaa sitä, että ServiceMix ei itsessään tule kehittymään, ja se on tämän takia jäänyt integraatioalustana perusratkaisuksi, joka ei tarjoa parempia ominaisuuksia, ellei niitä lisätä siihen erikseen.

4.2 Apache Karaf

Karaf on OSGi-pohjainen kontti, jonka tarkoituksena on hallinnoida ServiceMixin sisällä toimivia osia, eli ohjelmia ja mahdollisia asennettavia palveluita ja lisäosia (The Apache Software Foundation 2016). OSGi tarkoittaa ohjelmien modulaarista asentamista: sen sijaan, että esimerkiksi java-ratkaisu tehtäisiin yhtenä isona pakettina, se voidaan tehdä pienempinä ja täsmällisempinä osina. Suurissa ohjelmistoratkaisuissa modulaarinen ratkaisu mahdollistaa kokonaisuuden helpomman hallinnoinnin ja päivittämisen, sekä monen eri tahon samanaikaisen toiminnan eri osien kanssa. OSGi on dynaaminen, mikä tarkoittaa sitä, että koko järjestelmää ei ole pakko käynnistää uudestaan, jotta joku sen

osista voidaan päivittää. Muutoksia voidaan siis tehdä tosiajassa ilman että itse järjestelmän toimintaan joudutaan häiritsemään. (OSGI alliance 2016)

Karaf sisältää sekä selainpohjaisen konsolin että peruskonsolin, joiden kautta voidaan hallita jo olemassa olevia osia tai asentaa uusia. Karaf tarjoaa hallintatyökaluja, kuten esimerkiksi asennettujen pakettien tai ominaisuuksien listaus, osien tapahtumakirjaus ja uusien ennalta määritettyjen lisäosien asentaminen.

4.3 Apache Camel ja ActiveMQ

ActiveMQ ja Camel toimivat ServiceMixissä viestittäjinä eri ohjelman osien välillä. Camel on reititinpalvelu, jota käytetään luomaan ServiceMixin sisällä logiikkaa. Cameliin voidaan määrittää reitti, joka kulkee saatujen parametrien ja vastausten perusteella, ja lopputuloksena on määritetyn muotoinen vastaus (The Apache Software Foundation 2015). ActiveMQ on viestivälittäjä, jonka tarkoituksena on toimia Cameliin määritetyssä reitissä varmistaen, että reitin ohjelmat pystyvät keskustelemaan keskenään (The Apache Software Foundation 2011). ServiceMix perustuu siihen, että viestitys tapahtuu Camelin välityksellä. Tällöin integraatioalustan sisäinen toiminta on mallinnettua ja ohjelmat toimivat keskenään suuremmalla varmuudella (The Apache Software Foundation 2014).

Camelin ja ActiveMQ:n yhdistelmäratkaisun suuri etu on, että lähetetyn pyynnön ei tarvitse tulla takaisin, jotta viestiketjua voidaan jatkaa. Toimintaa kutsutaan tällöin asynkrooniseksi. Reittiin voidaan määrittää useita kutsuja, ja kun niiden vastauksista on saatu koottua kokonaisuus, se voidaan tarkistaa, ja tämän pohjalta päättää seuraava toiminto. Myös erilaiset kirjaukset voidaan suorittaa ilman että kutsuketju täytyy keskeyttää niiden ajaksi (T Apurva 2014). Asynkrooninen toiminta on nopeampaa kuin synkrooninen ja mahdollistaa sujuvan toimintaketjun ServiceMixin sisällä, mutta tuo mukanaan omat ongelmansa vaikeamman virheiden käsittelyn ja viestiketjun eheyden muodossa.

4.4 Apache CXF

Apache CXF mahdollistaa restful-palveluiden käytön Camelissa. Restful-palvelulla tarkoitetaan palvelua, jota voidaan käyttää suoraan selaimesta. Kutsu palveluun tehdään selaimesta osoitteen kautta ja samalla annetaan mahdolliset argumentit. CXF:n avulla on mahdollista rakentaa rajapinnan kaltaista toimintaa, jossa osoitteilla tehdään Camel-

reitin käynnistävä kutsu. Reitti palauttaa lopuksi vastaussanoman (The Apache Software Foundation 2016 B). CXF:ää käyttäessä restful-kutsua ei tarvitse viedä ulos Camel-reitistä, jolloin ei aiheudu turhaa työtä rajapinnasta, jonka ainoana tehtävänä olisi käynnistää Camel-reittejä restful-kutsun jälkeen.

4.5 Muokattavuus

ServiceMix on perusmuodossaan hyvin yksinkertainen eikä tarjoa perusominaisuuksien lisäksi mitään erityistä. Tämän takia se on kevyt, helppokäyttöinen ja toimii suurimalla osalla alustoista. ServiceMixin yksinkertaisuuden vuoksi muokkaamaton versio siitä on kuitenkin usein riittämätön käyttäjille. Ilman kunnollista visuaalista selainta tai raportointia käyttäjät joutuvat turvautumaan konsolinäkymään ja tapahtumakirjaukseen.

ServiceMixin suurin puute on sen sisältämien hallinnollisten työkalujen puute. Ilman erillisiä ohjelmia ServiceMix ei tarjoa tiedosto- tai versiohallintaa ja soveltuu tämän takia huonosti suurempiin toteutuksiin, joissa on useita tekijöitä.

ServiceMix tarjoaa kuitenkin mahdollisuuden asennuttaa erilaisia kolmannen osapuolen ohjelmia, joilla saadaan aikaan lisää toiminnollisuuksia. Näitä ovat esimerkiksi pakettien parempi selaaminen tai Camel-reitittimen lisäominaisuudet. Kun käyttäjä saa itse määrittää haluamansa ominaisuudet, vältytään turhilta toiminnoilta, mikä varmistaa, että ServiceMix on mahdollisimman kevyt ratkaisu haluttujen toiminnollisuuksien aikaansaamiseksi.

5 MASTER DATA MANAGEMENT

5.1 Microsoft MDS

Microsoft MDS on SQL-serverin osa, jonka tarkoituksena on antaa työkaluja MDM:ään. MDS:n tarkoituksena on piilottaa ydintieto käyttäjältä ja tämän jälkeen antaa käyttäjälle työkaluja joiden kautta hallinnoida tätä piilotettua tietoa. MDS pyrkii näin ohjaamaan käyttäjää rakentamaan ydintiedosta oikean muotoista.

MDS tarjoaa myös tarpeelliset perustyökalut ylläpitoon, kuten versiohistorian, mahdollisuuden varmuuskopiointiin ja tiedon palauttamisen. Työkalut eivät itsessään anna suurempaa lisäarvoa, sillä ne voidaan jo nykypäivänä laskea pakollisiksi ominaisuuksiksi Microsoft MDS:än kaltaisissa ohjelmissa. On kuitenkin hyvä muistaa, että MDS rakentaa omanlaisensa tietomallin, johon se tallentaa tietoa, jolloin esimerkiksi suoraan MDS:än käyttämästä kannasta katsomalla on mahdotonta esimerkiksi katsoa jonkin tietueen historiaa.

Projektissa MDS oli tärkeässä roolissa, sillä sen kautta asiakkaalle rakennettiin uusi tietomalli. Kaikki toteutetut rajapinnat käyttivät tätä tietomallia, joten MDS oli suuri osa rakennettua ratkaisua.

5.2 Master Data Managementin tarkoitus

Ydintiedon hallinnointi on tärkeä osa erilaisten ydintietopalveluiden toiminnollisuutta. Monissa tapauksissa se on itse palvelun päämäärä. Ydintiedon oikea muoto, yleisyys sekä toimivuus ovat asioita, joiden puutteet voivat hetkessä rikkoa niiden ympärille rakennetun järjestelmän. (Wolter & Haselden 2006)

Ydintiedolla tarkoitetaan yleistä pysyvää dataa, jota käytetään laajasti koko järjestelmän halki. Tallaista dataa voivat olla esimerkiksi toimipaikat, työntekijät tai sopimukset. Tämän datan pitää pysyä puhtaana, sillä sen laajan käytön takia vääränlainen muotoilu tai väärä tieto voivat nopeasti johtaa laajoihin virheisiin. Integraatiossa ydintieto saa vielä suuremman merkityksen, sillä kaikki integraatiossa olevat ohjelmat käyttävät samaa dataa. Tiedon puhtaudelle tulee tästä johtuen integraatioissa suurempi paino. Tämän

vuoksi pitää pystyä varmistamaan, että dataa syötetään oikealla tavalla koko integraation läpi ja itse dataan ei mieluiten kosketa suoraan tietokannasta. Tämän takia MDS:n tapaiset palvelut ovat käytännöllisiä, sillä ne piilottavat datan ja pakottavat tekemään sen muuttamisen itsensä kautta. Tällä tavalla voidaan varmistua ainakin jollain tasolla, että esimerkiksi viittaukset pysyvät ehjinä eikä tyhjiä viittauksia synny. Palvelut myös auttavat pitämään ydintiedon rakenteen kunnossa, jotta datan yleisyys säilyy eikä synny tilanteita, joissa dataa ei ole normalisoitu tarpeeksi.

6 RAJAPINTOJEN RAKENTAMINEN

Lukurajapintojen rakentaminen aloitettiin selvittämällä tulosten muoto ja pituus. Tämän jälkeen tietomallista etsittiin sopiva tieto ja se lisättiin paluusanomaan. Jos tietoa ei löytynyt, se pyrittiin lisäämään tietomalliin mielekkäästi. Joskus tämä tarkoitti lisätietotaulun käyttöä ja joskus uuden tietueen tekemistä. Yksi suurimpia haasteita sanomien rakentamisessa oli oman tietomallin käyttö. Tietomallien eroista johtuen joidenkin sanomien arvo oli koottu useista eri tuloksista tai sitä ei voitu tällä hetkellä lisätä ollenkaan.

Ongelmia synnyttivät myös esimerkkirajapintojen vaillinaiset sanomat. Tulokset, joissa arvo oli aina null ja joita ei käytetty laisinkaan mallirajapinnassa, piti jättää myös toteutetussa rajapinnassa tyhjäksi mahdollisten epäloogisuuksien välttämiseksi.

Rajapintoja rakennettaessa tuli esille MDS:n suurin ongelma. MDS:n toimintamallin takia monimutkaisemmat kyselyt alkoivat aiheuttaa suorituskyvyllisiä ongelmia. MDS:ään ei voi tehdä kokoavia kyselyjä eli kahden eri taulun välistä dataa ei pystytä hakemaan yhdellä kyselyllä. Suuremmissa kysymyksissä, joissa kyselyitä tehtiin melkein kaikkiin alkuperäiseen tauluun liittyviin tauluihin ja tämän jälkeen vielä niihin liittyviin tauluihin, huomattiin kyselyiden määrän eksponentiaalista kasvua. Kun yhden rivin hakemiseen alkoi mennä useita kymmeniä kyselyitä ja rivien määrä haussa oli useita satoja, alkoi herätä kysymys tehokkuudesta ja nopeudesta. Tehokkuuden lisäämiseksi tehtiin sama ratkaisu käyttäen hyväksi MDS:n mahdollisuutta luoda näkymiä sen sisältämistä tauluista. Tällä tavalla itse ydintietoon ei vieläkään koskettu, mutta sitä pystyi lukemaan normaalilla SQL-kyselyllä.

Lukurajapintojen lisäksi integraatioon on tarkoitus rakentaa kirjoitusrajapinnat, joiden avulla MDS:ään voidaan kirjoittaa uutta tietoa. Näissä kirjoitusrajapinnoissa pitää ottaa huomioon sekä kirjoittaja että kirjoitettava tieto. Alun perin rakennettu yleinen kirjoitusrajapinta, jonka avulla olisi periaatteessa mahdollista kirjoittaa mitä vain oikealla sanomalla, ei ole kuitenkaan tarpeeksi hyvä vaihtoehto: on esimerkiksi hyvin vaikeaa kirjoittaa paikka, jonka tiedot löytyvät monen taulun poikki, ellei tiedä tarkkaan mistä tiedot löytyvät. Tämän takia kyseisten tietojen kirjoittamiseen on rakennettava omat rajapintansa, joiden sisään otettu sanoma olisi yksinkertaisempi ja jotka hoitavat tiedon oikean paikoittamisen kirjoittajan puolesta.

Rajapintojen rakentaminen oman tietomallin päälle aiheuttaa oman ongelmansa suhteessa mallirajapintaan. Koska molempia ratkaisuja kehitetään rinnakkain, syntyy ylläpidollinen ongelma: vaikka toinen ratkaisu valittaisiin määrääväksi versioksi ja muiden ratkaisujen tulisi seurata sitä, siihen tehdyt muutokset eivät ole käytettävissä toimeksiannon versiossa. Tästä johtuen on tulevaisuudessa oletettavaa, että kehitystyö tulee olemaan suurena osana jatkoprojekteja, kun nykyisiä rajapintoja päivitetään yhtenäisiksi mallirajapintojen kanssa.

7 CAMEL PROJEKTISSA

Merkittävä tekijä integraation rakentamisessa skaalautuvaksi on jokin middleware, jonka kautta toimintalogiikan rakentaminen olisi mahdollisimman helppoa. Tällä hetkellä integraatioratkaisu on toteutettu seuraten ennemmin point-to-point-mallia kuin middlewaren avulla tehtyä ratkaisua.

Camelin avulla rakennetut reitit ovat helpommin muokattavissa kuin itse palveluiden sisällä olevat reitit, koska ohjelmoinnin osuus on pienempi. Reiteissä määritetään päätepisteitä, joita Camel kutsuu saaduilla arvoilla. Nämä arvot voivat olla ennalta määritettyjä, tai toisesta päätepisteestä palautuneita. Tämän avulla voidaan esimerkiksi kutsua kolmea päätepistettä, joista rakennetaan kokonaisuus, ja tämä kokonaisuus palautetaan lopulliselle päätepisteelle. Camelia käyttämällä voidaan rakentaa uusia reittejä nopeammin ja varmemmin verrattuna vaihtoehtoihin, joissa koko reitti rakennetaan itse ohjelmien sisälle – etenkin tapauksissa, joissa tiedon muoto on ongelma.

Ongelma Camelin käyttöönotossa on, että jokainen rajapinta tai luokka, joka halutaan käytettäväksi OSGi:n sisällä, vaatii, että siitä on tehty yhteensopiva kyseisen teknologian kanssa. Uusissa ratkaisuissa ohjelman määrittäminen toimivaksi ei ole ongelma, mutta vanhemmissa ohjelmissa se voi olla työlästä.

Camel hyödyntää ServiceMixissä myös ActiveMQ:ta, jonka avulla toimintojen jonottaminen on mahdollista. Tämä tarkoittaa, että viestiketjun sisällä on mahdollista lähettää tietoa ActiveMQ:n jonoon päätepisteen sijaan. Tämä tieto pysyy jonossa, kunnes jonoon liitetty reitti käynnistetään. Projektissa tätä ominaisuutta olisi paras käyttää ylläpidon sähköposti-ilmoituksille. Asiakas haluaa, että virheiden sattuessa ylläpidolle koottaisiin ja lähetettäisiin virheilmoitus, jotta ohjelman toimivuus olisi helpommin seurattavissa. Virheitä voi kuitenkin tapahtua paljon, ja jos jokaisesta virheestä lähetetään erikseen sähköposti, viestien määrä voisi nopeasti kasvaa liian suureksi. Tämän takia olisi viisainta koota tietoa jonoon, joka lähetettäisiin säädetyin aikavälein ylläpidolle. Tiedosta voitaisiin rakentaa yhteenveto ja yksityiskohtaisemmat tiedot olisivat liitteenä mukana. Näin ylläpito voisi helposti nähdä kuinka monta varoitusta ja virhettä on tullut ja tarvittaessa tarkistaa tarkemmat tiedot lokista.

Yksi Camelin ylläpitoa helpottavista tekijöistä on, että reittejä on helpompi seurata kuin ohjelman sisään rakennettua logiikkaa. Cameliin ja ServiceMixiin löytyy valmisratkaisuja,

joiden tarkoituksena on visualisoida OSGi:n sisältämiä komponentteja. Camel-reittien visualisointi yksinkertaistaisi ylläpitoa antamalla helposti tulkittavan kuvan reitin toiminnasta. Tällaiset ratkaisut antavat myös tilastoja reittien käytöstä, jolloin voidaan esimerkiksi seurata, sattuuko jossain reitissä useasti virheitä, tai tarkkailla reitin ajoaikaa ja käyttötiheyttä, jolloin järjestelmän mahdolliset pullonkaulat voidaan tunnistaa.

Ongelma ylläpidollisen datan tuottamisessa on asiakkaan tarkoitus ajaa palveluitaan klusterissa, jonka takia dataa keräävän osan pitäisi pystyä toimimaan myös klusterissa. Tämä tarkoittaa, että sen pitää pystyä valvomaan useita OSGi säiliöitä ja keräämään yhtenäistä dataa reittien käytöstä. Tämän aikaansaamiseksi ratkaisun pitäisi pystyä yhdistämään samanlaisten Camel-reittien tilastot.

8 RATKAISUJEN VERTAILU

Suurena osana projektia oli erilaisten ratkaisumallien vertailu sekä testaus. Projektin alussa valittu ServiceMix-integraatioalusta on pohjimmiltaan minimiratkaisu integraatioalustaksi ja tämän takia siitä puuttuvat monimutkaisemmat ominaisuudet, kuten visuaalinen tieto alustan sisäisistä toiminnoista. Lisäksi ServiceMix on moneen pienempään osaan jakautunut projekti ja on siksi muuttunut ennemminkin kokoavaksi projektiksi. Tästä johtuen lisäominaisuuksien julkaisu ServiceMixiin on kyseenalaista ja on todennäköisempää, että kyseiset ominaisuudet on rakennettava itse. Näistä syistä johtuen oli tärkeää kartoittaa vaihtoehtoja ServiceMixin tilalle ja selvittää, mitä ominaisuuksia ne tarjoaisivat.

Tarkemman tarkkailun alle otimme Fabric8 integraatioalustan, joka toimi samalla pohjaratkaisulla kuin ServiceMix. Fabric8 oli sen vuoksi hyvä esimerkki, että se sai alkunsa, kun ennen ServiceMixin parissa työskennelleet henkilöt olivat siirtyneet kehittymään Fabric8:iä ServiceMixin kehityttyä olemaan toisia ratkaisuja kokoava kokonaisuus. Tämän takia Fabric8:saa voidaan pitää eräänlaisena esimerkkinä siitä mitä ServiceMix voisi olla ja joka on todennäköisesti ServiceMixiä monipuolisempi. Fabric8 toimi Linux-pohjaisella alustalla, minkä takia sitä ei voitu käyttää projektissa. Fabric8 antoi hyvän käsityksen siitä, kuinka yksinkertainen ratkaisu ServiceMix pohjimmiltaan on ja mitä siihen pitäisi lisätä, jotta se täyttäisi asiakkaan vaatimukset.

Projektissa on myös alettu tutkimaan MDS:n toimivuutta ja sen tilalle on esitetty CodeServer-nimistä ydintiedon hallinnointiratkaisua. Codeserverillä tehtyä ratkaisua verrataan MDS:n ratkaisuun ja tämän perusteella pyritään saamaan käsitys, onko vaihto kannattava. Vertailu ja mahdollinen vaihdos kannattaisi tehdä tässä vaiheessa, kun muutos tulisi ainoastaan yhteen palveluun, jolloin muutoksen aiheuttamat kustannukset olisivat vielä melko pienet.

Kahta samankaltaista teknologiaa verrattaessa on tärkeää pitää mielessä, että tietolähteenä ei saa käyttää itse teknologia omia kotisivuja muuhun kuin sen toiminnollisuuden ymmärtämiseen. Tuotteista halutaan antaa mahdollisimman hyvä kuva asiakkaille, joten kotisivut ovat yleensä täynnä suuria lupauksia ja niissä harvoin mainitaan huonoja puolia. Tämän vuoksi on tärkeää etsiä mahdollisia projekteja, joissa teknologioita on käytetty, ja pyrkiä selvittämään, mitä niistä on sanottu ja mitä niihin on jouduttu lisäämään

tai tekemään, että ne on saatu toimimaan. Tietoja saatiin myös kokeilemalla teknologioiden toimintaa itse ja rakentamalla samankaltaisia ratkaisuja verrattavilla tuotteilla. Tärkeää on, että kutsu ja tulos ovat molemmissa ratkaisuissa samanlaisia ja välissä tapahtuvaa tarkastellaan. Vasteajat, rakentamisen ja ohjelmoinnin vaikeus, teknologian päivitystahti, hinta ja sopivuus ovat yleensäkin hyviä vertailtavia ominaisuuksia.

9 JOHTOPÄÄTÖKSET

Työssä on käyty läpi kattavasti integraation peruspiirteet ja esitelty, miten muutama ratkaisumalli toimii. Tämä on kuitenkin vain pieni osa siitä, mitä integraatio todellisuudessa on. Toimeksiannon puitteissa haettu tieto integraatiosta ja integraatioalustoista on kuitenkin tarpeeksi ja sen pohjalta saatiin tehtyä toimeksiannossa määritelty ratkaisu. Ratkaisu ei ole kuitenkaan valmis, vaan tämä on vasta ensimmäinen vaihe.

Integraatioalusta tullaan vielä muuttamaan, kun siihen lisätään uusia ohjelmia, ja tämän vuoksi selvitystyö mahdollisista lisäominaisuuksista oletettavasti jatkuu. On myös otettava huomioon, että kaikki rajapinnat tehtiin samalla kielelle, joten alustojen välistä toimivuutta ei työssä päästy lasinkaan testaamaan.

Projektissa itse rajapintaratkaisu saatiin toteutettua hyvin, mutta se muistuttaa perus rajapintaratkaisua eikä hyödynnä itse integraatioalustaa kunnolla. ServiceMixin tarjoama valmis ESB-ratkaisu on kokonaan käyttämättä ja suositeltu Camel on edelleen vasta demovaiheessa. Tämän takia mahdolliset lisäykset ja uudet ominaisuudet nykyiseen ratkaisuun joudutaan tekemään itse palveluihin. Jos ratkaisun koon voitaisiin olettaa pysyvän nykyisellään, tämä olisi hyväksyttävää. Integraatioratkaisuun on kuitenkin tulossa uusia ohjelmia, jotka voivat olla rakenteeltaan hyvin erilaisia ja joiden halutaan mahdollisesti keskustelevan keskenään, eikä ratkaisuun ole tehty valmisteluja tätä varten. Jokainen uusi järjestelmä jouduttaisiin nykymallissa muokkaamaan sopivaksi.

ServiceMix on hyvä integraatioalusta pieniin ratkaisuihin, mutta jos siihen halutaan tuoda useita uusia ohjelmia, on siihen tuotava paljon uusia ominaisuuksia, jotta sen ylläpito olisi mahdollista. Jos projekti ollaan rakentamassa klusteriin, tulee esimerkiksi päivittämisestä vaikeaa.

Tulevaisuudessa selkeitä kehityskohteita ovat ESB:n oikea implementointi sekä ylläpidollisten toiminnollisuuksien rakentaminen. Ilman näitä lisäyksiä on hyvin mahdollista, että integraatioalusta ei tuo mitään lisäarvoa ja se aletaan nähdä ennemmin haittana kuin hyötynä.

Ratkaisun toteuttamisen lisäksi olisi tärkeää tehdä ohjeet integraatioalustaan tuotavien ohjelmien dokumentoinnista. Esimerkiksi muille näkyvät luokat ja mahdolliset määritykset sekä käytetyt tietokannat olisivat tärkeitä olla tiedossa, jotta voidaan olla varmoja,

että integraatioalustan ylläpito ja päivitys olisivat tulevaisuudessa mahdollisimman helppoja.

LÄHTEET

Aburva (2014) Difference Between Synchronous and Asynchronous Messaging? Viitattu 14.11.2016 http://peoplesofttutorial.com/difference-between-synchronous-and-asynchronous-messaging/.

Briefcliffe, Mike (2016) The pros and cons of system integration for small businesses. Viitattu 23.8.2016 https://www.hiscox.co.uk/business-blog/pros-cons-systems-integration-small-businesses/.

Carnegie Mellon University (2012) Software System Integration. Viitattu 11.11.2016 http://www.sei.cmu.edu/productlines/frame_report/softwareSI.htm.

Chappell, David (2004) Enterprise Service Bus. Sebastopol: O'Reilly Media Inc.

Enterprise Service Bus (2015) Viitattu 12.12.2015 Wikipedia.

Fenner, James (2003) Enterprise Application Integration Techniques. Viitattu 12.11.2015 http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/3C05-02-03/aswe21-essay.pdf.

Hanson, Jeffrey (2005) ServiceMix as an enterprise service bus. Use ServiceMix 2.0 as a service-oriented message routing framework. Viitattu 10.12.2015 http://www.javaworld.com/article/2072293/soa/servicemix-as-an-enterprise-service-bus.html.

Hohpe, Gregor & Woolf, Bobby (2004) Enterprise Integration Patterns. Designing, building and deploying messaging solutions. Boston: Pearson Educational, Inc.

Ibsen, Claus & Anstey, Jonathan (2010) Camel in Action. New York: Manning Publications Co.

Javlin (2015) Data Integration. Viitattu 15.12.2015 http://www.dataintegration.info/data-integration.

Javlin (2015) Master Data Integration. Viitattu 15.12.2015 http://www.dataintegration.info/master-data-management.

Javlin (2015) Data Migration. Viitattu 4.10.2016 http://www.dataintegration.info/.

Linthicum, David (2000) Enterprise Application Integration. Boston: Addison-Wesley.

Lithucum, David (2000) Enterprise Service Bus. Boston: Addison-Wesley.

Linthicum, David (2003) Next Generation Application Integration: From Simple Information to Web Services. Boston: Longman Publishing Co.

Loshin, David (2009) Master Data Management. Burlington: Elsevier Inc.

OSGI alliance (2016) The Benefits of Using OSGI. Viitattu 14.11.2016 https://www.osgi.org/developer/benefits-of-using-osgi/.

Serain, Daniel (2002) Middleware and Enterprise Application Integration. Berlin: Springer Science & Business Media

The Apache Software Foundation (2011) Apache ActiveMQ features. Viitattu 11.11.2016 http://activemq.apache.org/features.html.

The Apache Software Foundation (2015) Arcitechture. Viitattu 11.11.2016 https://camel.apache.org/architecture.html.

The Apache Software Foundation (2016) Apache CXF Software Architecture Guide. Viitattu 11.11.2016 http://cxf.apache.org/docs/cxf-architecture.html.

The Apache Software Foundation (2016 A) Apache Karaf Overview. Viitattu 11.11.2016 http://karaf.apache.org/manual/latest/overview.html.

The Apache Software Foundation (2014) What is ServiceMix 4? Viitattu 11.11.2016 http://servicemix.apache.org/docs/5.0.x/user/what-is-smx4.html.

Wolter, Roger & Haselden, Kirk (2006) The What, Why and How of Master Data Management. Viitattu 14.11.2015 https://msdn.microsoft.com/en-us/library/bb190163.aspx.