LAPIN AMK
Lapland University of Applied Sciences

SELVITYS INTEGRAATIO-OHJELMISTOISTA

Kimmo Kähkönen

Opinnäytetyö
Tekniikka ja liikenne
Tietotekniikka
Insinööri (AMK)

2016

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tekniikka ja liikenne
Tietotekniikka

Tekijä Kimmo Kähkönen Vuosi 2016
Ohjaaja Aku Kesti
Työn nimi Selvitys integraatio-ohjelmistoista
Sivu- ja liitemäärä 62 + 9

Nykypäivän liike-elämä laajoine ja usein hajanaisine tietojärjestelmäympäristöineen asettaa haasteen järjestelmien väliselle yhdistämiselle. Järjestelmäintegraatio on siksi tärkeä ala informaatioteknologiassa nyt ja tulevaisuudessa.

Järjestelmäintegraatioita on totuttu toteuttamaan erilaisin tavoin, ja usein niin sanotut point-to-point-ratkaisut tietojärjestelmien määrän noustessa ovat asettaneet yritykset yhä kasvavien IT-kustannuksien eteen. Integraatioita onkin pyritty helpottamaan ja tekemään joustavammaksi tarjoamalla erillisiä integraatio-ohjelmistoja. Palveluntarjoajia integraatio-ohjelmistojen osalta on paljon, ja ohjelmistojen ominaisuudet vaihtelevat.

Tässä opinnäytetyöraportissa on tehty selkoa järjestelmäintegraatioista yleensä, ja taustoitettu sitä kautta tarvetta erillisille integraatio-ohjelmistoille. Teoriataustaa on tämän lisäksi pyritty tarkentamaan käytännön osuudella, jossa muutamasta markkinoilla olevasta integraatio-ohjelmistosta on tehty pienien käytännön harjoituksien kautta tarkempaa selvyyttä.

Raportin luonne on osittain oppimispäiväkirjatyylinen. Se tarjoaa läpikatsauksen järjestelmäintegraatioihin ja niitä tukeviin sovelluksiin. Sitä ei kuitenkaan tule nähdä oppaana integraatio-ohjelmistojen käyttöön.

Avainsanat Järjestelmäintegraatio, sovellusintegraatio, integraatio-ohjelmisto, integraatio-alusta

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Technology, Communication and Transport
Degree Programme in Information Technology

Author	Kimmo Kähkönen	Year	2016
Supervisor(s)	Aku Kesti		
Subject of thesis	Investigation of Application Integration Softwares		
Number of pages	62 + 9		

Present-day business environment with vast and often disjointed information systems set a challenge for connecting different applications. Because of this, application integration is and will be a special and essential field in information technology.

The case with application integration is often complex and integrations are implemented as so-called point-to-point solutions. This increases the costs and the manageability gets more difficult. To resolve this dilemma there have been a range of specified integration software. There is also lot of service providers and the properties of software vary. This report will give an overview of application integration and sets a couple of specified integration application (ESB) in a pedestal to take a closer look for the properties of a typical ESB. The method for investigation of applications is through simple exercises.

The style of the report is partly diary-based describing the learning process. It provides a cross-section to the field of application integration and supporting software. However it should not be considered as a tutorial for the featured applications.

Key words: Enterprise Application Integration, ESB, Integration Software, Integration Platform

SISÄLLYS

KÄYTETYT MERKIT JA LYHENTEET

CRM	Customer Relationship Management, asiakastiedon ja –suhteiden hallintaan käytetty järjestelmä
CSV	Comma-separed values, tiedostomuoto, jossa tieto on tallennettuna taulukkomuotoisena tekstitiedostoon
EAI	Enterprise Application Integration, järjestelmäintegraatio
ERP	Enterprise Resource Planning, tuotannonohjausjärjestelmä
ESB	Enterprise Service Bus, arkkitehtuurimalli ohjelmistojen väliseen järjestelmäintegraatioon, voidaan myös käsittää integraatio-ohjelmistona
HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla
JDK	Java Development Kit, Javakehittäjille tarkoitettu ohjelmistokehityspaketti
JSON	JavaScript Object Notation, avoimen standardin tiedostomuoto tiedonvälitykseen
POC	Proof of Concept, malli jolla on tarkoitus esittää jonkin ratkaisun toteutuskelpoisuus (voi olla rajattu ominaisuuksiltaan)
Point-to-point	Järjestelmäintegraatiotyyppi jossa kaksi sovellusta on yhdistetty suoraan toisiinsa
SOA	Service Oriented Architecture, palvelukeskeinen arkkitehtuuri, jossa yhteydessä olevien tietojärjestelmien keskinäinen toiminta perustuu itsenäisiin ja avoimiin palveluihin
XML	Extensible Markup Language, tiedon esittämistavan standardi, usein tiedonvälityksessä käytetty formaatti

1 JOHDANTO

Keväällä 2015 muodostetun hallituksen laatimassa hallitusohjelmassa *Ratkaisujen Suomi* mainitaan digitalisaation kehittäminen yhdeksi Suomea eteenpäin ajavaksi voimaksi. Hallituskauden kärkihankkeisiin kuuluu julkisten palveluiden digitalisointi, jolla tavoitellaan muun muassa tuottavuusloikkaa julkishallinnossa. Myös yksityisen sektorin kannustamista digitalisoinnin talkoisiin esitetään lainsäädäntöön tehtävillä muutoksilla. (Ratkaisujen Suomi 2015, 26–27.) Nämä ovat hyviä tavoitteita ja tuonevat muiden hyötyjen lisäksi myös töitä suomalaiselle, osaavalle tietotyövoimalle. Yksi digitalisaation haasteista on erilaisten järjestelmien yhteensovittaminen. On saatu lukea esimerkiksi potilastietojen ja sähköisten resepteihin liittyvien tietojen mutkikkaasta siirtämisestä järjestelmästä toiseen. Työelämässä myös tuskaillaan usein, kun samoja tietoja joudutaan tallentamaan useisiin eri järjestelmiin erikseen. Opettajien, lääkäreiden ja viranomaisten arvokasta työaikaa saattaakin kulua turhaan manuaalisen tiedon tallentamiseen, eikä esimerkiksi asiakkaan tai oppilaan kohtaamiseen, aitoon vuorovaikutukseen (Hautamäki 2015, D3).

Tämä opinnäytetyö avaa järjestelmäintegraation moniulotteista ympäristöä. Sen tausta-/teoriaosuudessa kuvataan yleisellä tasolla erilaisia järjestelmäintegraatiotyyppejä ja pohjustetaan, mistä järjestelmien yhdistämisestä on käytännössä nykypäivän tietojärjestelmien tapauksessa kyse. Opinnäytetyössä tutustutaan myös, mihin suuntaan nykyiset pilvipohjaiset järjestelmät tai kasvava mobiililaitteiden määrä liiketoiminnassa integraatioita mahdollisesti vievät. Käytännön osuudessa luodaan katsaus avoimien ohjelmistojen valikoimaan ja tutustutaan konkreettisesti, miten toimivat vapaassa tarjonnassa olevat integraatio-ohjelmistot.

Aihealueen ollessa verrattain laaja, rajataan yleisen järjestelmäintegraation jälkeen järjestelmäympäristöä koskettamaan erityisesti liiketalouden digitaalisia sovelluksia. Digitaalisen taloushallinnon aihepiiri on kirjoittajalle tuttu, ja siihen liittyvä tietopohjan syventäminen on yksi opinnäytetyön tavoitteista. Opinnäytetyön teoreettisen osuuden ollessa varsin faktapitoinen, käytännön osuudessa

luodaan kokeellisempi ympäristö: järjestelmäintegraatioon liittyvää toimeksiantajaa työllä ei ole, joten kaikki työssä esitetyt esimerkit ovat kuvitteellisia. Taustasta huolimatta työn käytännön esimerkit on pidetty mahdollisimman lähellä liike-elämän mahdollisia, todellisia tarpeita. Esimerkiksi millaisia järjestelmiä on tarve yhdistää ja miten yhdistämisessä voi integraatio-ohjelmisto olla hyödyksi. Tutkimusongelmana työllä on yhtäältä integraatio-ohjelmiston tekninen toiminta, toisaalta taas uuteen ja outoon tutustumisen askelmerkkien sovittaminen.

Kirjallisen raportin rakenne jakautuu kahteen osaan: ensimmäisessä osassa perehdytään järjestelmäintegraatioon teoreettisessa viitekehyksessä kun taas jälkimmäinen osa keskittyy enemmän käytännön harjoituksiin integraatio-ohjelmistoilla. Teoriaosuudessa (luvut 2-3) on muun muassa kuvattu eri integraatiomenetelmiä, taustoitettu integraatio-ohjelmiston toimintaperiaatteita ja esitelty muutama keskeinen ohjelmisto ylätasolla. Käytännön osuus keskittyy kuvaamaan neljännessä luvussa kolmen eri integraatio-ohjelmiston toimintaa harjoitusten (nk. tapausesimerkkien) kautta. Viimeisessä luvussa on käyty läpi oppimisprosessia ja hieman järjestelmäintegraation tulevaisuutta.

2 JÄRJESTELMÄINTEGRAATIOT

2.1 Tietojärjestelmien integraatioista yleisesti

Tarve tietojärjestelmien integraatioille on ollut olemassa aina. Ongelma on ilmennyt eri tavalla tekniikoiden kehittyessä. Varhaisimpien tietojärjestelmien tapauksessa eri järjestelmien yhdistäminen saattoi tarkoittaa yksinkertaisesti sitä, että johonkin järjestelmään tallennettu tieto tallennettiin täysin manuaalisesti toiseen järjestelmään tai käyttäen esimerkiksi levykkeitä tai nauhavarmistuksia tiedon siirtoon. Sitä mukaa kun järjestelmät ja tietoliikennetekniikat kehittyivät, integraatioiden toteutus on muuttunut. Järjestelmäintegraatio voidaan määritellä ”toimintamalleiksi ja tekniikoiksi, joiden avulla voidaan saattaa vähintään kaksi eri toiminnallisuutta tarjoavaa tietojärjestelmää jakamaan informaatiota siten, että informaation siirto ja muunnokset ovat kontrolloitavissa ja monitoroitavissa yhdestä tai useammasta keskitetystä pisteestä” (Tähtinen 2005, 48).

Kuten yleensäkin tietotekniikassa, erilaisia tekniikoita, protokollia, määrittelyitä ja standardeja löytyy myös järjestelmäintegraation parista. Kirjallisuuteen ja palveluntarjoajiin tutustuessa törmää siihen, että järjestelmäintegraation tapauksessa kaikkia käsitteitä ja standardeja ei suinkaan ole vahvistettu tai yleisesti hyväksytty; osa palveluntarjoajista voi puhua samoista käsitteistä eri kirjainyhdistelmin kuin toinen. Ennen kaikkea tämä koskee erilaisia integraatio-ohjelmistoja. Lisäksi suomenkielisessä kirjallisuudessa ja palveluntarjoajien termeissä voi olla vaihtelua. Tämä on otettu raportissa huomioon.

Yhdistettävien järjestelmien määrän kasvaessa, myös kompleksisuus niin integraation toteutuksen kuin sitä koskevan määrittelynkin suhteen kasvaa. Tässä raportissa itse käsitteistö pyritään pitämään mahdollisimman suppeana. Kun tulee tarve yhdistää tietojärjestelmiä, voi kyseessä olla saman organisaation käytössä olevat järjestelmät (nk. EAI/Enterprise Application Integration) tai organisaatiorajat ylittävä järjestelmien yhdistäminen (B2Bi). Yhdistettävät järjestelmät voivat molemmissa tapauksissa olla toteutetuilta tekniikoiltaan mielivaltaisia (Microsoft 2004). Tässä raportissa ei jatkossa tehdä rajausta sen välillä,

onko kyseessä organisaatiorajat ylittävä järjestelmäintegraatio; jaon vaikutus on tässä tapauksessa ensisijaisesti järjestelmäarkkitehtuuri-kohtainen kysymys, eikä siten ollen olennainen (Tähtinen 2005, 33–34).

Lopuksi ennen kuin seuraavassa luvussa kerrotaan tarkemmin integraatioiden toteutustavoista, tarkastellaan muutaman konkreettisen esimerkin kautta tilanteita, joissa organisaatio(t) joutuvat hyödyntämään järjestelmäintegraatioita. Yksinkertaisimmillaan integraatiotarpeessa on kyse alkujaan tietojärjestelmien kehittymisestä ylipäätään mahdollistamaan integraatiot ja myöhemmin globalisaatio ja organisaatioiden lisääntynyt yhteistoiminta IT-järjestelmien tukemana on ollut keskeinen syy integraatioille (Manouvrier & Menard 2010, 5). Yhdellä organisaatiolla on usein käytössään monelta toimittajalta järjestelmiä eri tarpeisiin. Esimerkiksi ERP-järjestelmään (Enterprise Resource Planning, toiminnanohjausjärjestelmä) voi olla integroitu useita taloushallinnon erillisjärjestelmiä (Lahtinen & Salminen 2014, 41).

Käytännön sovellutuksia järjestelmäintegraatioille on helppo löytää. Seuraavassa on listattuna muutamien suomalaisten järjestelmäintegraatioita palvelunaan tuottavien yritysten case-esimerkkejä integraatiototeutuksista:

- Vantaan Energialle HiQ Finland toteutti integraatioratkaisun, jolla seurataan prosessien etenemistä, tietojen oikeellisuutta ja palvelutasojen (SLA) toteutumista reaaliajassa.
- Suomen Lähikauppa hallitsee integraatioiden avulla eri järjestelmissä tilausliikennettään (HiQ).
- Turun Energia hyödyntää raportointiratkaisussaan järjestelmäintegraatiota etäluettaviin mittareihin (HiQ).
- Televisioyhtiölle toteutettu integraatio yhdisti tilaus-, toimitus- ja laskutoiminnot sekä eri maissa toimivien partnereiden tiedonsiirrot (HiQ).
- Flahsnode on ratkaissut M Storen ja Pulju.net verkkokauppojen tuottamien myyntitietojen yhdistämisen kirjanpidon, laskutuksen ja

raportoinnin järjestelmiin säästäen aikaa ennen manuaalisina hoidetuista prosesseista.

- Lehtitalo Allerin haasteena ollut hajanaisten asiakastietojen sijainti useissa CRM-järjestelmissä ja toisaalta asiakkaiden pääsy muokkaamaan omia tietojaan ratkaistiin Youred Oy:n toimesta ketterällä integraatioteknologialla (teknologia-alustana Microsoft Azure).
- Teollisuuden kunnossapitoon ja käyttöpalveluihin erikoistuneen palveluyritys Maintpartnerin integraatiohaasteet liittyivät asiakkaiden tietojärjestelmiin kytkeytymiseen ja sitä kautta kankeisiin päivittäisiin prosesseihin asentajilla: Solita Oy rakensi prosesseja tukevan integraatiomallin MuleSoft Anypoint Platform –ohjelmistolla.
- Solita Oy auttoi myös Liikennevirastoa rautatieliikenteen ratakapasiteetin hallintaan usean rautatieoperaattorin toimintaympäristössä.
- Lassila & Tikanojan palvelutarjontaan kuuluu niin jätehuoltoa, kierrätystä, kiinteistöhuoltoa ja siivousta, mikä tuo myös tarpeen usealle eri järjestelmälle. Solita Oy rakensi Microsoft BizTalk –integraatioalustaan pohjautuvan, erillisen integraatiopalvelukeskuksen. (HiQ 2015; Flashnode 2016; Solita 2016; Youredi 2016.)

Edellisten esimerkkien tukemana, voidaan vielä käsitellä Roy Schulten esittämä kategorisointi integraatiotarpeista. Hänen mukaansa järjestelmäintegraatio pyrkii toteutustavasta riippumatta vastaamaan vähintään yhteen seuraavista ongelmista

- informaation välitys järjestelmästä toiseen ja johdonmukaisuus
- monivaiheisten prosessien hallinta
- yhdistelmäsovelluksien (composite application) laadinta tarkoittaen käytännössä olemassa olevista tietovarannoista ja/tai järjestelmäpalveluista yhdistettyä uutta ohjelmaa (Manouvrier & Menard 2010, 14.)

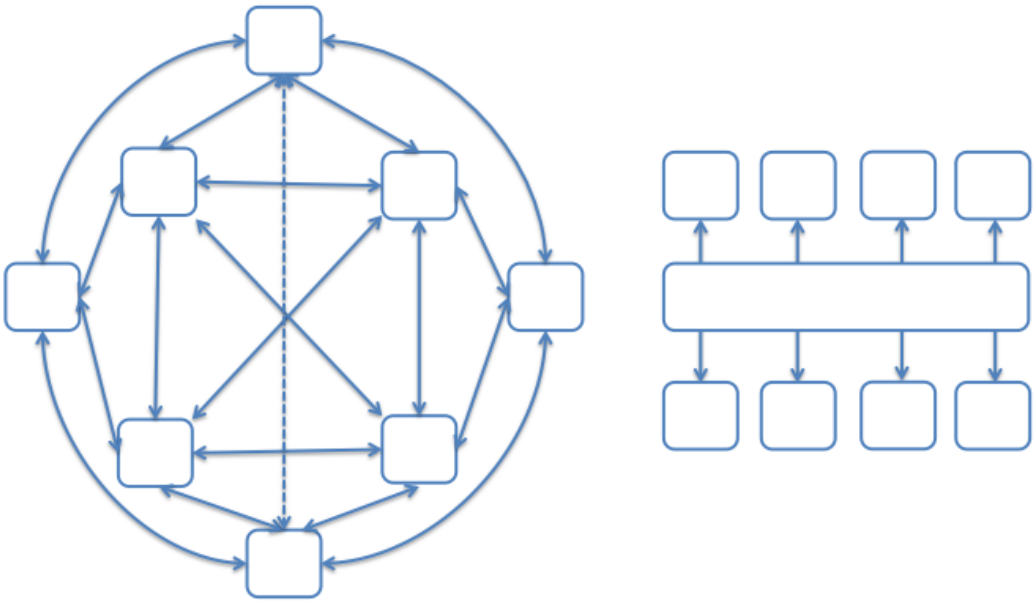
2.2 Integraatiotyypeistä

Järjestelmäintegraatiot voidaan jakaa karkeasti jaoteltuna kahteen tyyppiin menemättä vielä tarkemmin käytettäviin teknologioihin tai järjestelmiin. Suoraviivaisin ja yksinkertaisin integraatiotyyppi on point-to-point-integraatio, missä kaksi erillistä järjestelmää on yhdistetty keskenään. Toiseen integraatiotyyppiin voidaan laskea erilaiset palveluväylä- ja hub-ratkaisuilla toteutetut integraatiot, jotka perustuvat usean järjestelmän yhdistämiseen erillisen keskuksen, esimerkiksi niin kutsutun hubin kautta. Tämä malli mahdollistaa keskitettyä ylläpitoa integraatioissa ja vähentää myös kompleksisuutta, kun yhdistettäviä järjestelmiä on useampia.

Point-to-point-toteutusten vahvuutena on nopeus ja tehokkuus: tämä on yksi syy, miksi suoraan toisiinsa kytkettyjä järjestelmiä jatkossakin on (O'Brien 2008). Ehdottomana heikkoutena on kuitenkin niiden ylläpidettävyys. Mitä enemmän toisiinsa liitettäviä järjestelmiä on, sitä enemmän on järjestelmien välisiä liittymiä ja sitä kautta ylläpidettävää.

Yhtälö 1: $y = n(n-1)/2$

Yhtälössä 1 n on yhdistettävien järjestelmien määrä. Y kertoo tarvittavien yhteyksien määrän. Kahden järjestelmän tapauksessa yhteyksiä on yksi, kymmenen järjestelmää yhdistettäessä yhteyksiä on jo 45. Tässä tietysti oletetaan, että kaikkien järjestelmien tulee olla yhteydessä toisiinsa (mikä ei luonnollisesti aina ole käytännössä tarpeellista). Joka tapauksessa useampi erikseen integroitu järjestelmä johtaa lopulta vaikeasti hallittavaan ”spagetti-solmuun”. Kuvioon 1 on avattu point-to-point- ja hub-and-spoke-integraatiotyyppien eroa hallittavuuden näkökulmasta. (Tähtinen 2005, 66 ; Manouvrier & Menard 2010, 12.)



Kuvio 1: Erilaiset integraatiotyypit

Keskitetty integraatioratkaisu tarjoaa point-to-point-yhteyksiä parempaa hallittavuutta ja tehokkuutta. Tiedonsiirtoa eri järjestelmien välillä voidaan monitoroida keskitetysti yhdeltä palvelimelta tai työasemalta. Keskitetyn ratkaisun etuna on myös rajapintojen määrän kasvu lineaarisesti yhdistettävien järjestelmien suhteessa. Keskitetyssä ratkaisussakin on toki huonoja puolia: vikatilanteissa integraatioratkaisu voi asettaa yhdestä pisteestä koko liiketoiminnan tärkeät tietovirrat riskialttiiksi. (Tähtinen 2005, 66–68.)

Keskitetty integraatioratkaisu voidaan määritellä tarkemmin toteuttavan arkkitehtuurin pohjalta. Aikaisemmin käytössä olleista niin kutsutuista hub-and-spoke-malleista on siirrytty sittemmin hajautettujen toteutusten kautta palveluväylä- ja integraatioalusta-toteutuksiin, jotka noudattelevat seuraavassa alaluvussa tarkemmin avattua SOA-arkkitehtuurimallia. (O'Brien 2008.)

Liiketoiminnan tarpeen tulisi olla peruste valittavalle integraatiotyypille. Tekniikka sinänsä ei tee autuaaksi. Siinä mielessä keskitetyn integraatioratkaisun hyödyntäminen ei ole perusteltua, mikäli yhdistettäviä järjestelmiä on vain kourallinen. Nykyään markkinoilla olevat integraatio-ohjelmistot (joihin tässä raportissa ensisijaisesti keskitytään) ovat pullollaan toiminnallisuuksia, mutta nekin voivat pahimmassa tapauksessa haitata alkuperäistä ideaa, sovellusten vaivatonta yhdistämistä (Fowler & Webber 2008).

Oli toteutustapana mikä tahansa, integraatiototeutuksen palvelut voidaan jakaa kolmeen tasoon: tiedonsiirto ja liitettävyys, tiedon sovittaminen, sekä viimeisenä liiketoimintaprosessien automatisointi (Manouvrier & Menard 2010, 29 ; Tähtinen 2005, 48). Tätä jakoa tarkennetaan edelleen, kun seuraavissa luvuissa esitetään tarkemmin tyypillisen integraatio-ohjelmiston tehtävät.

2.3 SOA – Service Oriented Architecture eli palvelukeskeinen arkkitehtuuri

SOA-termi (Service oriented architecture l. palvelukeskeinen arkkitehtuuri) esiintyy usein puhuttaessa nykypäivän järjestelmäintegraatiototeutuksista. Pelkistetyimmillään määriteltynä sillä tarkoitetaan arkkitehtuurimallia, jossa erityyppiset ohjelmistot (esim. taloushallinnon kirjanpitosovellus, ERP-järjestelmän tilauksenhallintamoduuli tai asiakkuudenhallintajärjestelmä CRM) julkaisevat ulospäin palveluita, joita muut järjestelmät pystyvät tarvittaessa kutsumaan. Lisäksi koska kyseessä on arkkitehtuurimalli, eikä puhtaasti teknologia, SOA on riippumaton laitteistoista ja ohjelmointikielistä. (Tähtinen 2005, 96–97, 145.)

SOA-käsitteen ollessa tärkeä myöhemmin esitettävien integraatio-ohjelmistojen toiminnallisuuden kannalta, esitellään sen periaatteita tässä vielä hieman tarkemmin. Palvelukeskeisyydestä puhuttaessa on olennaista tietää, mitä ovat palvelut tässä tapauksessa. Hyvät esimerkit on helpointa johtaa käytännöstä: ajatukset palveluista selkiytyvät, kun tarkastellaan yrityksen tavoitteita liiketoiminnan kannalta, mitä tehtäviä/toimintoja sen tulisi tarjota, ja miten kyseiset tehtävät voisi tarjota teknologian tai ohjelmistojen kannalta. Pankki, yhtenä esimerkkinä, voisi tarvita *talletus(), nosto() ja tilisiirto()* -toimintoja. Kun kyseiset

toiminnallisuudet on toteutettu siten, että niitä on hyödynnettävissä muista sovelluksista, ollaan hyvin lähellä SOA:n ydinperiaatteita. (Havey 2008, 9–10.)

Edellisen kappaleen lähes liiallisen pelkistetty abstraktio kertoi yhden esimerkin palveluista. Samalla tapaa myös esimerkiksi vakuutuksia tarjoavalta yritykseltä on löydettävissä palvelut, jotka ovat liiketoiminnan informaation kanssa tiiviisti tekemisissä (eri vaiheet asiakkaan vahinkoilmoituksesta maksuun saakka). ”Palvelu on siis yrityksen asiakkaalleen tuottama hyödyllinen ydintoiminto” (Dikmans & van Luttikhuizen 2012, 32). Tekniikoista puhuttaessa SOA:n yhteydessä on vallalla väärinkäsitys, jonka mukaan palveluita SOA-periaatteiden mukaisesti voisi tarjota tietojärjestelmistä vain web serviceiden avulla. Palveluita voidaan toteuttaa kuitenkin myös esimerkiksi PL/SQL-proseduurein ja .NET-luokkametodein (toisin sanoen lähes minkä tahansa ohjelmointikielen avulla). Palvelukeskeisyys ulottuu liiketoiminnan prosesseista tietojärjestelmien tarjoamiin toimintoihin ja talletettuihin tietoihin. (Dikmans & van Luttikhuizen 2012, 11, 47.)

Tämän kaiken yhteys järjestelmäintegraatioihin voidaan johtaa hyötynäkökulmasta. Selkeintä on nähdä SOA-arkkitehtuurimallin tarjoamat hyödyt integraatioiden toteuttamiselle. Jos tietojärjestelmät on toteutettu SOA-periaatteiden mukaisesti, niissä on määrittelyn mukaisesti järjestelmän ulkopuolelle tarjottavat palvelut, joilla muun muassa tietoa voidaan viedä ja toisaalta hakea. (Microsoft 2015.)

2.4 BPM – Business Process Management

Jos ajatellaan puhtaasti teknisessä mielessä, liiketoimintaprosessit ja niiden kuvaaminen tai hallinta eivät liity järjestelmäintegraatioihin. Tämän raportin osalta liiketoimintaprosessit on ajateltu kuitenkin avaavan ja syventävän olennaisesti järjestelmäintegraation tarvetta ja perusteita. Pelkän järjestelmän tai järjestelmien tarjoamien ominaisuuksien ei tulisi olla ajureina liiketoiminnan prosesseille, vaan päinvastoin; niin järjestelmävalintojen kuin niihin liittyvien integraatiotarpeidenkin perusteet tulee olla liiketoimintalähtöisiä. Tämän huomaa myös

järjestelmäintegraatioihin liittyvässä kirjallisuudessa. BPM eli liiketoimintaprosessien mallintaminen on sisällytetty useaan järjestelmäintegraatiota käsittelevään perusteokseen.

Liiketoimintaprosessilla tarkoitetaan toisiinsa liittyvien toimintojen ja tehtävien muodostamaa kokonaisuutta, joka alkaa asiakkaan tarpeesta ja päättyy asiakkaan tarpeen tyydyttämiseen (Berman 2014, 12–13). Jos SOA-arkkitehtuurin mukainen palvelu oli nähtävissä yksittäisenä toimintona, liiketoimintaprosessin voi katsoa kokoavan useamman toiminnon yhdeksi kokonaiseksi ketjuksi.

Raportin toiseksi viimeisessä luvussa on kuvattu esimerkinomaisesti liiketoimintaprosessi (yrityksen taloushallintoon kuuluva matkalaskujen käsittely). Kuvauskielestä riippuen symbolit voivat vaihdella, mutta yleisesti ottaen hyvästä prosessikuvauksesta on nähtävissä/tunnistettavissa järjestelmässä liikkuva tieto, ja järjestelmien välisien rajapintojen yhdistämisessä hyödynnetään usein järjestelmäintegraatiota, kun huonoimmassa tilanteessa vaihtoehtona voisi olla tietojen kopiointi täysin manuaalisesti järjestelmästä toiseen.

Yksi syy, miksi liiketoimintaprosessien voi tulkita liittyvän tiiviisti järjestelmäintegraatioihin, onkin prosessien jakautuminen useimmiten useiden eri järjestelmien alueelle. Kun vielä yksittäinen liiketoimintaprosessi voi jakautua usean yrityksen välillä, on tarpeena siirtää erityyppistä informaatiota, esimerkiksi tilauksia, tilausvahvistuksia, osto- ja myyntilaskuja ja niin edelleen. On helppo kuvitella järjestelmäintegraation tuovan lisäarvoa, kun useat eri toimittajien, eri alustoilla ja eri tekniikoilla toimivat ohjelmistot ”pakotetaan” keskustelemaan keskenään. (Tähtinen 2005, 35.)

On olemassa teknisiä työkaluja ja käytänteitä liiketoimintaprosessien mallintamiseen ja näistä on mahdollista edelleen jalostaa suoraan hyödynnettäviä tietovirtakaavioita käytettäväksi järjestelmäintegraation tueksi. Johtuen useasta eri standardista näihin ei tässä vaiheessa syvennytä tarkemmin. Raportin käytännön osuudessa integraatiosovelluksen tarjoomaa tämän osalta kuitenkin käydään läpi. Yhteenvetona liiketoimintaprosessien hahmottaminen ja kuvaaminen

ei pelkästään kehitä yrityksen toimintoja, mutta auttaa myös järjestelmävalinnoissa ja järkeistää järjestelmäintegraatioita.

3 INTEGRAATIOSOVELLUKSET

3.1 Yleisesti integraatiosovelluksista

Kolmas luku syventyy kahden ensimmäisen, hieman teoreettisemman luvun jälkeen enemmän tekniikkaan ja käytännön sovelluksiin. Tämä näkyy myös termien määrittelyissä; kun aikaisemmissa luvuissa termien ja paradigmojen määrittely oli kirjallisuudessa yhdenmukaista, tekniikoiden osalta muodostuu hankalammaksi löytää yleisesti hyväksyttyjä määritelmiä. Integraatiosovelluksilla on jo historiaa niiden kehityttyä yhdistettävien sovelluksien muuttuessa muun muassa rajapintojen myötä. Integraatiosovelluksien sanastoa ja termejä ovat olleet luomassa ennen kaikkea sovellustoimittajat. Tästä johtuen samat termit voivat valmistajasta riippuen tarkoittaa eri asiaa tai samalla toteutuksella voi valmistajasta riippuen olla eri nimityksiä. Lähin analogia löytyy esimerkiksi autoista ja dieselmoottoreista: Volkswagenilla TDi, Audilla Turbodiesel ja niin edelleen.

Tässä raportissa integraatiosovelluksella tarkoitetaan englanninkielisessä kirjallisuudessa Enterprise Service Bus (ESB) -nimellä esiintyviä sovelluksia. Määrittely ei ole aukoton, sillä suomenkielisessä tekniikan sanastossa ei ole vakiintunutta määritelmää ESB- tai integraatiosovelluksille. Termit tekniikoiden osalta vaihtelevat integraatiosovelluksista integraatioalustoihin ja integraatiojärjestelmistä integraatioratkaisuihin.

Tässäkin raportissa paljon lähteenä käytetyssä Järjestelmäintegraatio-kirjassa Tähtinen tekee eron integraatioalustan ja -ohjelmiston välille integraatiotyyppien pohjalta. Integraatioalustalla Tähtinen viittaa aikaisemman sukupolven hub-and-spoke-arkkitehtuurien toteutuksiin, kun taas integraatio-ohjelmistolla hänen voi katsoa viittaavan edellistä arkkitehtuuria kehittyneempää eli SOA:a toteuttavaan ESB:hen (Tähtinen 2005, 143, 145). Toisaalta tämänkaltaista jakoa ei suoraan englanninkielisestä kirjallisuudesta löytynyt. Yhtälailla myös integraatioalusta-termiä käytetään tarkoittamaan nykyaikaisia toteutuksia, jotka huolehtivat palve-

luiden välisestä viestinnästä (eli SOA:n ydinperiaatteiden mukaisesti) ja toisaalta ESB:hen viitataan palveluväylä-käsitteellä (Malinen 2013).

3.2 ESB – Enterprise Service Bus

Aikaisemmista raportissa esitetyistä termeistä ja tekniikoista poiketen ESB-termillä ei ole yleisesti hyväksyttyä määritelmää. Ensinnäkin kyseessä on lähteestä riippuen arkkitehtuurimalli tai valmis teknologia (Herreira 2010 ; Del Rio Benito & Edidin 2013, 6). Sen tarkoituksena on toimia organisaation tietojärjestelmien yhdistämisessä *selkärankana*, jonka läpi tietosanomat kulkevat (Laliwala, Samad, Desai & Vyas 2013, 8).

ESB on siis käytännössä kokoelma teknologioita, joilla voidaan toteuttaa palvelukeskeistä arkkitehtuurimallia ohjelmistojen yhdistämisessä (Tähtinen 2005, 145). Kuten sanottu, täysin ongelmaton ESB:n rinnastaminen suoraan integraatiosovellukseksi ei ole. Seuraavien kahden aliluvun osalta rinnastusta on kuitenkin käytetty, kun kuvataan integraatiosovelluksien yleisimpiä ominaisuuksia ja toisaalta näiden kautta selkeitä käyttökohteita.

3.2.1 ESB:n ominaisuudet

Integraatiosovellus (ESB) on joukko teknologioita, joilla toteutetaan SOA-arkkitehtuurin mukaista ohjelmistojen yhdistämistä. Voidaankin listata joukko tyypillisimpiä tehtäviä, joita integraatiosovellus hoitaa. Nämä ovat aikaisemmassa luvussa esitetyn teoreettisemman pohjan syventämistä teknisempään suuntaan. Yksinkertaisimmillaan integraatiosovelluksen täytyy huolehtia informaation siirtämisestä järjestelmien välillä, mahdollisista tietomuunnoksista järjestelmien sisäisten esitysmuotojen välillä ja kontrolloida kokonaisprosessia edellä mainituista tehtävistä (Tähtinen 2010, 48).

Menemättä vielä yksittäisten sovellustoimittajien tuotteisiin, seuraavassa listatut, edellistä yksityiskohtaisemmat perusominaisuudet käytännössä myös määritte-

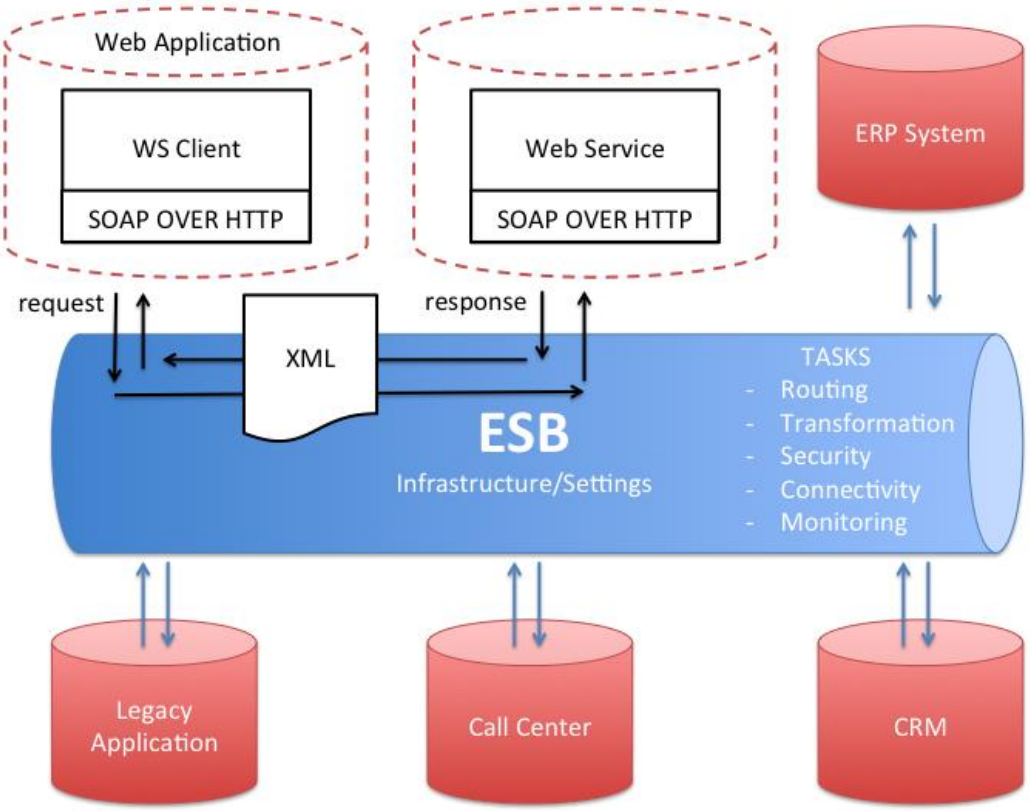
levät integraatiosovelluksen ja ovat tavallaan vähimmäisvaatimuksia ESB-järjestelmälle:

- paikkatuntumattomuus (location transparency)
    - järjestelmien fyysisellä sijainnilla ei ole merkitystä, vaan ESB huolehtii tietoliikenteen reitityksestä toisin sanoen lähdejärjestelmän ei tarvitse tietää kohdejärjestelmän sijaintia
- tietoliikenneprotokolla konversio
    - yhdistettävät järjestelmät voivat käyttää tiedon tuontiin ja vientiin eri protokollia (esim. http, JMS, FTP, SMPT tai TCP), integraatiosovelluksen on kyettävä muuntamaan tarvittaessa protokolla
- viestimuunnokset
    - eri tiedostoformaattien muuntaminen, esimerkkinä XML JSONiksi
- viestiliikenne
    - ESB:n yksi olennaisimmista toiminnoista on huolehtia tiedonvälityksestä ja tarvittaessa päätellä saapuvan viestin kohdejärjestelmä
- viestin rikastaminen
    - joissain tapauksissa alkuperäisen viestin välittäminen kohdejärjestelmään vaatii tiedon täydentämistä (esim. etuliittein)
- tietoturva
    - laaja kokonaisuus, johon voidaan sisällyttää niin autentikointi, valtuuttaminen, tietojen salaus
- valvonta ja hallinta
    - välttämätön ESB:n tukitoiminto, jolla viestiliikennettä ja virhetilanteita voidaan valvoa ja korjata

(Rademakers & Dirksen 2009, 13)

Edellinen luettelo on yhden asiantuntijan näkemys, mutta suuria eroja ESB:n toiminnallisuuksissa ei eri lähteissä ole löydettävissä huolimatta varsinaisen ”standardi-ESB:n” puutteesta. Liitteestä 1 (Kress ym. 2013) löytyy vielä kuvaus vaihtoehtoisesta, hieman teknisemmästä toimintoluetteloinnista. Se on laadittu Oracle Technology Networkin tarjoamista materiaaleista, teknisiä termejä ei ole käännetty. Kuvassa 2 on havainnollistettu ESB:n toimintaa. Ero esimerkiksi vastaavien järjestelmien yhdistämiseen perinteisellä point-to-point-ratkaisulla on selkeä.

Alla olevassa kuviossa 2 on esitetty ESB:n toimintaa ylätasolla. Siinä on usea sovellus, joka keskustelee palveluväylän/integraatiosovelluksen kanssa ja välityksellä. Esimerkinomaisesti web-pohjainen sovellus on lähettänyt palvelupyynnön ja saa siihen vastauksen (SOA:n periaatteiden mukaisesti). Kyseinen pyyntö-/-vastausrutiini on toteutettu http:n kautta XML-tiedostoa hyödyntäen. ESB:n on mahdollista toteuttaa välitettävään XML-tiedostoon esimerkiksi tiedon rikastamista tai muuntamista, jotta kutsuva sovellus voi ottaa tiedon vastaan. Raportin käytännön osuus keskittyy tekemään selkoa ESB-sovelluksen toiminnasta erilaisten ohjelmien tietoliikenteen yhdistämisessä hyödyntäen riittävän laajasti edellä mainittuja perusominaisuuksia.



Kuvio 2: ESB:n toiminta (mukaillen Juric, Loaganathan, Sarang & Jennings 2007, 273; Rademakers & Dirksen 2009, 7)

3.2.2 Integraatiosovellusten yleisimmät käyttökohteet

Yksi selkeä tekijä integraatiosovellusten käyttötarpeelle löytyy jo aikaisemmin esitellystä ongelmasta point-to-point-integraatioissa: kun yhdistettävien järjestelmien määrä kasvaa, myös ylläpidettävien liittymien määrä voi nousta lopulta hallitsemattomaksi. Myös tilanteessa, jossa yrityksellä ei itsellään ole ylläpitoa yhdistettävään sovellukseen, vaan ainoastaan saapuvaan viestiliikenteeseen, voi integraatiosovellus huolehtia riittävän palvelutason valvonnasta. Tavallaan integraatiosovellus voi myös siis vapauttaa yrityksen hyödynnettävien sovellusten täydestä omistajuudesta. Kun lisäksi järjestelmien saapuvan ja lähtevän viestiliikenteen protokollat vaihtelevat, tarjoaa integraatiosovellus näppärän keinon hallita keskitetysti sanomaliikennettä. (Rademakers & Dirksen 2009, 4-7; Kress ym. 2013.)

Kun muistetaan ESB:n toteuttavan palvelukeskeisen arkkitehtuurin periaatteita, integraatiosovellus parhaassa tapauksessa palvelee sitä hyödyntäviä sovelluksia tarjoten palveluiden rekisteröinnin, julkaisun, pyyntöjen välityksen ja myös tekniset liittymät yksittäisille sovelluksille (Tähtinen 2005, 100). Lopulta kuitenkin yrityksen on ratkaistava tietohallinnon ja kokonaisarkkitehtuurinsa näkökulmasta, hyödyntääkö erillistä integraatiosovellusta vai pitäytyykö point-to-point-integraatioissa. Mitään absoluuttista rajanvetoa integraatiosovelluksen käytölle ei voida määrittää.

3.3 Lyhyesti integraatiosovellusvalikoimasta

Tämän raportin tarkoituksena ei ole tehdä selkoa jokaisesta markkinoilla olevasta integraatiotuotteesta. Jo pelkästään nopealla haulla internetistä saa käsityksen laajasta valikoimasta. Seuraavaksi on esitelty muutamia niin kaupallisia kuin avoimiakin ratkaisuja tarjoavia sovellustoimittajia, joiden valikoimasta löytyy integraatiosovellus. Valikoimaa on tiivistetty ottamalla mukaan vain aktiivisesti kehitettäviä tuotteita ja nimenomaan ESB-toteutuksia. Kuten on tullut ilmi, jälkimmäiseen rajaus voi olla häilyvä, mutta esitellyt tuotteet ovat laajempia toteutuksia kuin *integration frameworkit* (integraatioita tukevat ja mahdollistavat ohjelmistokomponenttikirjastot) ja toisaalta suppeampia kuin laajat *integration suite*-kokonaisuudet (termi kääntyisi loogisesti integraatioalustaksi, mutta suomenkielistä termistöä ei voi aukottomasti vahvistaa).

Seuraava ”tuote-esittely” ei myöskään vastaa kysymykseen, mikä vaihtoehdoista on paras valinta. Valintaan vaikuttaisi luonnollisesti monta kriteeriä, ja näiden tunnistaminen liiketoiminnan tarpeista olisi isomman tutkimuksen kohde. Tuotteet tarjoavat vaihtelevan määrän toiminnallisuuksia, ja johtuen aktiivisesta kehitystyöstä, ne voivat myös päivittyä melkeinpä päivittäin saman tuotteen osalta. Näin on usein tietysti muidenkin IT-järjestelmien osalta.

3.3.1 Mulesoft

Kun lähtee tutustumaan ohjelmistointegraatioon ja sen sovelluksiin internetiä hyödyntäen, törmää varsin nopeasti Mulesoft-yritykseen. Se markkinoi aktiivisesti tuotteitaan ja vielä vuoden 2015 kevääseen saakka tarjosi ESB-ratkaisustaan community-editionia. Valitettavasti kyseinen versio oli poistunut ja tällä hetkellä Mulesoft tarjoaa vain kaupallisia versioita tuotteistaan. Keihäänkärki-tuotteenaan yritys tarjoaa Anypoint platformia, minkä Mulesoft toteaa olevan markkinoiden laajimmin käytetty työkalu integraatioihin. Varsinaista faktaa ei kuitenkaan tämän väitteen tueksi löydy.

Mulesoftin sivuilla on paljon järjestelmäintegraatiota koskevaa julkaisuja, joiden avulla aiheeseen on hyvä tutustua. Toki osa teksteistä on varsin markkinointisävytteisiä ja niiden on sen suhteen hyvä suhtautua pienellä varauksella. Mulesoft on kasvanut viimeisten vuosien aikana isohkoksi toimijaksi ja sen kehitysyhteisö on kohtalaisen vilkas. Vielä kaksi vuotta sitten Mule-yhteisön pientä kokoa pidettiin yhtenä heikkoutena sen ESB-ratkaisulle (Wähner 2013). Myös ohjeistus on varsin kattavaa MuleSoftin kehittäjäsivustolla, minkä vuoksi kyseinen toimittaja oli yhtenä ehdokkaana käytännön osuuden tutustumisalustaksi.

Tarjoamatta syvällisempää katsausta Mulesoftin tuotteiden ominaisuuksiin, Mule ESB:stä löytyy niin tuki tietomuunnoksille kuin 120 valmista konnektoria (tilanne keväällä 2015) laajasti käytössä oleville sovelluksille. Helppokäyttöisyyttä tarjoaa yhden yhtenäisen käyttöliittymän kautta tehtävä integraatiomääritysten luonti ja tietoliikenteen valvonta. Nämäkin toki jo varsin yleisiä muidenkin toimijoiden tuotteissa. (MuleSoft 2015a)

3.3.2 Talend

Kuten Mulesoft, tarjoaa Talend sivuillaan hyvän tietopaketin järjestelmäintegraatioista. Sen tuotevalikoima keskittyy lähes pelkästään järjestelmäintegraatioon ja ESB-tuotteesta on tarjolla myös community-versio. Monen muun ESB:n tavoin, viimeaikaisin kehitys on painottunut graafisen käyttöliittymän tehokkaa-

seen hyödyntämiseen integraatioita toteuttaessa. Tällainen käyttöliittymä voi madaltaa kynnystä vähemmän teknisorientoituneelta saada yksinkertaisia integraatioita toteutettua, mutta sen ei saa antaa hämätä liiaksi: mitä monimutkaisempia toteutuksia vaaditaan, sitä enemmän drag'n dropin avulla toteutettuja liittymiä joudutaan täydentämään raa'alla koodaamisella (Comparison of Open Source Software ESB Solutions 2010).

Talendin *Open Studio for Data Integration* on Apahce-lisensoitu (Talend 2015). Sen ydinominaisuuksista on tehty tarkempaa selkoa käytännön osuudessa ensimmäisen ja neljännen tapausesimerkin myötä.

3.3.3 Red Hat JBoss Fuse

Melkoisen monen omistajan kautta Fuse on tätä nykyä Red Hatin hallinnoima. Fuse on avoimen lähdekoodin ESB, jossa ei ole lainkaan lisenssimaksuja. Tämä seikka tekeekin siitä varsin houkuttelevan tuotteen tutustumistarkoitukseen.

Toisaalta, koska Fuse yhdistää useita eri teknologioita tarjotakseen toimivan integraatioalustan (mm. Apache Camel, Apache CFX, Apache ActiveMQ, Apache Karaf), sen käyttöönottoon vaadittava perehtyminen vie aikaa. Eduksi voi kuitenkin katsoa kyseisen teknologian levinneisyyden taaten sille hyvän kehittäjäverkoston ja sitä kautta kattavaa ohjeistusta. (Red Hat 2015)

3.3.4 Oracle Service Bus

Oracle, yhdessä Microsoftin kanssa, on järjestelmäintegraation isoja, kaupallisia toimittajia. Sen viimeisin tuote keskittyy tarjoamaan pilviteknologiaa tukeville sovelluksille järjestelmäintegraatioita ICS-tuotteellaan (Integration Cloud Service). Vanhempaa, SOA-periaatteita noudatteleva Service Bus perustuu useisiin Oraclen ostamiin standardeihin, mikä on johtanut lukuisan ohjelmointi-/luokkakirjaston taakkaan. Tämän ja kalliin lisensoinnin vuoksi tuote on suositeltavampi ammattilais- kuin harraste- ja kokeilukäyttöön. Jos nyt kukaan harrastemielessä järjestelmäintegraatioita on toteuttamassakaan. (Oracle 2015.)

Johtuen Oraclen kaupallisuudesta tarjoaa sekin oppimistarkoitukseen hyvää aineistoa muun muassa teknisten tutkimusraporttien ja esittelymateriaaleiden kautta. Näihin materiaaleihin on luonnollisesti osattava suhtautua riittävällä kriittisyydellä, aivan kuten muidenkin sovellustoimittajien tarjoamien materiaalien osalta.

3.3.5 Microsoft BizTalk ja Azure

BizTalk on laajasti käytössä suurilla yrityksillä. Se on kaupallinen ratkaisu, eikä ole luokiteltavissa puhtaasti pelkäksi ESB-ratkaisuksi. Muiden joukossa se on tässä listauksessa lähinnä sen levinneisyyden vuoksi. Microsoft tarjoaa myös nykyaikaista pilvipalveluiden integraatiopalvelua, minkä osalta yhtiön Azure-tuote tarjoaa uusinta teknologiaa palveluväylä-tuotteen (ts. ESB) muodossa. Hinnoittelu perustuu tapahtuneisiin siirto-transaktioihin toisin kuin BizTalkissa, joka on hinnoiteltu alustana perinteisempään SaaS-malliin (software as a service).

BizTalk-palvelinohjelmisto perustuu tekniikaltaan Microsoftin .NET-kehitysympäristöön sekä MS SQL Server –tietokantaan, ja lienee sitä kautta looginen valinta, mikäli yhdistettävät ohjelmistot on rakennettu samaan ympäristöön. Tutustuttaessa 2.1-luvussa esiteltyihin käytännön esimerkkeihin suomalaisyritysten toteuttamista integraatioista Microsoftin BizTalk-teknologia oli varsin yleinen alusta integraatioratkaisuissa. (Microsoft 2014.)

3.3.6 WSO2

WSO2 on MuleSoftin tapaan keskittynyt liiketoiminnassaan puhtaasti integraatioratkaisuiden kehittämiseen. Yritys on verrattain nuori, mutta on saanut haalittua referensseikseen varsin isoja nimiä (mm. eBay ja Cisco) ja sen tuotevalikoima käsittää laajan kirjon eri ratkaisuja integraatiotarpeisiin.

WSO2 tarjoaa ilmaiseksi ladattavan version ESB-tuotteestaan. Sen ominaisuudet vastaavat aika tavalla samaa laajuutta kuin alan muutkin edelläkävijät. Liitteistä löytyy listaus tuetuista protokollista ja tietoformaateista. Näistä saa hyvän käsityksen ESB:n toiminnasta teknisemmällä tasolla. (WSO2 2015.)

3.3.7 Ensemble ja Mirth

Viimeisenä maininta vielä kahdesta erikoistapauksesta terveydenhuoltoalan integraatio-ohjelmistoista. Mirth Connect tarjoaa avoimen ratkaisun ja Ensemble myy kaupallista työkalua. Toiminnallisuuksiltaan kumpikaan ohjelmisto ei merkittävästi eroa niin sanotuista yleiskäyttöisistä tuotteista: panostus terveydenhuoltoalan sovelluksien yhdistämiseen näkyy ennen kaikkea tietoformaattina, joka alalla on HL7 (Kansallinen Terveysarkisto 2015).

Yhteenvetona, nämä terveydenhuoltoalan integraatio-ohjelmistot oikeastaan korostavat, miten universaalista asiasta järjestelmäintegraatioissa onkaan kyse. Yhdistettävät ohjelmistot ja sanomamuodot ovat eroavia, mutta periaatteet hyvinkin yhteneviä.

4 TESTISOVELLUKSET JA –YMPÄRISTÖ

4.1 Testiympäristö

Raportin neljäs luku keskittyy kuvaamaan integraatio-ohjelmistoja käytännön sovelluksissa. Tällä on pyritty sekä syventämään edellä kuvattua teoriapohjaa että kerryttämään käytännön osaamista järjestelmäintegraatioihin liittyen. Ennen kuin testaamiseen ja syventävään opiskeluun ryhdyttiin suoritettiin perusteellinen pohdinta, millaisella testi- ja opiskeluympäristöllä tapausesimerkkejä konkreettisesti saataisiin parhaalla tavalla toistettua. Ideana oli tutustua integraatio-ohjelmistoihin kehittämällä tapausesimerkkejä, jotka voisivat olla nykypäivän järjestelmäintegraatioita hyödyntävissä yrityksissä mahdollisia.

Tätä nykyä yritykset toimialasta riippumatta käyttävät lukuisia erilaisia sovelluksia, joten itse yhdistettävillä sovelluksilla ei tässä ollut isoa merkitystä. Pääasia oli, että niistä saatiin luotua mahdollisimman hyvin siirrettävää aineistoa ja toisaalta luettua sitä sisään useilla eri tavoilla, esimerkiksi suoraan tietokantaan kirjoittamalla tai lukemalla niitä sovelluksen käyttöliittymän kautta eri tiedostomuodoin (esim. Excel-, csv- tai tekstitiedostoja).

Yhdistettäviksi sovelluksiksi valittiin muutama omalle työasemalle asennettava, avoin ja ilmainen ratkaisu. Tämä mahdollisti myös tietokantaan kirjoittamisen tarvittaessa. Näiden sovelluksien edellytyksenä oli mahdollisuus pyörittää PHP:tä tukevaa web-palvelinta ja yhteensopiva tietokanta-palvelin. Kevyin ratkaisu tähän tarkoitukseen löytyi MAMP:ista (My Apache – MySQL - PHP). Tämän asentaminen oli vaivaton prosessi. Lisäksi tietokantojen kanssa työskentelyä varten asennettiin erikseen vielä MySQLWorkbench. Tämän avulla SQL-komentoja sai testattua tehokkaammin kuin MAMPin kevyen selainpohjaisen ratkaisun kautta.

Työasemalle asennettiin FrontAccounting, SugarCRM ja GnuCash. Näistä kaksi ensimmäistä olivat niin sanottuja web-sovelluksia, jotka asentamisen jälkeen toimivat MAMPin avulla localhostilla www-selaimen kautta. Asennus oli suora-

viivainen: tarvittavat tiedostot kopioitiin web-palvelimen lukemaan tiedostosijaintiin, luotiin tarvittava tietokanta ja ajettiin sovelluksien oma Wizard-työkalu, joka mahdollisti nopeiden oletusasetusten syöttämisen. FrontAccounting ja GnuCash ovat hyvin pelkistettyjä taloushallinnon sovelluksia, jotka on tarkoitettu lähinnä Pohjois-Amerikan markkinoille. Tarkempaa analyysiä sovelluksien käytön laajuudesta ei tehty. SugarCRM on maksullinen asiakastietojärjestelmä, josta on tarjolla community-edition vapaaseen käyttöön. Erityistä PHP-osaamista ei tarvittu sovellusten asentamiseen tai muokkaamiseen. Lähinnä kyseeseen tuli ainoastaan muutaman asetustiedoston muokkaaminen tekstieditorissa.

Lisäksi internetistä etsittiin muutamaa aitoa pilvipohjaista ratkaisua. Näiden sovellusten testaaminen olisi kuitenkin vaatinut laajempaa sitoutumista muun muassa kehittäjäohjelmiin, jotta tarjottavia API-ratkaisuja olisi voinut hyödyntää omien web-serviceiden kautta. Tässä katsottiin kuitenkin jo opiskeltavan alueen laajuuden ylittyvän ja näinpä keskityttiin valittujen sovellusten valmiisiin, käyttöliittymien kautta tapahtuviin tiedostojen tuonti- ja vientitoiminnallisuuksiin. Näiltä osin siis integraatiot eivät olleet täysin automatisoituja, vaan vaativat hieman manuaalisia työvaiheita.

Testausympäristöistä ja sovellusten opiskelusta rajattiin pois muutamia merkittäviäkin asiakokonaisuuksia. Tähän päädyttiin huomattaessa raportin ja toisaalta perehtymiseen vaadittavan ajan kasvavan liian suureksi. Pois jäivät muun muassa:

- tietoturvakysymykset
- FTP/SFTP-tiedostosiirrot
- laajempi palvelinarkkitehtuuri (toimittiin localhostin avulla)
- loki-toiminnallisuudet integraatiotehtävissä (testausmielessä käytettiin kehitysympäristön trace-toimintoa tarvittaessa)
- virhehallinta ja poikkeukset

4.2 Valitut integraatio-ohjelmistot ja perehtymismetodit

Riittävän laajan käsityksen saamiseksi valittiin kolme eri integraatio-ohjelmistoa. Näiden valinnassa hyödynnettiin teoria-osuuteen tehtyä kartoitusta: valinnassa painotettiin mahdollisuutta käyttää sovellusta ilmaiseksi ja riittävää tukea ohjeistuksien muodossa. Testatut ja opiskellut ohjelmistot olivat Talend Open Studio for ESB, MuleSoftin ESB ja JBossin Fuse.

Talend Open Studio ja MuleSoftin MuleStudio olivat molemmat graafista käyttöliittymää hyödyntäviä järjestelmäintegraatio-kehitysympäristöjä. JBossin Fuse oli komentotulkin kautta ajettava ohjelmisto.

Integraatio-ohjelmistoiden käyttöön liittyvä opiskelu alkoi teoriapohjan rakentamisen jälkeen. Suomalaisessa tekniikan kirjallisuudessa ei ole aiheesta tarjontaa, joten englanninkieliset kirjat, internet, Youtube ja erityisesti sovellustoimittajien omat sivustot tarjosivat parhaan opinlähteen. Riittävän kokonaiskuvan saamiseksi luettiin alkuun sovelluskohtaisesti perusteos. Näistä sai hyvin käsityksen perustoiminnoista, joita sitten tuli testattua (tarvittaessa Youtube- tai webinar-oppaiden avustuksella) aidossa ympäristössä. Huolimatta siitä, kuinka paljon asioista lukee manuaaleista ja kokeilee itse vaihe vaiheelta eri toimintoja, haastavimmat toteutukset vaativat useita erehtymisen ja onnistumisen yhdistelmiä. Kokeilla voi toki rajattomasti, mutta erilaisten yhdistelmien vaikutusten selvittäminen ja samanaikainen lukuisien virheilmoitusten ilmaantuminen alkaa jossain kohden turhauttaa ja syvällisempi perehtyminen käy yhä työläämmäksi.

Harjoitusten teko kuitenkin opettaa ja antaa itsevarmuutta. Kun harjoituksia soveltaa käytäntöön, syventyy osaaminen entisestään. Toisaalta syventävien sovellusten laadinta vaatisi huomattavasti enemmän määrittelyä, suunnittelua ja ennen kaikkea aikaa. Sovelluksiin tutustuessa kuitenkin teki havainnon, mikä pätee muun muassa toimistotyöläisten excel-osaamiseen; sovellusta saa käytettyä auttavasti ja tarpeeseen, vaikka sovelluksen toiminnallisuuksista osaisi vain alle 10%. Perusasiat oppii nopeasti, esimerkiksi ohjelman logiikan. Syvemmälle menevistä ominaisuuksista oppii nopeasti Java-pohjaisen koodauk-

sen, mutta sovelluksien niin sanotun oman kielen oppiminen vaatisi syvempää tutkintaa esimerkiksi keskustelupalstoja selaamalla. Tämä taas kääntäen vaatisi joka tapauksessa monimutkaisempia sovellutuksia.

4.3 Ensimmäinen tapausesimerkki - Talend

Ensimmäisellä tutustumisella Talendin ja Mulesoftin integraatio-ohjelmistot vaikuttivat samanlaisilta; Talendin osalta löytyi kuitenkin helpommin lähestyttävää ohjeistusta, joten se valikoitui ensimmäiseksi sovellukseksi käytännön osuudessa. Sovelluksen opiskelusta tuli luonnollisesti konkreettisempaa (ja antoisampaakin), mikäli oli selkeä käytännön tavoite, jolla sovelluksen opiskelua pystyi tavoitteellistamaan. Tähän tarkoitukseen oli etsitty jo edellä esitellyt muutamat taloushallinnon ja asiakashallinnan sovellukset, joiden avulla luotiin integraatiolle tarve ja luonnosteltiin se tapausesimerkiksi. Tämän laadinnassa käytettiin lopulta vain mielikuvitusta.

Kokonaiskuva toteutettavasta integraatiosta oli seuraava: tavoitteena oli hakea joukko tietyin ehdoin rajattuja yrityksiä internetistä, lukea ne sisään sekä asiakashallinnan ja taloushallinnon järjestelmiin. Koska ymmärrettiin että molempien järjestelmien sisäänlukema aineisto olisi eri muotoista, integraatio-ohjelmistolle löytyisi selkeät käytännön tehtävät.

Konkreettisemmalla tasolla yritykset noudettiin YTJ:n avoimen rajapinnan kautta, muunnettiin sieltä noudettu JSON-sanoma eri sovelluksien lukemaan muotoon. Enimmäkseen sisäänluettu muoto valituissa sovelluksissa oli CSV-muotoista eli niin sanottua taulukkomuotoista dataa tekstitiedostona. Lisäksi tietoa kirjoitettiin suoraan sovelluksen tietokantaan Talendin toimesta. Kyseinen toimintojen sarja voisi tulla kyseeseen esimerkiksi tilanteessa, jossa yritys hakee kontaktoitavia asiakasyrityksiä (ns. liidejä), joille voisi myydä palveluitaan. Tätä tukevia järjestelmiä ovat luonnollisesti muun muassa asiakashallinnan järjestelmä, sekä taloushallinnon sovellukset esimerkiksi myyntilaskutusta varten.

Talendin opiskelussa hyödynnettiin sovellustoimittajan omia oppaita ja videotutoriaaleja, sekä kolmansien osapuolien tuottamia youtube-opastuksia. Näistä kuitenkin kokonaiskuvan hahmottaminen oli alussa hankalaa. Erityisen hyvän tuen oppimiselle tarjosi Getting Started with Talend Open Studio for Data Integration –teos. Se sisälsi käytännön esimerkkejä riittävän seikkaperäisesti selostettuna. Lopulta kuitenkin varsinainen oppiminen tapahtui ennen kaikkea aidossa ympäristössä harjoitellen, kokeillen ja toteuttaen tavoitteeksi asetettua integraatio-työtä. Tässä luvussa on seuraavaksi kuvattu vaihe vaiheelta työn laadinta Talendin sovellusympäristössä. Iso osa kuvamateriaalista on selkeyden vuoksi lisätty raportin loppuun liite-osioon.

4.3.1 Projektin luonti Talendissa

Ensimmäisenä tehtävänä oli luoda Talendiin projekti, jonka alle oli mahdollista kerätä niin sanottuja jobeja (tässä raportissa on päätetty jättää suomentamatta ohjelmistojen termit ja toiminnallisuudet). Talendin työskentelynäkymä vastaa aika tavalla tavallista ohjelmointikehitysympäristöä (Eclipse, NetBeans, jne.), missä ohjelman toiminnallisuudet on jaettu erilaisiin ikkunoihin. Näissä ikkunoissa tai moduuleissa sijaitsevat Talendissa muun muassa repository (jobien ja metadatan hallintaa varten läpi projektin), paikallisten muuttujien kirjasto Outline-ikkunassa, varsinainen ”työmaa” eli Design workspace, jossa käytännössä integraatio-job määritellään komponenttien ja niitä yhdistävien prosessivirtojen myötä.

Lisäksi kehitysympäristöstä löytyvät Palette-ikkuna, jossa sijaitsevat käytännössä kaikki Talendin oletuskomponentit ja -työkalut (Talendin mukaan valmiita komponentteja on yli 600). Näkymän alalaidasta löytyvät asetus-välilehdet, joissa päästään tekemään määrityksiä jokaiselle työssä käytettävälle komponentilla erikseen. Liitteessä 2 on kuvattu kehitysympäristö, ja siitä näkyy vasemman laidan Palette-ikkunassa esimerkinomaisesti Talendissa osa hyödynnettävistä tiedon prosessointi komponenteista (liite 2).

Tapausesimerkin luonti lähtee liikkeelle projektin lisäämisestä. Projektin alle luotiin ensimmäinen job, joka taas edelleen koostui myöhemmin tehtävistä subjobeista ja näiden muodostamista tietovirtoja käsittelevistä komponenteista. Käytännössä nämä palaset ovat integraatio-ohjelmistossa rakennettavan integraation ”moottorin” osasia: niillä käsitellään käytännössä tietoa eri tavoin integraatio-ohjelmiston toiminnallisuuksien mukaisesti. Integraatiotyön rakentaminen koostuikin yksinkertaistettuna näiden palasten yhdistämisestä ja määrittelemisestä.

4.3.2 Toiminnallisuuksien määrittely Talendissa

Tapausesimerkin ensimmäinen tehtävä oli noutaa sopiva yrityslista internetistä. Tämä toteutettiin PRH:n tarjoaman avoimen YTJ-rajanpinnan kautta (osoitteesta http://avoindata.prh.fi/ytj.html). Jotta yrityslistasta saatiin rajallinen, annettiin tarkempina määrittelyinä hakea vain ohjelmistoyrityksiä (tilastokeskuksen toimialaluokittelulla 62). HTTP-requestin URLiksi muodostui:

http://avoindata.prh.fi:80/bis/v1?totalResults=false&maxResults=10&resultsFrom=0&businessLineCode=62

Tämä listasi 10 viimeksi perustettua ohjelmistoalan yritystä JSON-muodossa. Tieto ei sisältänyt yritysten osoitetietoja, joita tarvittiin tarkentavina tietoina asiakasrekisteriä varten. Näiden tietojen noutamista varten tehtiin toinen URL-request, minkä hakemat tiedot myöhemmin yhdistettiin Talendissa omassa komponentissaan. Molempien GET-komentojen hakemat JSON-yritystiedot tallennettiin Talendin komponentissa erillisiin tiedostoihin, joista niiden tietoja muokattiin CSV-muotoon. Tietojen noutamiseen käytettiin Talendin tHttpRequest-komponenttia.

Osoitetietoja varten tehtiin Talendissa iteraatio-prosessi, jossa haettiin rivi riviltä yrityksen y-tunnusta vastaavat osoitetiedot. Näistä poimittiin myöhemmin kaupunki-tieto täydennykseksi varsinaiseen yrityslistaan. Iteraation toteuttaminen oli haasteellista, ja kävikin selväksi, ettei tHttpRequest-komponentti sellaisenaan tukenut tämänkaltaista toiminnallisuutta. Lähinnä ongelmaksi muodostui

noudettavan JSON-muotoisen aineiston kerryttäminen samaan tiedostoon. Tämän osalta päädyttiin keräämään tiedot käsin erilliseen listaukseen, josta niitä voitiin hyödyntää myöhemmin. Iterointi kuitenkin tehtiin, ja se toteutettiin POC-tyylisesti (proof of concept): toteutus toimi sillä tasolla, että y-tunnuksia vastaavien yritysten tarkemmat tiedot saatiin listattua Talendin loki-toiminnolla (tRowLog-komponentti).

Tarkempia osoitetietoja noutavassa alijobissa hyödynnettiin paikallista muuttujaa, johon tallennettiin y-tunnus, mitä sitten edelleen hyödynnettiin tHttpRequestin URI:ssa. Asetukset on kuvattu liitteessä 3. Paikallisen muuttujan sai haettua kehitysympäristössä näppärästi ctrl-välilyönti -komennolla. Itse iterointi toteutettiin hyödyntämällä Talendin tFlowToIterate- (kävi rivi riviltä läpi yrityslistaa) ja tForeach-komponentteja (poimimaan y-tunnuksen ja välittämään sen eteenpäin httprequest-komponentille). Ennen iteraatiota ja subjobin suorittamista täytyi kuitenkin tallentaa haettu yrityslista JSON-muodosta puolipisteillä erotelluksi CSV-tiedostoksi.

JSON-muotoisen tietosyötteen lukeminen CSV-tiedostoon aloitettiin luomalla niin kutsuttu metadata JSON-schemalle, jota käytettiin pohjana tiedoston sisäänlukua varten. Metadatan luonti oli varsin yleinen työvaihe Talendin päivittäisessä käytössä. Pohjan sai luotua vaivattomasti hyödyntämällä tHttpRequestin noutamaa JSONia (syöte oli ensin tallennettu manuaalisesti tiedostoon selainta ja tekstieditoria hyödyntäen). Tämä vaati vain muutaman määrittelyn sisältäen muun muassa tietojen valinnan ja sen mitä tietoa ”loopattiin” JSONin ollessa rakenteellista tietoa. Yritystieto-syötteestä haluttiin erotella y-tunnus, yrityksen nimi, yhtiömuoto ja detailsuri (kuvattu liitteessä 4).

Tämän jälkeen JSON-muotoinen data luettiin csv-tiedostoon, josta varsinainen tiedon käsittely (rikastaminen, ”mäppääminen” ja reitittäminen) aloitettiin. Tätä varten lisättiin Talendissa uusi komponentti tFileOutputDelimited, mikä yhdistettiin linkillä edelliseen komponenttiin. Sen määrittely oli helppoa, koska voitiin hyödyntää JSONin syöttämää tietovirtaa (esim. ennalta määritellyt sarakkeet).

Tämän subjobin käynnistyminen edellytti http-komennon onnistumista, Talendissa tämä määriteltiin linkillä ’on subjob ok’.

Kun kaikkien yrityksien kaupunki-tiedot oli saatu erilliseen tiedostoon, rikastettiin alkuperäistä yrityslistausta haetuilla tiedoilla Talendin tJoin-työkalun avulla. Tämä toteutettiin luomalla ensin rikastavalle tiedostolle oma skeemansa ja yhdistämällä se (ja alkuperäinen tietovirta) tJoin-komponenttiin. Näiden lopputuloksena oli yhdellä sarakkeella laajentunut yrityslistaus. Monimutkaisin vaihe oli määritellä tJoin-komponentin asetuksista oikeat, yhdistettävät tiedot (vaihe on kuvattu liitteessä 5). Tätäkin helpotti johdonmukaisesti nimetyt sarakkeet skeemojen määrittelyssä. Samassa yhteydessä, tjoin-skeeman asetuksissa jätettiin yrityslistasta kaupunkien hakuun käytetty ’detailsuri’-sarake pois (liite 5).

Tässä vaiheessa hyödynnettävissä oli siis yrityslista, joka sisälsi sarakkeet: y-tunnus, yrityksen nimi ja kaupunki. Seuraava vaihe oli täydentää listausta kaupunkia vastaavilla myyjillä. Tätä varten siirryttiin tekemään ensin määrityksiä SugarCRM:ään.

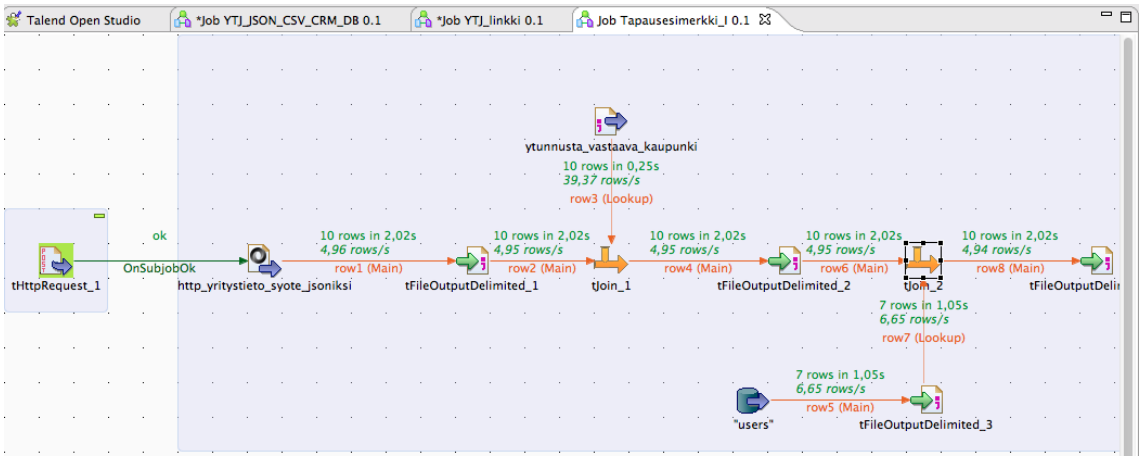
4.3.3 Asetusten määrittäminen SugarCRM-asiakastietojärjestelmässä

Sugarin pyörittämiseksi täytyi ensin käynnistää MAMPin avulla localhostina toimiva web-server. MAMPin avulla saatiin myös tiedot Talendin tietokantayhteyttä varten: tietokantayhteyteenkin tehtiin oma skeemansa, jotta tietoja voisi tarvittaessa käyttää kätevästi uudelleen.

Ensimmäisenä SugarCRM:ään lisättiin käyttöliittymän kautta käyttäjät, joille määriteltiin vastuu-alueet tallentamalla myyjän käyttäjätietoihin department-kenttään kaupunki. SugarCRM:n käyttöä varten ei tutustuttu ohjeisiin, vaan pitäydyttiin insinöörimallisessa ”kokeile, erehdy, korjaa ja kokeile uudestaan” -käytännössä. Kun käyttäjät/myyjät oli tallennettu Sugariin, tehtiin tietokantaan hakuja SQL-kyselyin oikean taulun ja sitä kautta oikeiden kenttien löytämiseksi. MySQLWorkbenchissä oli myös mahdollista selailla suoraan taulujen rakennet-

ta ja sisältöä. Taulun rakenteen ja sisältöjen tuntemuksesta oli apua myöhemmin Talendissa tietokanta-yhteyttä määrittäessä.

Koska tietokantayhteyttä varten oli aiemmin luotu valmis pohja, saatiin sen avulla luettua myös taulut omiksi metadata-komponenteiksi. Tästä otettiin hyödyksi Sugarin taulu ’users’, minkä metadata-komponentin kyselyä muokattiin noutamaan vain olennaiset tiedot: user_name, first_name, last_name ja department (tMySQLInput-komponentin asetukset on kuvattu liitteessä 6). Seuraavaksi haettu myyjätieto täytyi yhdistää alkuperäiseen, kaupunkitiedolla rikastettuun yrityslistaan (departmentin perusteella). Tämän jälkeen oli siis hyödynnettävissä listaus yrityksistä, jossa jokaista kuviteltua asiakasyritystä kohden oli kaupungin vastuumyyjä, jos alueelle oli sellainen nimetty. Tässä vaiheessa job oli kuviossa 3 esitetyn kaltainen. Siitä on nähtävissä Talendin logiikka erotella subjobit (harmaalla pohjalla).



Kuvio 3: Talend job

Seuraava vaihe oli muuntaa olemassa oleva yrityslista SugarCRM:n luettavaksi (asiakkaiksi asiakasjärjestelmään), GnuCashin (asiakkaiksi kuvitteelliseen laskutusohjelmaan) sekä FrontAccountingin (toimittajaksi erilliseen taloushallintosovellukseen) luettavaksi. Ylätasolla ideana oli siis mallintaa tilannetta, jossa yritys hakee mahdollisia niin sanottuja liidejä myynnin tueksi CRM-ohjelmaansa, missä ne kohdistuvat vastuumyyjille esimerkiksi puhelimitse kontaktoitaviksi. Toisaalta samat yritykset voivat jatkossa toimia mahdollisina asiakkaina, ja näin

niitä on voitava laskuttaa; kuvitteellisella yrityksellä olisi tässä tapauksessa käytössään GnuCash kirjanpitoaan varten.

FrontAccount tässä simulaatiossa mallintaisi hyvinkin usein ilmenevää esimerkkiä, jossa yrityksillä on erilaisia sovelluksia käytössään niiden toiminnallisuuksien perusteella: esimerkiksi yksittäinen taloushallinnon sovellus voi toimia paremmin myyntilaskutuksessa, kun taas toinen sovellus voi palvella paremmin ostolaskujen käsittelyssä. Tämän skenaarion valossa FrontAccountissa samat asiakasyritykset toimisivat mahdollisina palveluiden toimittajina, malliyrityksen toimiessa näin ollen vuorovaikutteisessa toimintaympäristössä.

SugarCRM:n käyttäjätietojen sisäänluku käyttöliittymän kautta tapahtui mallitiedoston avulla. Sama tiedosto kertoi myös tietokantataulun kenttien nimet, ja tätä tietoa olisi voitu hyödyntää tietokantaan kirjoittamisen yhteydessä. Mallitiedosto osoittautui muodoltaan poikkeavaksi tähän saakka käsitellyn yrityslistan kanssa (puolipisteillä eroteltu taulukkomuotoinen tekstitiedosto). Ylipäätään, jotta mallitiedosto saatiin järjellisesti luettavaan muotoon, jouduttiin käyttämään Microsoft Excelin tietojen sisäänluku tekstistä -toimintoa. Tätä kautta mallitiedosto oli helpommin tulkittavissa: kenttien erottimena toimi pilkku ja kaiken lisäksi tekstin lisätarkenteena olivat lainausmerkit.

Tavanomaisesta hieman poikkeava sisäänlukumuoto muun muassa erotinmerkin tapauksessa toi eteen toisaalta hienon mahdollisuuden syventyä Talendin toiminnallisuuksiin, mutta erityisesti se tarjosi erittäin käytännöllisen esimerkin integraatio-ohjelmistojen hyödyistä ylipäätään; juuri tämänkaltaisten muunnoksien kanssa erityiset ohjelmistot ovat vahvoilla. Toki asetukset saatiin metadatan muodostuksessa määritettyä varsin kätevästi, mikä toisaalta kertoi integraatio-ohjelmistojen (Talendin, tässä tapauksessa) käytettävyydestä. Erotinmerkkien lisäksi metadataa määrittäessä otettiin huomioon eurooppalaisesta poikkeava päivämäärän merkintämuoto.

4.3.4 Tietovirtojen reitittäminen ja rikastaminen Talendissa

Talendin ’Replicate’-komponentilla tietovirta monistettiin, jotta sitä voitiin hyödyntää sekä SugarCRM:ään, GnuCashiin että FrontAccountingiin menevässä liikenteessä. Tiedon muuntamiseksi kutakin sisäänlukevaa järjestelmää hyödyntäväksi käytettiin ’tMap’-toimintoa, joka ohjeita ja tutoriaaleja tulkiten on yksi käytetyimmistä toiminnoista. Käytännössä sen avulla on mahdollista niin rikastaa kuin ”mäpätä” esimerkiksi sarakkeita lähdetauluista kohdetauluihin tai suodattaa rivejä ja muokata tietoja Java-koodilla. tMap-komponentin toiminnan säätäminen toteutetaan Map-editorissa, jota muiden Talendin työkalujen tapaan käytetään selkeän graafisen käyttöliittymän kautta. Tässä vaiheessa tMap-komponentin lukuisia ominaisuuksia ei hyödynnetty, vaan pyrittiin tekemään ainoastaan yksinkertainen mäppäys olemassa olevasta, rikastetusta yrityslistasta SugarCRM:n lukemaan muotoon. Liitteessä 7 on kuva Map-editorista ja siitä miten tiedot yrityslistasta mäpättiin SugariCRM:ään.

Kun tiedosto oli tällä tavoin saatu muodostettua, siirryttiin se lukemaan sovelluksen käyttöliittymän kautta sisään. Tämä tapahtui SugarCRM:n ’Import Accounts’ –toiminnolla. Tässä vaiheessa oli mahdollista puuttua CRM-sovelluksen kautta sisäänluettaviin kenttiin. Nämä jätettiin kuitenkin ennalleen, jotta varmistuttiin, että sisäänluku toimisi sellaisenaan Talendin syöttämässä muodossa. Onnistuneen sisäänluvun jälkeen SugarCRM kertoi asiakasyritysten luodun. Varmistus myös vastuumyyjien nimeämisestä näkyi samassa ilmoituksessa; tosin kaupungille, jolle ei oltu määritelty myyjää, valikoitui Sugarissa automaattisesti järjestelmävalvoja. Asiakasyritys id:n osalta täytyi varmistaa tietokannasta käsin, jotta haluttu y-tunnus oli löytynyt oikeaan kenttään. Tämä tehtiin yksinkertaisella kyselyllä:

```
use sugarcrm_test;
select id, name from accounts
where date_entered between ('2015-12-08') and ('2015-12-09') ;
```

GnuCashin tarvitsema tieto selvisi myös sovelluksen oman käyttöliittymän kautta (’Import Customers and vendors’). Toisin kuin SugarCRM:ssä GnuCash ei

tarjonnut varsinaista mallipohjaa, vaan se laadittiin tekemällä itse yksinkertainen, puolipistein eroteltu mallipohja tekstieditorissa. Tätä mallipohjaa hyödynnettiin samalla tapaa Talendissa kuin SugarCRM:ää varten tehtyä Metadata-määrittelyä.

Koska GnuCashiin ei tarvinnut lukea sisään tietoa myyjistä, jätettiin se tMap-komponentin asetuksista mäppäämättä. Tietoa kuitenkin rikastettiin lisäämällä GnuCashiin menevään asiakaslistaukseen yhtiötunnus (nimeltään ”Lokayritys”, joka oli aikaisemmin luotu yhtiöksi sovelluksessa), jotta se kohdistuisi oikealle yhtiölle sisäänluettaessa, sekä notes-kenttään tallennettiin tekstinä tieto, milloin asiakas oli lisätty sisäänlukupohjaan. tMap-komponentissa Talendissa on erillinen ’Expression Builder’, jossa tämänkaltaisia toiminnallisuuksia on kätevä tehdä. Toisaalta samat voi tehdä suoraan sarakemäärityksissä Java-koodilla. Tekstin lisäämiseen riitti komento: ”tuotu ” + TalendDate.getCurrentDate. Olio-ohjelmoinnin periaatteiden mukaisesti komennossa hyödynnettiin Talendin valmiita luokkia ja sen metodeja. Lopulta asiakkaat luettiin sisään GnuCashiin käyttöliittymän kautta.

Tässä vaiheessa pääjobin kasvettua laajemmaksi huomattiin alkuperäisen tHttpRequest-komponentin hakevan yrityksiä YTJ:n avoimen rajapinnan kautta ilman päivämäärärajauksia. Tämä aiheutti sen, että yrityslista vaihtui kuukausittain alkuperäisestä. Toisin sanoen noudetut yritykset eivät olleet alkuperäisiä. Tämän ei annettu haitata, koska päällimmäisenä tavoitteena oli oppia Talendin toimintaa ja erityisesti sen komponenttien toiminnallisuuksia yhdessä laajassa jobissa, eikä niinkään rakentaa sovellutusta hyödynnettäväksi aidossa ympäristössä. Ongelman olisi voinut kiertää määrittelemällä http:n GET-komennon URI:in rajattu päivämääräväli.

Lopuksi toteutettiin vielä liittymä FrontAccount-sovellukseen, johon käsitelty yrityslista kirjoitettiin suoraan tietokantaan. Tässä ei noudatettu FrontAccountingin omaa, suositeltua tapaa, joka ei olisi olennaisesti poikennut edellä toteutetuista SugarCRM- ja GnuCash-liittymistä: FrontAccounting ohjeisti tallentamaan tiedot ensin tiedostoon, mistä ne oltaisiin kirjoitettu välitaulun kautta sisään varsinai-

seen tietokantaan (FrontAccounting 2015). Talendin kannalta tämän kaltainen operaatio ei olisi eronnut jo tehdyistä, ja koska haluttiin ennen kaikkea oppimismielessä testata eri toimintoja, päädyttiin kokeilemaan hieman riskialttiimpaa ratkaisua, ja kirjoittamaan tiedot suoraan tietokantaan.

Varsinaisesti edellä kuvattu ratkaisu ei ole suositeltava johtuen muun muassa mahdollisista virhetilanteista, joissa tietokanta-komentojen peruuttaminen voi olla hankalaa ilman erityisen hyvin suunniteltuja tietokantavarmistuksia palautusratkaisuineen. Näistä johtuen sovellukset tarjoavatkin nykyään hyvin usein tarkoin dokumentoidun ja helposti hyödynnettävissä olevan API:n, missä avoimen rajapinnan kautta tietoa voi sekä hakea että sitä voi tallentaa http-protokollan GET- ja POST-metodeilla (Bowen 2012, 113). Koska sovellus oli yksinomaan harjoitusta varten, eikä mitään tuotannolle kriittistä sijainnut tietokannassa, voitiin tietokanta-käskyjä ajaa kuitenkin lähes huoletta harjoituksenomaisesti.

Tietokanta-osion toteutus aloitettiin luomalla metadata tietokanta-yhteydelle. Tämän toteutus eteni samalla tapaa kuin aiemmin luotu SugarCRM-tietokantayhteys (minkä avulla noudettiin aiemmin SugarCRM:n tietokannasta myyjätiedot). Tietokanta-yhteyden määrittämisen jälkeen Talendin ’retrieve schema’ –komennolla oli mahdollista hakea tarkemmat tiedot tietokannan sisällöstä Talendin kehitysympäristön käytettäväksi. Tässä tapauksessa noudettiin pelkästään suppliers-taulun sisältö. Se poimittiin jobiin ja valittiin komponentin tyypiksi ’tMySQLOutput’ (toisin kuin tietoja haettaessa SugarCRM:stä komponentti oli Input-tyyppinen).

Työläänä vaiheena tämän jälkeen oli selvittää FrontAccountingin tietokannasta, mitä suppliers-taulun tiedoista oli pakollisia ja mitkä niistä edelleen olivat määriteltyjä muissa tauluissa. Tätä selvitystyötä tehtiin selailemalla tauluja MySQL-Workbenchissä. Koska vietävää tietoa tuli joka tapauksessa rikastaa pakollisten kenttien osalta, yhdistettiin SQL-komponentti tMap-komponenttiin, joka tarjosi tähän tehtävään hyvän työkalun.

Ennen varsinaista tiedon määrittelyä, säädettiin tietoa kirjoittavan SQL-komponentin asetuksia niiltä osin kuin oli tarpeellista:

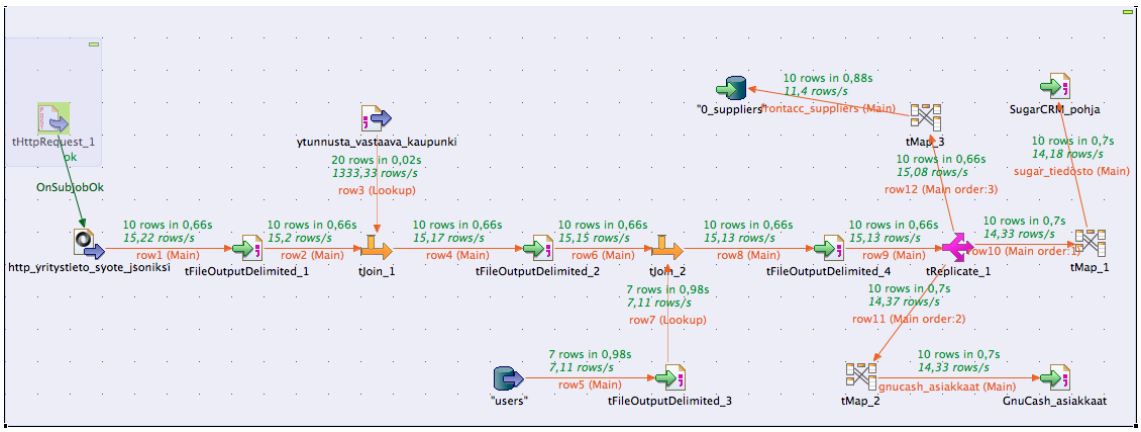
- *action on table: none* (tämä määritteli, ettei kirjoitettavaan tauluun tehtäisi muutoksia)
- *action on data: insert* (tämä määritteli, kirjoitettavan tiedon osalta, että vain lisättävät rivit kirjoitettiin ja jos tuplatietueita olisi ollut, ne olisi jätetty huomioitta)

tMap-komponentin editorissa asetettiin kohdetaulun vaatimat pakolliset tietosisällöt. Supplier_id-kenttään luotiin juokseva numerointi koodilla ”Numeric.sequence(”s1”, 4, 1)”, tämä siitä johtuen että FrontAccountissa sijaitsi jo kolme esimerkkitoimittajaa (numeroilla 1-3). Toki tähän olisi täytynyt tehdä kyselyllä haku, joka palauttaisi viimeisimmän toimittajanumeron ja asettaisi tästä seuraavan aina juoksevan numeron ensimmäiseksi parametriksi. Lähdetiedoista y-tunnusta hyödynnettiin notes-kentässä, ja loogisesti nimi ja osoitetiedot asetettiin niitä vastaaviin kohdetaulun kenttiin. Koska tietokanta ei hyväksynyt tyhjiä tietoja, täytettiin osa kentistä syöttämällä vain lainausmerkit.

Kun viimeiselläkin tehtävällä ryyditetty jobi käynnistettiin, saatiin Talendissa kuittaus tietojen läpimenosta. Tiedot oli onnistuneesti luettu sisään FrontAccountingin tietokantaan, mikä varmistettiin vielä sekä käyttöliittymästä käsin että tietokantakyselyllä. Kaikki halutut arvot olivat löytäneet omille paikoilleen.

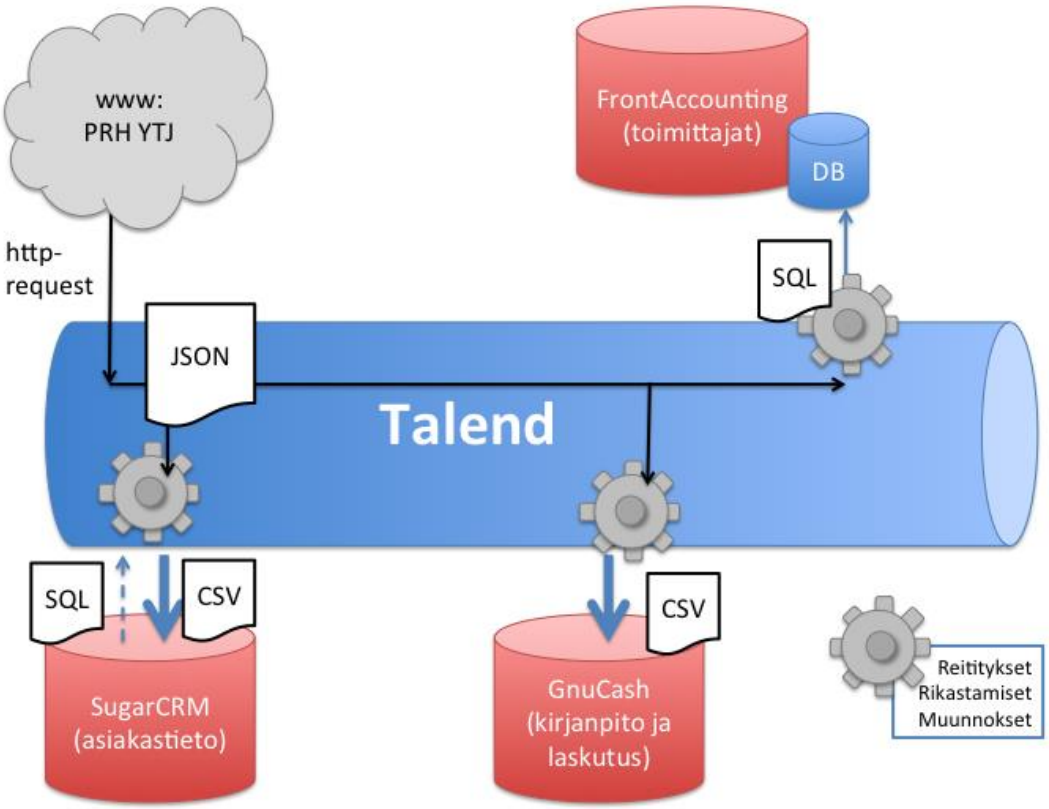
4.3.5 Ensimmäinen tapausesimerkki – yhteenveto

Kokonaisuudessaan jobi oli kuvion 4 kaltainen. Jokaisen jobin sisältämän komponentin asetukset oli erikseen määritelty ja osassa niiden luonnissa hyödynnettiin erikseen luotuja metadata-määrityksiä. Kaikkiaan jobissa saatiin hyödynnettyä laajasti Talendin eri toiminnallisuuksia. Käytännössä kyseisenkaltainen toteutus tehtäisiin hyödyntämällä erillisiä jobeja (kuten tässä tapauksessa tehty erillinen alijobi kaupunkitiedon hakua varten iteroimalla).



Kuvio 4: Tapausesimerkki I ja luotu job kokonaisuudessaan

Ylätasolla edellä kuvattu tehtävien kokonaisuus hahmottui hieman yksinkertaisempana, mikäli sitä peilasi teoriaosuudessa esitetyn hahmotuksen muodossa (kuviossa 5):



Kuvio 5: Ensimmäisen tapausesimerkin havainnollistus

Talendissa toteutettu integraatio-työ opetti ohjelmiston käytöstä hyvin, ja vähintäänkin alkeet tuli omaksuttua kiitettävästi. Osa komponenttien käytöstä tuli tutuksi ja niiden asetuksien määrittelyssä oli hyvä turvautua keskustelufoorumeihin, joissa huomasi monen muunkin painineen samojen ongelmien parissa. Toisaalta keskusteluita lukiessa huomasi myös, kuinka monimutkaisia sovellutuksia Talendilla on mahdollista toteuttaa. Kehitysympäristö on graafinen työkalu, mutta niin sanotusti konepellin alla on mahdollisuuksia virittää asioita (mikäli vain riittävää on perehtyneisyyttä). Esimerkiksi edellä kuvatun isomman jobin koodinäkymässä Talendissa oli yli 9000 riviä ohjelmiston generoimaa, toimivaa Java-lähdekoodia. Tässä vaiheessa koodiin ei tehty tarkempaa tutustumista pois lukien niitä tilanteita, joissa virhetilanteita yritettiin selvitellä koodi-tasolla.

Ainoa kokonaisuus, jonka poisjäänti Talendin harjoittelusta jätti pohdittavaa oli xml-tiedostojen käsittely: ne ovat nykypäivänä hyvin yleinen tiedostomuoto, ja moni avoimia API-rajapintoja hyödyntävä perustuu niitä käsitteleviin webserviceihin. Toisaalta http-protokollan GET-komento ja JSON-tiedostomuoto ovat myös yleisiä, ja niitä ensimmäisessä tapausesimerkissä hyödynnettiin. Ylipäätään webserviceihin pohjautunut integraatio olisi vaatinut hieman järeämpiä asennuksia liittyen testisovelluksiin, ja kuten todettua tämä rajattiin käsiteltävän laajuuden ulkopuolelle.

4.4 Toinen tapausesimerkki – MuleStudio

Talendin Open Studioon tutustumisen jälkeen vaihdettiin integraatio-ohjelmistoa. Vaikka alkuperäinen ajatus oli hyödyntää edellisessäkin tapausesimerkissä käytettyjä yhdistettäviä sovelluksia, kävi muutaman syyn vuoksi ilmeiseksi, ettei samaista testausympäristöä olisi tarkoituksenmukaista käyttää:

- Mulesoftin Mulestudion ”integraatio-logiikka” oli hieman erilainen siinä mielessä, että integraatiot perustuivat vahvemmin viestipohjaisuuteen (eikä tiedostojen välittämiseen, johon edellisen tapausesimerkin käsittelyssä tehty testausympäristö oli rakennettu), ja tästä syystä testausympäristö olisi täytynyt rakentaa web-pohjaisten sovellusten

vuorovaikutukseen esimerkiksi REST-teknologiaa ja web-serviceitä hyödyntäen, ja tämä taas olisi toisaalta vaatinut huomattavan lisäpanostuksen perehtyä ylipäätään web-ohjelmointiin.

- Koska edellisessä tapausesimerkissä oli tehty tiedostopohjaista integraatiota, samaisen harjoituksen tekeminen hieman varioiden ei palvellut oppimismielessä.

Edellä mainituista syistä johtuen toisen tapausesimerkin lähtökohtia hieman muutettiin: tavoitteena oli tehdä tiivis harjoitelma, jolla pääsi riittävän syvällisesti perille MuleStudion käytöstä ja logiikasta. Perehtyminen tapahtui yhden kirjallisen oppaan myötä (Mule ESB Cookbook) sekä tukeutuen vahvasti MuleSoftin tarjoamaan ohjeistukseen. Jälkimmäinen ohjeistus oli jäsennelty varsin käyttäjäystävällisesti: yleisopastus oli jaettu pikaoppaaseen (First 30 minutes with Mule), hieman syventävään peruskäytön opastukseen (First day with Mule) ja lopulta peruskäytön yksityiskohtaisempaan esittelyyn (First week with Mule).

Konkreettisissa harjoituksissa oli käytössä MuleSoftin sivuilta 2015 keväällä ladattavissa ollut avoin MuleStudio (versio 3.5). Myöhemmin kyseistä sovellusta ei ollut tarjolla avoimena vaan kaupallisesta versiosta MuleSoft tarjoaa nykyisin muutaman viikon trial-versiota. ”Vanhentunut” versio ei sinällään häirinnyt harjoittelua, lähinnä tämä näkyi muutamien ominaisuuksien ja toiminnallisuuksien pieninä muutoksina.

4.4.1 Projektin luonti MuleStudiossa

Ulkoisesti MuleStudion käyttöliittymä muistuttaa huomattavan paljon Talendin Open Studion vastaavaa. Molemmat pohjautuvat Eclipse-kehitysympäristöön. Käyttöönotto ja uuden projektin perustaminen toimivat samalla tapaa. MuleStudiossa integraatio-työn aloittaminen tapahtuu avaamalla uuden projektin ja lisäämällä siihen ensimmäinen ”Mule Flow”. Käytännössä Flow MuleStudiossa vastaa Talendin Jobia. Edellisen erona on kuitenkin integraatio-töiden laadinnan koodi-lähtöisempi ajattelutapa. Luokkakirjastot ovat navigoitavissa päänäkymässä, sekä työkalun pikakomennot esimerkiksi uusien luokkien luomiseen ker-

tovat tästä. Talendissa uudelleenkäytettävien metadata-tietojen hyödyntäminen esimerkiksi tietokanta-yhteyksien määrittelyssä oli MuleStudiossa toteutettu Global Elementsien avulla. Sinänsä suurta eroa näiden suhteen sovelluksissa ei ollut.

MuleStudion kehitysympäristö koostuu viiteen eri elementtiin jaetusta näkymästä: package explorer, canvas, palette, connection explorer ja console. Näkymän yleiskuva on esitetty liitteessä 8. ’Package explorer’ on eräänlainen tiedostonhallintatyökalu, jossa projektin tiedostot ovat selailtavissa. ’Canvas’ toimii ”piirtoalustana” johon integraatiovirrat (flow) hahmotellaan MuleStudion tarjoamilla valmiilla työkaluilla, jotka ovat selailtavissa ’Palette’-listassa. Graafisen näkymän lisäksi MuleStudio tarjoaa canvasissa myös suoran näkymän integraatiotyön xml-koodiin. ’Console’ tarjoaa ajonaikaisen näkymän ohjelman suoritukseen. Saman elementin välilehdillä (’Mule Properties View’) pääsee säätämään komponenttien asetuksia.

MuleStudion keskeisimmät käsitteet integraatiologiikkaan liittyen ovat connectorit, endpointit, transformerit, filterit ja flow:t. Näiden tarkempi esittely olennaisilta osin tapahtuu tapausesimerkki-harjoituksen myötä. Kaikkiaan toiminnallisuudet tässäkin tapauksessa ovat integraatio-ohjelmistojen yleisperiaatteiden mukaiset.

Ennen tarkempaa syventymistä varsinaiseen tapausesimerkki-harjoitukseen nostetaan esille muutama yksittäinen MuleStudion ominaispiirre. Jo aikaisemmin mainittu viestipohjaisuus näkyy vahvasti ohjelmiston käytössä: integraatioiden toteuttaminen perustuu viestin käsittelyyn, missä viestin rakenne on käsitemallissa jaettu ensin otsikko- ja payload-osaan. Otsikko-osa on vielä jaettu inbound- ja outbound-ominaisuuksiin. Payload-osassa kuljetetaan varsinaista viestin tietosisältöä (esimerkiksi xml-tiedosto). Toisaalta viestin käsittelyssä MuleStudiossa tukeudutaan sovelluksen omaan kieleen, Mule Expression Languageen (MEL). Tämän opettelu ei eronnut suuresti esimerkiksi excelin funktioiden opettelusta tai olio-ohjelmoinnissa käytetystä pistenotaation omaksumises-

ta. Toki syvällisempi hallinta olisi vaatinut pidempää harjoittelua. (MuleSoft 2015b)

4.4.2 Toiminnallisuuksien määrittely MuleStudiossa

Alkuperäisen tavoitteen muututtua ensimmäisen tapaus esimerkin testausympäristön hyödyntämisestä puhtaasti MuleStudion ominaisuuksien perehtymiseen yksinkertaistetulla esimerkillä ei poistanut toista tavoitetta; tavoitteena oli edelleen laatia tapausesimerkki, jolla olisi selkeästi aito yhteys todellisen liikeelämän tarpeisiin ja konkreettiseen järjestelmä-integraatioon. Toisessa tapausesimerkissä hyödynnetään MuleStudion tiedoston sisältöpohjaista reitittämistä sekä sähköposti-liityntää. Ohjelmiston ominaisuuksista pyritään hyödyntämään laaja-alaisesti muiden muassa omaa skripti-kieltä, useita komponentteja, xml-määrittelyä, flow:n jakamista ”ali-flow:ksi”.

Konkreettisesti toisessa tapausesimerkissä hallitaan kansioon saapuvia xml-tiedostoja ja reititetään ne niiden sisällön perusteella edelleen eri kansioihin, ja mikäli jossakin tiedostossa esiintyy ehdot täyttävä sisältö, ohjautuu tästä tieto sähköpostitse käyttäjälle. Jos näitä peilaa todelliseen tarpeeseen, esimerkkinä voi esittää tapauksen, jossa yritys saa verkkokaupastaan tilaustiedot xml-tiedostoina ja tilauksien perusteella täytyisi tehdä ennalta määritettyjä jatkotoimia (tässä tilanteessa esimerkiksi sähköpostitse lähtisi tieto yhteyshenkilölle tai toiselle tavarantoimittajalle).

Koska toisen tapausesimerkin testausympäristössä ei ollut aitoja sovelluksia, joita yhdistää, ensimmäinen tehtävä oli laatia muutama testi-xml-tiedosto. Tätä varten ladattiin netistä satunnaista tilaustietoa sisältävä xml-tiedosto, josta muutamia kenttiä muokkaamalla tehtiin erilaisia tiedostoja (Sample XML-file haettiin osoitteesta: https://msdn.microsoft.com/en-us/library/bb387012.aspx). Muokattuja kenttiä olivat muun muassa nimi-, osoite-, tuotetietoja sisältävät tagit.

Tiedostojen muokkaamisella tähdättiin siihen, että toteutettavassa integraatiotapausesimerkissä xml-tiedostot ohjattiin sisältöperusteisesti ennalta määritet-

tyihin kansioihin. Sisältöperusteisuus reitittäminen tarkoittaa aineiston ohjaamista xml-tiedoston tietyn kentän arvon perusteella, esimerkiksi jos tuotteen nimi sisältää sanan ”palvelu” ja niin edelleen. Xml-tiedoston editointi tehtiin yksinkertaisella tekstieditorilla. Valmiit muokatut tiedostot (yhteensä viisi kappaletta) tallennettiin MuleStudion työhakemiston In-kansioon.

MuleStudion asentamisen jälkeen luotiin projekti ja flow varsinaista työtä varten. Flow:n ensimmäinen komponentti oli File-endpoint. Tilanne vastasi sitä, että esimerkiksi verkkokaupan tilauksista ohjautui (esimerkiksi SFTP-siirroin) xml-tilaustiedostot yrityksen tiedostopalvelimelle ja siellä edelleen kansioon. Koska kyseinen komponentti asennettiin flow:n ensimmäiseksi, tunnisti MuleStudio sen inbound-tyyliseksi, jossa ohjelmisto ”haistelee” kansiota ja laukeaa toimenpiteisiin, mikäli kansioon saapuu käsiteltäviä aineistoja.

Seuraavaksi komponentiksi flow:ssa lisättiin choice-router, jolla pystyy MuleStudiossa reitittämään viestejä/tiedostoja niiden tietosisällön perusteella. Sen toiminnan määrittely oli varsin suoraviivainen toimenpide: when-ehdon arvoksi määriteltiin XML-tagien sisällön mukainen vaatimus. Esimerkkinä kaikki Rovaniemelle toimitettavat tavarat määritettiin ohjattavaksi Rovaniemi-kansioon when-ehdolla *\PurchaseOrder\Address\City='Rovaniemi'*. Choice-routeriin voi määritellä tarvittavan määrän eri ehdon mukaisia toimintoja. Tässä tapauksessa luotiin kolme kansiota, joihin xml-tiedostot ohjautuivat riippuen niiden sisällöstä: Rovaniemelle toimitettavat tilaukset Rovaniemi-kansioon, ja jos vastaanottajalle täytyi soittaa ennen toimitusta (määritetty xml-tiedostossa kohdassa tagin ’deliverynotes’ sisällä). Jos kumpikaan ehdoista ei toteutunut, tiedosto ohjattiin oletuksena eri kansioon.

Tässä vaiheessa ensimmäisen tapausesimerkin kokemuksen opettamana oli parempi testata sovelluksen toimivuutta jokaisen vähäisemmänkin toiminnallisuuden lisäyksen jälkeen. Tähän saakka tehty toteutus siis ajettiin ja ohjelma kaatui virheisiin. Varsin kuvaavia ohjelmiston palauttamat virheilmoitukset eivät olleet. Lopulta ohjeita tutkiessa selvisi syy ja korjauskeino: integraation konfiguraatio-xml:ää joutui muokkaamaan siten, että when-ehdon jälkeen lisättiin ” eva-

luator=”xpath” ” (xpath, XML Path Language), millä käytännössä kerrottiin sovellukselle, miten käsiteltäviä xml-tiedoston tietokenttiä tuli tulkita. Lisäksi whenehdon keno-viivat olivat väärinpäin.

Seuraava kohdattu ongelma oli, vaikka ohjelma sinällään jo toimikin, MuleStudion muokkaus xml-tiedostot reitittämisen jäljiltä ei-luettavaan muotoon. Tämä vaikutti selkeältä bugilta. Toisaalta kyseessä saattoi olla raportoijan ongelma koneessa ja esimerkiksi Javan (JDK) tai Mavenin kehitysasetusten viallisuus. Koska mitään virheilmoitusta aiheesta ei tullut, ongelmaa oli erittäin työläs lähteä ratkomaan. Ainoa viite Mulesoftin keskustelupalstoilta (https://www.mulesoft.org/jira/browse/MULE-6842) viittasi siihen, että ’choice router’ muunsi xpath-attribuuttia käytettäessä viestin sisältöä (mikä ei ollut toivottu ominaisuus ja luokiteltavissa siis bugiksi).

Edellä mainittujen ongelmien vuoksi alkuperäiset tavoitteet hieman muuttuivat; reitityksen jälkeen viallista xml-tiedostoa ei tietolähteenä oikein voinut enää hyödyntää. Tästä syystä xml-tiedoston muokkaus mapper-työkalulla excel-tiedostoksi ja paremmin luettavaan muotoon jätettiin tekemättä. Kyseiseen työkaluun kuitenkin tutustuttiin, ja päällisin puolin kyseessä käytöltään samaa ideaa hyödyntävä kuin Talendin vastaava. Erityisen käytännöllinen toiminnallisuus oli mahdollisuus muodostaa mapper-työkalulla xml-tiedostoa lukeva eräänlainen xslt-muotti (Extensible Stylesheet Language Transformations), jolla alkuperäisestä xml-tiedostosta oli mahdollista luoda eri tiedostomuotoja.

Ongelmat tarjosivat kuitenkin hyvän keinon tutustua MuleStudion lokitoimintoihin. Tähän tarkoitukseen hyödynnettiin sekä ajonaikaista ”debuggausta” että logger-työkalua. Edellisen osalta ympäristössä oli mahdollista asettaa flow:n eri vaiheisiin ’toggle breakpointeja’, ja pysäyttää suoritus näihin etappeihin. Logger-työkalulla taas oli mahdollista kirjoittaa ajonaikaisia viestejä konsoliin. Loggerin asetuksissa annettiin sovelluksen omalla MEL-kielellä tarvittavat tarkennukset. Jos esimerkiksi haluttiin tieto tietyssä vaiheessa, mikä kuljetettavan viestin tiedostotyyppi oli, määritettiin loggeriin: ”Kansioon ohjautuessa tiedostotyyppi on #[message.dataType]”. Näillä ei kuitenkaan isoja hyötyjä näin

pienimuotoisessa integraatiossa saavutettu, mutta käyttökokemuksesta syntyi käsitys sovelluksen toiminnallisuudesta.

Samantapaista toimintoihin tutustumista tehtiin myös sub-flow:n osalta. Sen avulla on mahdollista eriyttää flow:n vaiheita ja pilkkoa toiminnallisuuksia osiin. Määrittely tapahtuu luomalla flow:n komponenteista MuleStudiossa ’flow reference’. Nämä sitten kuvataan normaalisti samalla tapaa kuin ylätason flow:tkin. Sub-flown avulla integraation toimintaa saa jäsenneltyä ja tehtyä tarvittaessa ehdollisia alitoimintoja samaan tapaan kuin Talendin alijobeissa.

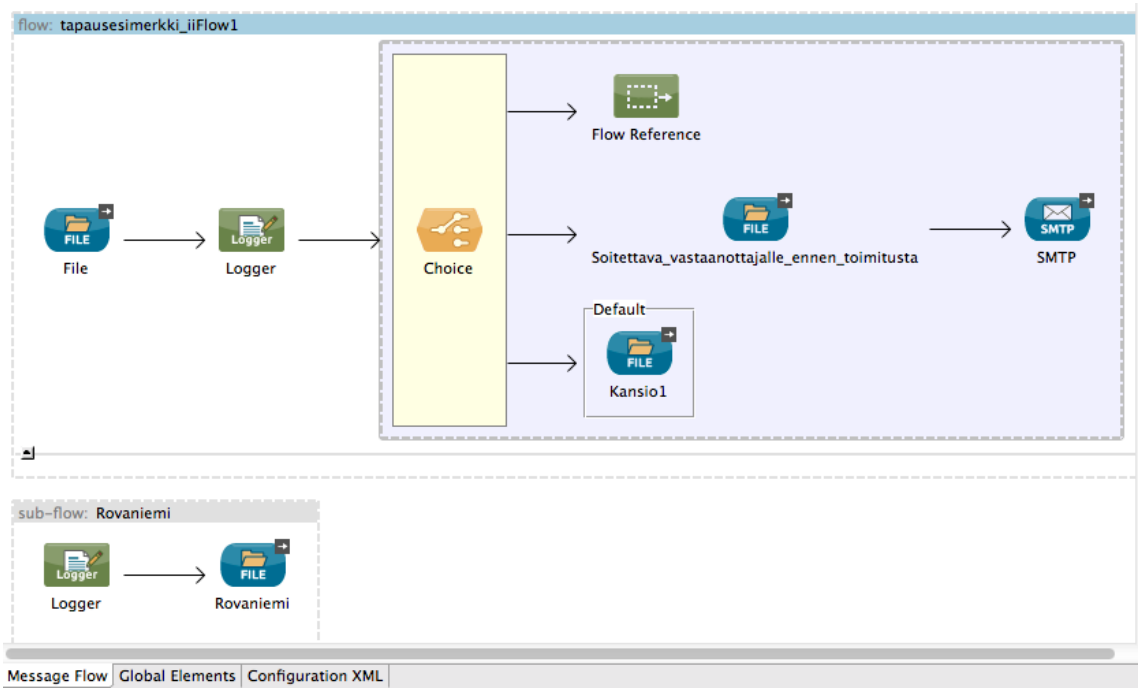
Lopuksi tapausesimerkkiin toteutettiin sähköposti-toiminto, jossa MuleStudion avulla lähetettiin tieto sähköpostitse aina tiedoston tiettyyn kansioon saapuessa. Toiminnon määrittely sinänsä oli yksinkertaista ja tähän käytettiin MuleStudion valmista SMTP-komponenttia (Simple Mail Transfer Protocol). Tarvittavat tiedot yhteyden luomiseksi pystyi antamaan asetuksissa tai määrittelemällä ne suoraan integraation konfiguraatio-XML-tiedostoon. Tämänkaltaisia yhteyksiä varten MuleStudiossa oli pakko luoda ’connector reference’, jossa annettiin yhteydenluonnille tarkemmat määritykset. Sama koski esimerkiksi FTP- ja tietokantayhteyksiä.

’Connector referencen’ määrittely oli versiopäivityksissä muuttunut, eikä tuorein ohjeistus tarjonnut enää tukea käyttöönotolle. Viestejä ei lopulta saatu lähtemään. MuleStudion komponentti oli toteutettu heidän dokumentaation mukaan javax.mailin API:a (Application Programming Interface) hyödyntäen. Yhteyden muodostamiseksi yritettiin hyödyntää Yahoon sähköpostitiliä SMTP-hostina. Näiden tarkempi tarkastelu jätettiin suosiolla pois. Testausympäristöön asennetusta asiakashallintajärjestelmästä SMTP-protokollan ja Yahoon välityspalvelimen kanssa viestejä kuitenkin onnistuttiin lähettämään.

4.4.3 Toinen Tapausesimerkki – yhteenveto

Kuviossa 6 on esitetty lopullinen MuleStudiolla toteutettu tapausesimerkki. Komponentteja on käytetty huomattavasti vähemmän kuin ensimmäisessä har-

joituksessa. Tapausesimerkin laadinta kuitenkin täytti oppimistavoitteet siltä osin, että MuleStudion hallinnan osaaminen kasvoi konkreettisen toteutuksen myötä. Syvällisin oppiminen tapahtui kuitenkin lukiessa oppaita ja etenkin tutustuessa käytännön esimerkkeihin, joita kehitysympäristöllä oli toteutettu. Opiskelu myös syvensi järjestelmäintegraatioiden ymmärrystä etenkin web-sovellusten ja esimerkiksi REST-periaatteiden mukaisesti toteutetuista liittymistä. Tässä kokonaisuudessa integraatio-ohjelmistolla (valmistajasta riippumatta) voi olla ratkaiseva merkitys toteutusten ja ylläpidon helpottamiseksi. Kuten ohjelmointiprojekteissa yleensäkin, myös järjestelmäintegraatioissa kokonaistilanteen ja riittävän suunnittelun merkitys korostuu.



Kuvio 6: MuleStudio ja toisen tapausesimerkin lopputilanne

4.5 Kolmas tapausesimerkki – JBoss Fuse

Viimeisessä toteutuksellisessa tapausesimerkissä oli tavoitteena tutustua vielä yhteen integraatio-ohjelmistoon, JBoss Fuseen. Ohjelmiston komentopohjaisen käyttöliittymän hyödyntämistä harjoitellaan varsin yksinkertaisella esimerkillä. Sovellus käy siinä läpi kansioita ja niiden tiedostoja ja ohjaa tiedostonimen perusteella tiedostoja edelleen eteenpäin haluttuihin kansioihin.

JBoss Fuse on avoimen lähdekoodin integraatioalusta (sisältäen ESB:n). Integraatioalusta JBossin tapauksessa koostuu useammastakin eri ohjelmistokehyksestä: muun muassa Apachen Active MQ:sta, Camelista ja CFX:stä. Suoranaista graafista kehitysympäristöä JBoss Fusessa ei ole tarjolla, vaan kehitys perustuu pitkälti tekstieditoriin ja toteutusten kääntäminen ja käynnistäminen komentotulkkiin. (Redhat 2015)

Koska kyseessä oli ohjelmisto ilman graafista kehitysympäristöä, sovelluksen opettelussa päästiin samalla syvemmälle myös integraatiototeutusten logiikkaan. Saman mekaniikan mukaan myös aiemmat esimerkit, Talend ja MuleStudio, toimivat mutta integraatioratkaisujen toteuttaminen niissä on käyttöliittymän avulla aloittelijalle huomattavasti helpompaa. Fusen asentaminen ja varsinaisten integraatioiden toimintaan saattaminen vaati enemmän vaivaa: komentotulkin (tässä tapauksessa Mac OS X:n Terminaali/Pääte) kautta tehtävät sovelluksien käynnistykset ja kääntäminen vaativat perehtymistä Terminaalin käytön alkeisiin ja Maven-kehitystyökaluun (ohjelmistoprojekteissa käytettävä työkalu).

Fusen avulla tehtävien integraatioiden opiskelu oli myöskin erilaista: valmiita tutoriaaleja ei ollut, vaan enemmänkin suoria koodiesimerkkejä, joiden ymmärrys taas vaati tiedonhakua eri aihealueista (mm. edellä mainittu Maven sekä XML-kieli). Fusen koostuessa useasta eri ohjelmistokehyksestä oli selvää, että kaikkien näiden omaksuminen edes hieman pintaa syvemmältä ei ollut mahdollista. Yksi tehtävä olikin valita järjestelmäintegraation kannalta olennaisin kokonaisuus ja pyrkiä syventymään tarkemmin siihen. Tavoite oli kuitenkin kerätä riittävästi tietoa, jotta pienimuotoisen harjoitustyön toteuttaminen oli mahdollista.

Perusasioiden opiskelussa hyödynnettiin Red Hatin omaa JBoss Fuse –dokumentaatiota sovellustoimittajien www-sivuilta sekä muutamaa Fusen kannalta olennaisen ohjelmistokehyksen perusteosta (mm. Instant Apache ServiceMix How-to, Learning Apache Karaf ja Instant Apache Camel Message routing). Varsinainen harjoitus tehtiin mukaillen JBoss Fuse oman kehityssivuston valmista demo-esimerkkiä (noudettavissa osoitteesta https://github.com/jboss-

fuse/quickstarts/tree/master/cbr). Kyseessä oli logiikaltaan varsin samanlainen kuin toisessa tapausesimerkissä MuleStudiolla toteutettu sisältöperusteinen tiedostonohjaus kansiosta toiseen. Harjoituksen suorittamisella haluttiin saada hieman konkreettista sisältöä oppaissa esitetyille asioille.

4.5.1 JBoss Fusen käyttötapaukset kolmannessa tapausesimerkissä

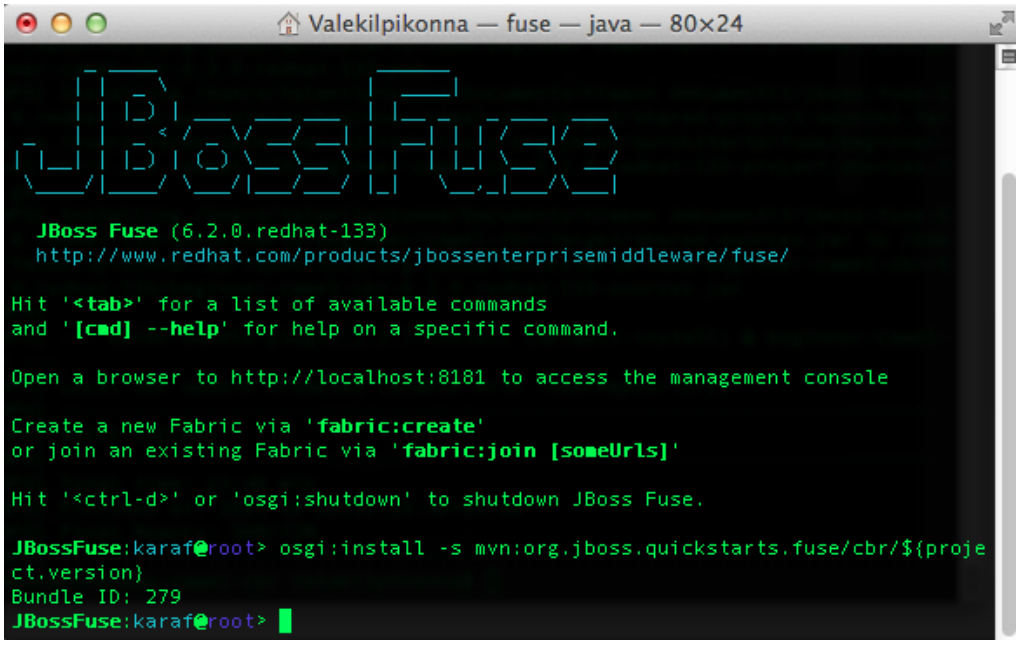
Ensimmäisenä tehtävänä oli asentaa JBoss Fuse (itse asennus tapahtui helpohkon asennustyökalun avulla). Tätä edeltävinä valmistelevina työvaiheina oli kuitenkin asentaa testikoneeseen Maven-työkalu, jota ei Mac OS X –koneissa nykyään oletuksena ole. Lisäksi myöhemmin ilmenneen tarpeen vuoksi nähdä järjestelmän piilotettuja tiedostoja avattiin Mac OS X –käyttöjärjestelmää hieman avoimemmaksi Terminaali-komennolla:

defaults write com.apple.finder AppleShowAllFiles TRUE

Lisäksi Mavenin ympäristömuuttujaa joutui välillä säätämään, tämä tapahtui myös Terminaalissa:

export PATH=/Library/apache-maven-3.3.3/bin:$PATH

Asennusvaiheiden jälkeen JBoss Fuse käynnistettiin Terminaalissa (kuvio 7). Tässä vaiheessa ohjelmisto oli käynnissä, mutta sillä ei ollut varsinaisesti mitään ”tekemistä”. Tästä päästiin varsinaiseen harjoitukseen ja hyödyntämään oppaiden tietosisältöä. Seuraavaksi onkin lyhyesti kuvattu harjoituksen sisältö ja samalla avattu hieman JBoss Fusen ohjelmistokehyksien keskinäistä mekaniikkaa.



Kuvio 7: JBoss Fuse käynnissä

Harjoitustyötä varten kolmannessa tapausesimerkissä hyödynnettiin varsin valmista demo-esimerkkiä. Tätä lähestyttiin tavallaan käänteisesti; tarkoitus oli saada harjoitustyö toimimaan ja tämän jälkeen selvittää tarkemmin, miten eri osaset harjoitustyössä vaikuttivat ja toisaalta ymmärtää oppaissa esille tulleita käsitteitä (joista oli alkuvaiheessa hankala saada konkreettista käsitystä).

Harjoitustyön tiedostot oli ensimmäiseksi kopioitu JBoss Fusen työhakemistoon (tosiasiassa sama harjoitustyö hieman eri muodossa löytyi myös asennuksen mukana tulleista Beginner-tutoriaaleista) ja varmistettiin että Maven työkalu oli oikein asennettu (komennolla ’mvn –v’). Tämän jälkeen Terminaalissa käynnistettiin projektin kääntäminen komennolla:

mvn clean install

Terminaali ilmoitti onnistuneen Maven-kääntämisen jälkeen yksinkertaisesti ”BUILD SUCCESS”. Maven käytännössä lukee työhakemistossa sijaitsevaa pom.xml-tiedostoa (Project Object Model), joka kertoo kääntäjälle projektiin liittyvät tiedot tarvittavien ohjelmistokirjastojen hakemista varten (Apache Software Foundation 2016). Kun kääntäminen oli valmis, käynnistettiin JBoss Fuse

(taas Terminaalissa, mutta erillisessä istunnossa). Suoritusta varten tarvittavat resurssit otettiin käyttöön komennolla (deploy bundle):

osgi:install -s mvn:org.jboss.quickstarts.fuse/cbr/${project.version}

Tällä saatiin asennettua OSGi-säiliökomponenttiin (container) kuuluvat sovelluskomponentit (bundles). Käytännössä komennon osista ja parametreista oli eroteltavissa ’-s’, jolla määrättiin sovellus käynnistymään asennuksen jälkeen ja jälkimmäisenä oleva Maveniin viittaava polku, mistä varsinaiset sovelluskomponentit noudettiin käännettäväksi. Testilaitteiston Maven-asetukset taas määrittelivät, mistä haettua kansiopolkua (org/jboss/quickstarts/fuse/cbr) lähdettiin hakemaan. Tämä osoite oli määritelty tarkalleen ottaen Mavenin asetusten settings.xml-tiedostossa ja sen repository-kohdassa (url-tagien sisällä): https://repo.fusesource.com/nexus/content/groups/public.

Onnistuneesta install-komennosta kertoi saatu Bundle ID (kuvio 7). Integraatio-ohjelmisto oli sovellusta käynnistettäessä luonut JBoss Fusen työhakemistoon Input- ja Output-kansiot. Näihin sijoitettiin demo-esimerkin mukana tulleet testi-xml-tiedostot, joiden oli määrä ohjautua sisällön perusteella oikeisiin kansioihin.

JBoss Fuselle toteutettu sovellus perustuu yleensä malliin, jonka pohjalla ovat Java-ohjelmointikieli, blueprint-riippuvuusinjektio ja jo hieman sivuttu Maven-kehitystyökalu. Yksinkertaisissa toteutuksissa, kun käytetään Redhatin tarjoamia valmiita harjoitusesimerkkejä, pärjää ilman syvällistä Java-tuntemusta ja toisaalta riippuvuusinjektioiden pintapuolinenkin ymmärtäminen olisi kokonaan toisen raportin aihe (ja liittyen enemmänkin ohjelmistokehitykseen kuin järjestelmäintegraatioihin). Saman voi katsoa koskevan JBoss Fusen OSGi-kehystä. Niinpä päädyttiin käytännön harjoituksen osalta tutustumaan erityisesti integraatioon liittyvään osaseen, joka tässä tapauksessa oli sovelluksen reititystä ohjaava cbr.xlm-tiedosto. Tämän tulkinta osoittautui jo aiempien tapausesimerkkien kokemuksien myötä helpoksi: tilaustiedostoja (xml-muotoisia) tulkittiin xpath-syntaksin perusteella, reititys itsessään oli samaisella choice-työkalulla kuin MuleStudiossa aikaisemmin toteutettu harjoitus (joskin nyt ei ollutkaan graafista

työkalua, vaan määritys oli tekstipohjainen, xml-tagien sisällä), lisäksi mukana oli loki-toimintoja reitityksien sanalliseksi kuvaamiseksi Terminaalissa.

Suuria muutoksia cbr.xml-tiedostoon ei ollut tehtävissä, jotta toiminnallisuutta olisi suuresti saanut muutettua. Kyseeseen tulivat lähinnä tilaustiedostojen reitittäminen eri sisältöperustein. Tämän toteuttaminen vaati Mavenin lukemien tiedostojen sijoittamista paikalliseen repositorioon (jotta asetuksia pääsi muokkaamaan). Alkuperäinen demo-esimerkki oli toteutettu niin, että tiedostot ohjautuivat maa-tiedon perusteella oikeisiin kansioihin (kuvio 8).

```
<route id="cbr-route">
    <from uri="file:work/cbr/input" />
    <log message="Receiving order ${file:name}" />
    <choice>
        <when>
            <xpath>/order:order/order:customer/order:country = 'UK'</xpath>
            <log message="Sending order ${file:name} to the UK" />
            <to uri="file:work/cbr/output/uk" />
        </when>
        <when>
            <xpath>/order:order/order:customer/order:country = 'US'</xpath>
            <log message="Sending order ${file:name} to the US" />
            <to uri="file:work/cbr/output/us" />
        </when>
        <otherwise>
            <log message="Sending order ${file:name} to another country" />
            <to uri="file:work/cbr/output/others" />
        </otherwise>
    </choice>
    <log message="Done processing ${file:name}" />
</route>
```

Kuvio 8: CBR.xml-tiedoston reititysmäärittely

4.5.2 Kolmas tapausesimerkki - yhteenveto

Kaikkiaan JBoss Fusen kanssa työskentely oli haastavaa: sovelluksien kääntäminen ja käynnistäminen oli työlästä ja virhetilanteissa ongelmien syyn selviäminen kesti. Aikaa tuhraantui paljon selvittelyyn: kun sovelluksia käännettiin, hukkui välillä käsitys siitä mistä julkisesta repositoriosta sovelluksen käyttämät tiedostot todellisuudessa haettiin. Ajonaikaisen suorituksen osalta tällä ei suurta merkitystä ollut, mutta toimintalogiikka ei auennut. Niinpä alun kunnianhimoiset

tavoitteet päästä hyödyntämään monimutkaisempiakin integraatio-sovelluksia kariutuivat ja sitä kautta syvällisempi perehtyminen myös integraatio-ohjelmistojen toimintalogiikkaan jäi toteutumatta. Sen jäljille kuitenkin päästiin.

Kaikkiin nyt selkoa tehtyihin ohjelmistoihin liittyi olennaisesti ’Enterprise Integration Patterns’ –käsite, eräänlainen integraatiologiikan malli, jota toiminnallisuus noudatti. Aiheeseen perehdyttiin kirjallisen aineiston avulla, mutta käytännön toteutus jäi haastavasta toimintaympäristöstä johtuen alkumetreille. Sinänsä mielenkiintoinen kommentti tarttui aineistoon tutustuttaessa mieleen, kun todettiin että järjestelmäintegraatioiden logiikka on liian kompleksinen, jotta sitä voisi hallita tavallisella ”cookbook”-lähestymistavalla (Hohpe 2016). Tällä tavoin kuitenkin oli nyt ohjelmistoihin pääasiassa tutustuttu ja kyseisen huomion voi tältä osin allekirjoittaa.

Kokonaisuus oli laaja ja niinpä monimutkaisempien toteutusten luonti jäi kesken, mikä aiheutti raportin muodon kääntyvän loppua kohden yhä enemmän oppimispäiväkirja-tyyliseksi. Tämä piti osin jo paikkansa edellisenkin tapausesimerkin osalta. Valitettavasti kokonaiskuvan hahmottamisen vaikeus teki myös kirjallisen raportin kirjoittamisen ymmärrettävään muotoon haastavaksi. Tavoitteiden täyttyminen kolmannen tapausesimerkin osalta jäi puutteelliseksi.

4.6 Neljäs tapausesimerkki – Talend ja Business Model

Luvussa 2.4 esiteltiin lyhyesti niin kutsuttujen BPM-sovellusten hyödyntäminen liiketoimintaprosessien kuvaamisessa ja yhtenä työkaluna integraatioiden teknisessä hahmottamisessa. Tässä luvussa on tarkoitus tutustua Talendin Open-Studion sisältämään BPM-työkaluun yhtenä liiketoimintaprosessin kuvausvälineenä. Sitä ennen kuitenkin tarkennetaan vielä hieman teoreettista taustaa BPM-sovelluksiin liittyen.

Talendin oma ohjeistus kertoo liiketoimintaprosessin/-mallin (business model) kuvaavan graafisesti tarvemäärittelyn esimerkiksi järjestelmäintegraatioprojektin ohjausryhmälle ottamatta suoraan kantaa vielä teknisiin vaatimuksiin. Toisaalta

samaa ei-teknistä-mallia voidaan hyödyntää apuna järjestelmätuen henkilöille ymmärtämään asetetut vaatimukset integraatioille. (Talend 2016)

4.6.1 Liiketoimintaprosessien kuvaamisesta yleisesti

Tähtinen erottaa Järjestelmäintegraatio-kirjassaan selkeästi liiketoimintaprosessi-käsitteestä omakseen integraatioprosessit; edellisen ollessa nimenomaan niittä toimintoja joilla yritys toimii ja tekee liiketoimintaansa luoden kilpailuetua, ja jälkimmäisen ollessa enemmän sitä miten ne käytännössä toteutetaan. Hänen käsityksensä BPM-työkaluista on niiden rooli teknisten prosessien mallinnuksessa ja hallinnoinnissa. (Tähtinen 2005, 62.) Tähtinen osuu oikeaan siinä mielessä että nykyisillä työkaluilla on vielä hankala päästä aidosti *hallinnoimaan* yrityksen prosesseja. Tässä raportissa BPM-työkalu Talendin tapauksessa voidaan kuitenkin katsoa toteuttavan tarkoitustaan puhtaasti prosessin kuvaamisessa.

Varsinainen tutustuminen Talendin BPM-työkaluun toteutetaan tekemällä yksinkertainen liiketoimintaprosessin kuvaaminen. Ennen yksityiskohtaisempaa kuvausta työkalun käytöstä on huomautettava, ettei Talendin tarjoama BPM-työkalu ole toiminnallisuuksiltaan erityisen kattava. Markkinoilta löytyy laajasti tarjontaa prosessityökaluista monipuolisemmilla ominaisuuksilla. Näistä yksi kokonaisvaltainen esimerkki on suomalaisen QPR Softwaren Process Designer.

4.6.2 Liiketoimintaprosessin kuvaaminen Talendissa – case matkalaskut

Kuvattava prosessi on matkalaskujen käsittely kuvitteellisessa yrityksessä. Edellä mainittu viittaus liiketoimintaprosessiin yrityksen kilpailuetuna ei tässä tilanteessa ole erityisen relevantti; harvan yritys voi saavuttaa kilpailuetua käsittelemällä tehokkaammin työntekijöidensä matkalaskuja (toki jos kyseessä on taloushallintopalveluita tarjoava yritys, joka myy niiden käsittelyä ulkoistuspalvelua, tilanteen voi ajatella olevan toinen).

Matkalaskuprosessi aiheutuu yrityksen työntekijän matkustaessa ja näin ollessa oikeutettu matkakulukorvauksiin (Lahti & Salminen 2014, 101). Prosessi on hahmoteltu hyödyntäen kirjallisia lähteitä Digitaalinen taloushallinto lukua ’matka- ja kululaskut’ sekä Successful Business Process Managementin lukua ’Mapping Your Process’, sekä raportoijan kokemusta oman organisaation matkalaskujen käsittelystä. Yksinkertaisimmillaan prosessi on kuvattavissa hahmottamalla ensin prosessin osallistujat/roolit, suoritettavat toimenpiteet, tulokset ja tarvittaessa myös tietojärjestelmät. Järjestelmäintegraatio-mielessä nyt esiteltävässä kuvauksessa erityisesti järjestelmänäkökulmaa on pyritty painottamaan.

Koska matkalaskuprosessi sinänsä on varsin yksinkertainen, ei sen kuvaamiseen sanallisesti käytetä laajemmin aikaa. Riittää kun kuvaa sen pääpiirteissään niin, että siitä ovat tunnistettavissa prosessin toimijat, sovellukset ja tuotokset. Tässä tapauksessa toki oletetaan kyseessä olevan sähköinen prosessi, eikä esimerkiksi matkalaskuja kierrätetä organisaatiossa paperisena.

Prosessi alkaa kun yrityksen työntekijä tekee matkan mahdollisesti omaa autoa käyttäen ja on oikeutettu päivärahoihin (päivärahaoikeuden määrittelee verohallinto sen mukaan miten kauas matka suuntautuu ja miten pitkään matka kestää). Joko matkan päättyessä tai matkan alkaessa (kilometrien tallennus esim. GPS:ää tukevan laitteen avulla) työntekijä laatii matkalaskusovelluksella matkasta matkalaskun (työntekijän tiedot matkalaskusovellukseen ajatellaan tässä tulevan automaattisesti palkkajärjestelmästä). Työntekijän esimies tyypillisesti hyväksyy tehdyn matkalaskun todeten käytännössä matkan aiheellisuuden liiketoimintanäkökulmasta. Tämän jälkeen vielä tarvittaessa yrityksen erillinen talous- tai henkilöstöosasto tarkastaa matkalaskun teknisen oikeellisuuden. Jälkimmäisen osalta matkalaskusovellukset ovat tehneet vaiheesta hieman tarpeettoman. Talous- tai henkilöstöosastolle on kuitenkin usein jäänyt tehtäväksi matkalaskun maksaminen työntekijälle (erillinen sovellus). Lisäksi matkalaskusta täytyy välittää tieto kirjanpitoon kulujen kirjausta varten.

Kuvauksesta on nopea tunnistaa prosessin kuvaamiseksi eri tekijät:

- toimijat: työntekijä (laatii matkalaskun), esimies (hyväksyy), talous-/henkilöstöhallinto (validoi, maksaa)
- tuotokset/tiedot: matkalasku, työntekijätiedot, maksu-/kirjanpitoaineisto
- järjestelmät: matkalaskusovellus, kirjanpitosovellus, palkkajärjestelmä, maksatusohjelma

Näiden tietojen avulla voidaan tutustua Talend Studion Business Models –työkaluun. Se on varsin yksinkertainen, eikä tue esimerkiksi useista muista prosessityökaluista oletuksena löytyvää uimaratakaaviota (esim. Microsoft Visio). Uuden mallin luomisen jälkeen käytettävissä ovat työkalupaletissa graafiset elementit mallinnusta varten. Talendin työkalu on lopulta hyvin yksinkertainen, joten käyttö ei ole kaukana piirtosovelluksen käyttämisestä: isoin työ prosessin kuvaamisessa onkin etukäteissuunnittelussa ja kokonaiskuvan hahmottamisessa, lopulta ”piirtäminen” on vain lyhyt työvaihe (toki riippuen jossain määrin prosessin laajuudesta). Liitteessä 9 on esitetty Talend Studion Business Models –työkalulla tehty prosessikuva (kuvasta on selkeyden vuoksi jätetty pois ehdolliset toimenpiteet, esim. onko esimies hyväksynyt laskun tai onko järjestelmäintegraatioissa virheitä).

Prosessikuvan piirtämisen jälkeen Talend ei tarjoa toimintoja kaavion hyödyntämiseen esimerkiksi tarjotakseen pohjaa jobien luonnille. Elementteihin on kuitenkin mahdollista yhdistää repositorio-moduulista esimerkiksi metadatatietueita, joten prosessikuvaa voi pitää eräänlaisena dokumentaation jatkeena integraatioille. Työkalusta siis on apua suunnittelussa, mutta sen ominaisuudet ovat puutteelliset verrattuna moneen muuhun prosessityökaluun. Etenkään varta vasten prosessinmallintamiseen keskittyneitä sovelluksia Talendin työkalu ei syrjäytä. Sitä voi kuitenkin käyttää tietyn rajauksin prosessimallien hyödyntämiseen yrityksien operatiivisessa toiminnassa, kuten tukena perehdyttämisessä, kommunikoinnissa ja johtamisessa yhtä hyvin kuin alustavasti tunnistamaan yksiköiden ja funktioiden välisiä rajapintoja, mitkä QPR Software listaa prosessimallinnuksen keskeisiksi tavoitteiksi (QPR Software 2016).

5 POHDINTA

Alun perin opinnäytetyön aihe valikoitui järjestelmäintegraatio-opintojaksolle suoritetusta harjoitustehtävästä, ja teoreettinen tausta pohjautuikin integraatio-ohjelmistoista tehtyyn lyhyeen esseeseen. Lopulta voi todeta taustatiedon keräämisen ja niihin tutustumisen olleen melkeinpä suurin osuus koko opinnäytetyöprosessissa. Ajallisesti koko työn valmistumiseen kului lähes vuosi, mikä sisälsi aktiivisempia jaksoja muun muassa raportin kirjoittamisen osalta. Koska aikajänne oli varsin pitkä, ongelmaksi nousi ajoittain liian pintapuoliseksi jääneet selvitystyöt eri ohjelmistoihin tutustumisessa. Tavoitteena oli myös tehdä riittävän laaja harjoitusympäristö erilaisine ohjelmistoineen, joilla järjestelmäintegraatioita pystyi teknisesti toteuttamaan. Esimerkiksi muutama sisällönhallintajärjestelmä asennettiin ja niiden käyttöön tutustuttiin, mutta lopulta niille ei löytynyt sopivaa roolia järjestelmäintegraatio-mielessä. Lisäksi käytiin läpi useita ilmaisia taloushallintosovelluksia niin pilvisovelluksina kuin työasemallekin asennettavina versioina. Tätä työtä auttoi selkeästi asetettu tavoite löytää sellaisia sovelluksia, joissa tietoa sai sovelluksiin näppärästi ulos ja sisään. Lopulta tähän sopivia ilmaisia sovelluksia löytyi yllättävän vähän.

Keskeinen tavoite opinnäytetyössä oli varsinainen oppimisprosessi: tutustua tarjolla olevien integraatio-ohjelmistojen toimintaan ja ominaisuuksiin. Tähän tarkoitukseen rakennettu testausympäristö toimi hyvin etenkin ensimmäisen tapausesimerkin ja ohjelmiston (Talend) osalta. Vaativa osuus itse ohjelmiston opiskelussa oli myös tyydyttävän tapausesimerkin rakentamisessa. Tähän käytettiin mielikuvitusta ja ideaa pyöriteltiin pitkäänkin ”suunnittelupöydällä”. Keskeisiin toiminnallisuuksiin Talendin integraatio-ohjelmistossa saatiin selkoa ja selkeitä yhteneväisyyksiä alun teoriakehykseen löytyi. Konkreettisimpana esimerkkinä olivat ESB:n toiminnallisuuksien hyödyntäminen (rikastaminen, reititys ja muunnokset). Kahden jälkimmäisen tapausesimerkin osalta tilanne oli hieman toinen. Niiden toimintalogiikka pohjautui raportoijan havaintojen mukaan enemmän sanomapohjaiseen järjestelmäintegraatioon. Tämä laajensi opiskeltavaa aluetta yli alkuperäisen alueen muun muassa web-ohjelmoinnin osalta. Eikä toisaalta sanomaliikennettä ohjaavaa testausympäristöä saatu kevyesti

rakennettua. Tästä johtuen loppua kohti raportin kirjoittaminen kääntyi enemmän ja enemmän oppimispäiväkirjatyyliseksi.

Tavoitteet täyttyivät osittain. Oli antoisaa saada aidonkaltaisessa ympäristössä toteutettua järjestelmäintegraatio hyödyntäen integraatio-ohjelmistoa. Syvällisempää logiikkaa ei kuitenkaan saatu avattua ohjelmistojen taustalla. Tämän osalta olisi ehkä tarvittu ”mestari-oppipoika”-menetelmää: nyt ohjelmistojen toiminta avautui jokseenkin teorian tasolla, mutta käytäntöä ei saatu todennettua johtuen esimerkiksi ohjelmistojen virhetilanteista ja aidon tuotantoympäristön puuttumisesta. Käytettävyydeltään kuitenkin sovellukset olivat (JBoss Fusea lukuun ottamatta) hyvin samanlaisia ja oppikirjojen lukuisat termit ja tekniikat saivat niiden myötä hieman lihaa luiden ympärille. Toteutetut tapausesimerkit pidettiin lopulta kohtalaisen yksinkertaisina, jotta saatiin jossain määrin toimivia integraatioita. Toisaalta nykypäivän ketterissä ohjelmistokehitysmenetelmissä tämä ideologia on vallalla: tehdään ensin jotain toimivaa, ja kehitetään siitä edistyksellisempää.

Vaikkakin lopulta osa harjoituksista jäi pintapuoliseksi toteutuksen osalta, oli rohkaisevaa huomata, että järjestelmäintegraatioihin teknisesti tutustuessa pystyi hyödyntämään monen eri opintojakson oppeja. Harjoitukset vaativat osaamista tietokannoista, ohjelmistokehityksestä, kokonaisarkkitehtuurista, web-ohjelmoinnista ja Unix-järjestelmistäkin. Harjoitusten kuvaamisen ei ole tarkoitus toimia kenellekään käyttöohjeena. Kokemuksesta voi kuitenkin suositella oppimismenetelmänä sitä, että aikaisessa vaiheessa asentaa opiskeltavat sovellukset ja testaa toiminnallisuuksia itse. Käyttöohjeiden lukeminen kuitenkin kannattaa aina.

Keskeinen huomio tutustuessa järjestelmäintegraatio-alaan liittyy sen nopeaan kehitykseen. Yrityksillä on vielä laajasti käytössään erillissovelluksia, joiden yhdistäminen keskenään on työlästä ja vaativat monesti laajoja IT-projekteja. Nykyään yrityksillä on yhä enemmän suuntaus SaaS-ratkaisuihin (Software as a Service), eikä sovellusten omistajuus heillä itsellään ole arvokasta. Tämä voi tarkoittaa, etteivät sovellukset ole niin vahvasti räätälöityjä mutta vaatimukset

niiden yhdistettävyydelle kasvavat. Tämä on yksi selittäjä API-tekniikoille (avoimille ja vähän suljetuimmillekin). Miten integraatio-ohjelmistot on suhteutettavissa tähän? Raportoijan havainnon mukaan usea integraatio-ohjelmisto ja etenkin varsinaiset ESB-ratkaisut ovat rakennettu etenkin erillisohjelmistojen integraatioiden helpottamiseen ja edustavat sitä kautta hieman mennyttä teknologiaa. Usealla sovellustoimittajalla on kuitenkin kääntynyt fokus enemmän ja enemmän tukemaan sanomaliikennettä ja olemaan yksi hyödyllinen osanen API-rajapintojen välisessä tiedonvälityksessä. Tulevaisuuden integraatio-ohjelmistot ovat kuitenkin todennäköisesti IaaS-tyyppisiä (Integration as a Service), aivan kuten erillissovellukset yleistyvät SaaS-ratkaisuiksi. (Collins & Sisk 2015, 2.)

Kaikkiaan järjestelmäintegraatioiden laajasta kokonaisuudesta saatiin sekä teoriaosuudessa ja teknisesti sovellusten osalta hyvä määrä oppia. On hyväksyttävä aihealueen laajuus ja se, että opiskelun lähtiessä lähes nollista kunnollisellakaan panostuksella ei saavuteta täyttä tietämystä. Raportoijan iloksi oli kuitenkin hienoa huomata työelämässä muutaman konkreettisen järjesteläintegraatioesimerkin osalta yhtymäkohtia tehtyyn tutkimukseen: Apache Servicemix (JBoss Fusen teknologiaa) hyödynnetään aidossa ympäristössä verkkolaskuliikenteen reitityksen ja taloushallintoon liittyvää konsernikonsolidointia järjestelmästä toiseen helpotetaan tutulla integraatio-ohjelmistolla hyödyntäen niin tiedon rikastamista kuin validointiakin. Tulevaisuudessa digitalisaation yleistyessä laajemminkin elinkeinoelämässä järjestelmäintegraatiot tulevat jatkossakin olemaan isossa roolissa. Aiheen ymmärryksestä on varmasti hyötyä.

LÄHTEET

Apache Software Foundation 2016. Apache Maven Project. What is Maven? Viitattu 15.1.2016 http://maven.apache.org/what-is-maven.html.

Berman, P. 2014. Successful Business Process Management: What You Need to Know to Get Results. New York: AMACOM Books.

Bowen, J. 2012. Getting Started with Talend Open Studio for Data Integration. Birmingham: Packt Publishing Ltd.

Collins, G. & Sisk, D. 2015. API economy. From systems to business services (näyte). Viitattu 13.2.2016 http://www2.deloitte.com/content/dam/Deloitte/uk/Documents/technology/deloitte-uk-api-economy.pdf.

A Comparison of Open Source Software ESB Solutions 2010. Webinaaritallenne. Esittäjä Rob Cope, a Roque Wave Company. Viitattu 16.6.2015 http://www.openlogic.com/events/on-demand-webinars/comparison-open-source-software-esb.

Del Río Benito, A & Edidin, H. 2013. Microsoft BizTalk ESB Toolkit 2.1. Birmingham: Packt Publishing Ltd.

Dikmans, L. & van Luttikhuizen, R. 2012. SOA Made Simple. Discover the true meaning behing the buzzword that is 'Service Oriented Architecture'. Birmingham: Packt Publishing Ltd.

Flashnode 2016. Flashnode Oy. Mitä meistä sanotaan? Asiakkaat. Viitattu 13.2.2016 http://www.flashnode.com/mita-meista-sanotaan.

Fowler, M. & Webber, J. 2008. Does My Bus Look Big in This? Viitattu 12.7.2015 http://www.infoq.com/presentations/soa-without-esb.

FrontAccounting 2015. FronAccounting Wiki: Manual Import of Customers. Viitattu 11.11.2015 http://frontaccounting.com/fawiki/index.php?n=Devel.ManualImportOfCustomers.

Hautamäki, T. 2015. Oikeaa vai turhaa työtä. Helsingin Sanomat 14.6.2015, D2-D3.

Havey, M. 2008. SOA Cookbook. Design Recipes for Building Better SOA Processes. Birmingham: Packt Publishing Ltd.

Herreira, R. 2010. ESB is an architectural style, a software product, or a group of software products? Viitattu 15.7.2015 http://www.consultoriajava.com/articulos/esb_arquitecture_software_product.jsp

HiQ Finland Oy 2015. Palvelut: Integraatiopalvelut. Viitattu 10.7.2015 http://www.hiqfinland.fi/Integraatiopalvelut.

Hohpe, G. 2016. Enteprise Integration Patterns. Patterns and Best Practices for Enterprise Integration. Viitattu 20.1.2016 http://www.enterpriseintegrationpatterns.com.

Juric, M., Loganathan, R., Sarang, P. & Jennings, F. 2007. SOA Approach to Integration. Birmingham: Packt Publishing Ltd.

Kansallinen Terveysarkisto 2015. HL7. Viitattu 18.7.2015 http://www.kanta.fi/web/ammattilaisille/hl7.

Kress, J., Berthold, M., Normann, H., Schmeidel, D., Schmutz, G., Trops, B., Utschig-Utschig, C. & Winterberg, T. 2013. Enterprise Service Bus. Answers to some of the most important questions surrounding the term "enterprise service bus" using concrete examples, so that the areas of application that can be deemed "correct" for ESBs can be clarified. Viitattu 15.7.2015 http://www.oracle.com/technetwork/articles/soa/ind-soa-esb-1967705.html.

Lahti, S. & Salminen, T. 2014. Digitaalinen taloushallinto. Helsinki: Sanoma Pro Oy.

Laliwala, Z., Samad, A., Desai, A. & Vyas, U. 2013. Mule ESB Cookbook. Birmingham: Packt Publishing Ltd.

Malinen, T. 2013. Mitä tarkoittaa ohjelmiston integraatio ja miksi se voi tuplata kilpailuetusi? Viitattu 15.7.2015 https://www.sofokus.com/blogi/mita-tarkoittaa-ohjelmiston-integraatio-ja-miksi-se-voi-tuplata-kilpailuetusi.

Manouvrier, B. & Menard, L. 2010. Application Integration. EAI, B2B, BPM and SOA. Lontoo: Wiley-ISTE.

Microsoft 2004. EAI Example. Viitattu 8.7.2015 https://msdn.microsoft.com/en-US/library/ee267542(v=bts.10).aspx.

Microsoft 2014. Introducing BizTalk Server. Viitattu 17.7.2015. https://technet.microsoft.com/fi-fi/library/aa547058.aspx.

Microsoft 2015. SOA in the Real World. Chapter 1: Service Oriented Architecture. Viitattu 11.7.2015 https://msdn.microsoft.com/en-us/library/bb833022.aspx.

MuleSoft 2015a. ESB Solutions. Viitattu 15.7.2015. https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb.

MuleSoft 2015b. MuleSoft // Dev. Mule Fundamentals. https://docs.mulesoft.com/mule-fundamentals/v/3.7.

O’Brien, R. 2008. Integration Architecture Explained. Viitattu 12.7.2015 http://hubpages.com/business/Integration-Architecture-Explained.

Oracle 2015. Oracle Service Bus. Viitattu 16.7.2015 http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html

QPR Software 2016. Palvelujen digitalisointi ja prosessijohtaminen. Tuloksellisempaa toimintaa prosessijohtamisella. Viitattu 2.2.2016 http://www.qpr.com/fi/konsultointipalvelut/prosessijohtaminen/prosessien-mallintaminen-ja-hyödyntäminen-kehitystyössä.

Rademakers, T. & Dirksen, J. 2009. Open Source ESBs in Action. Example Implementations in Mule and Servicemix. Greenwich: Manning Publications Co.

Ratkaisujen Suomi 2015. Pääministeri Juha Sipilän hallituksen strateginen ohjelma 29.5.2015. Hallituksen julkaisusarja. Viitattu 6.7.2015 http://valtioneuvosto.fi/sipilan-hallitus/hallitusohjelma.

Red Hat 2015. Red Hat JBoss Fuse. Overview. Viitattu 16.7.2015 http://www.jboss.org/products/fuse/overview/.

Solita 2016. Solita Oy. Näin olemme auttaneet asiakkaitamme. Viitattu 13.2.2016 http://www.solita.fi/asiakkaat/.

Talend 2015. Talend Products. Viitattu 16.7.2015 http://www.talend.com/products/application-integration.

Talend 2016. Talend Help Center. What is Business Model. Viitattu 25.1.2016 https://help.talend.com//pages/viewpage.action?pageId=265113057.

Tähtinen, S. 2005. Järjestelmäintegraatio. Tarve, vaihtoehdot, toteutus. Helsinki: Talentum.

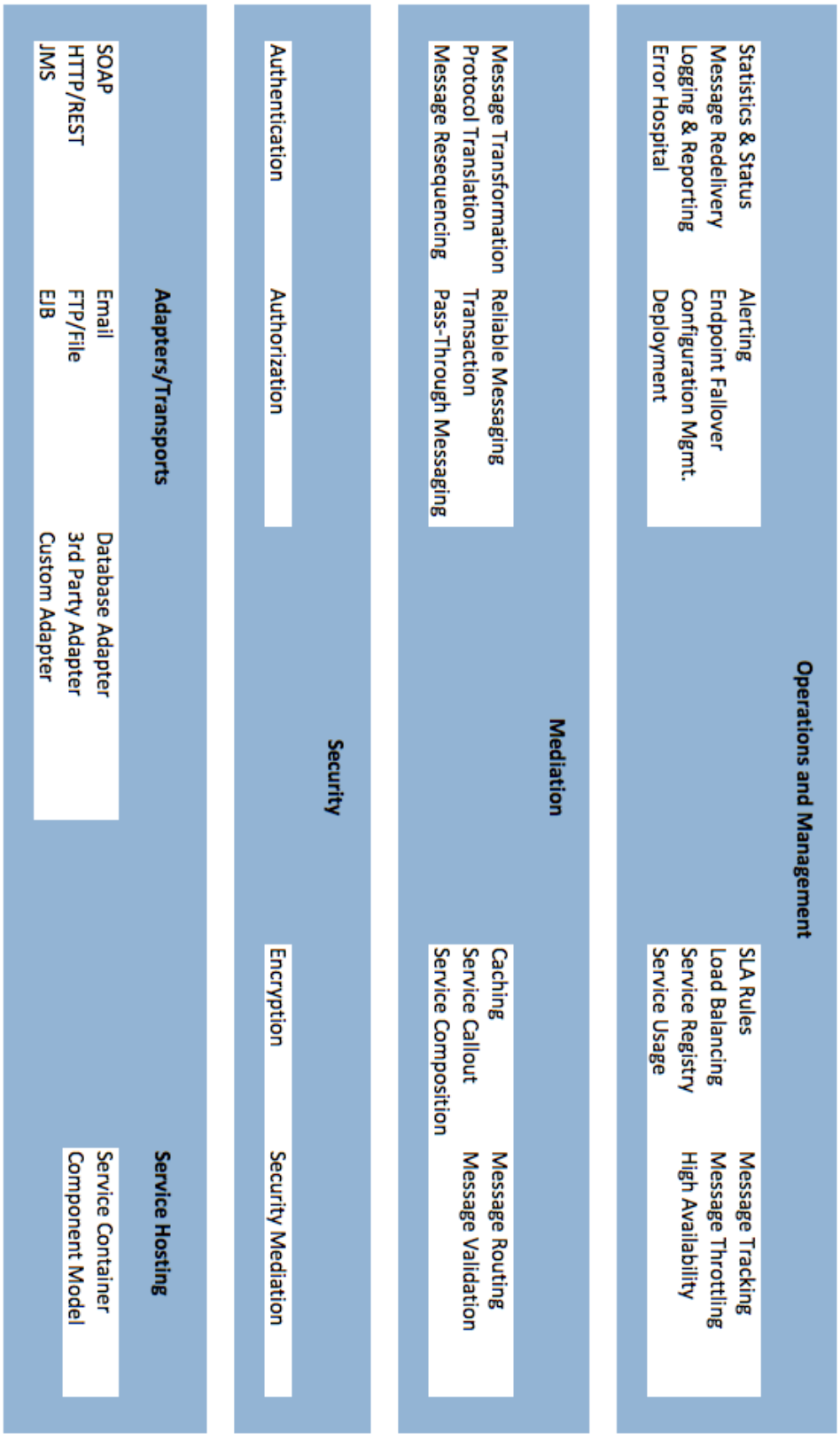
Youredi 2016. Youredi Cases. Viitattu 13.2.2016 http://www.youredi.com/case-studies.

WSO2 2015. WSO2 Enterprise Service Bus. Viitattu 17.7.2015 http://wso2.com/products/enterprise-service-bus/.

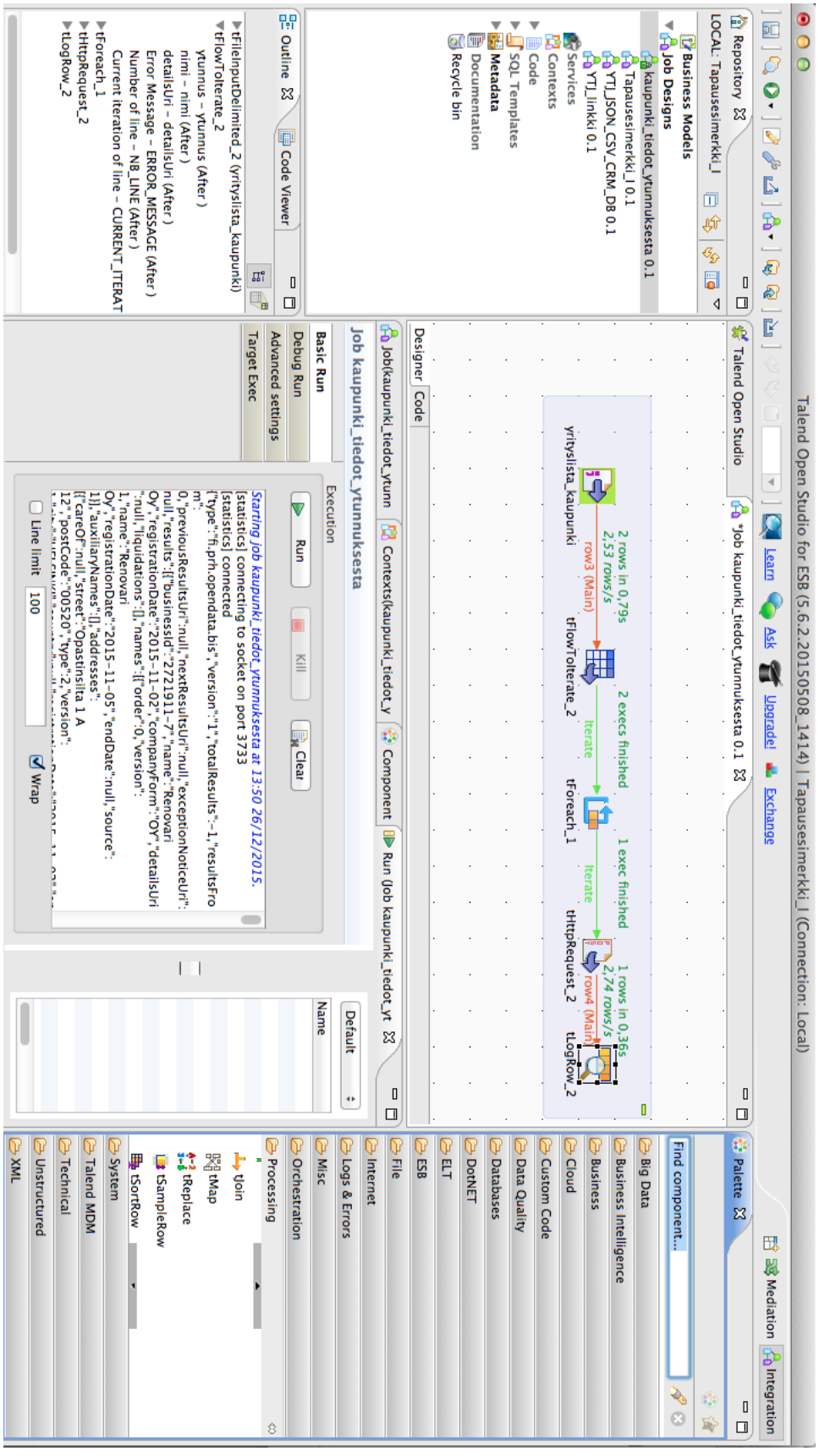
Wähner, K. 2013. Choosing the Right ESB for Your Integration Needs. Viitattu 13.7.2015 http://www.infoq.com/articles/ESB-Integration.

LIITTEET

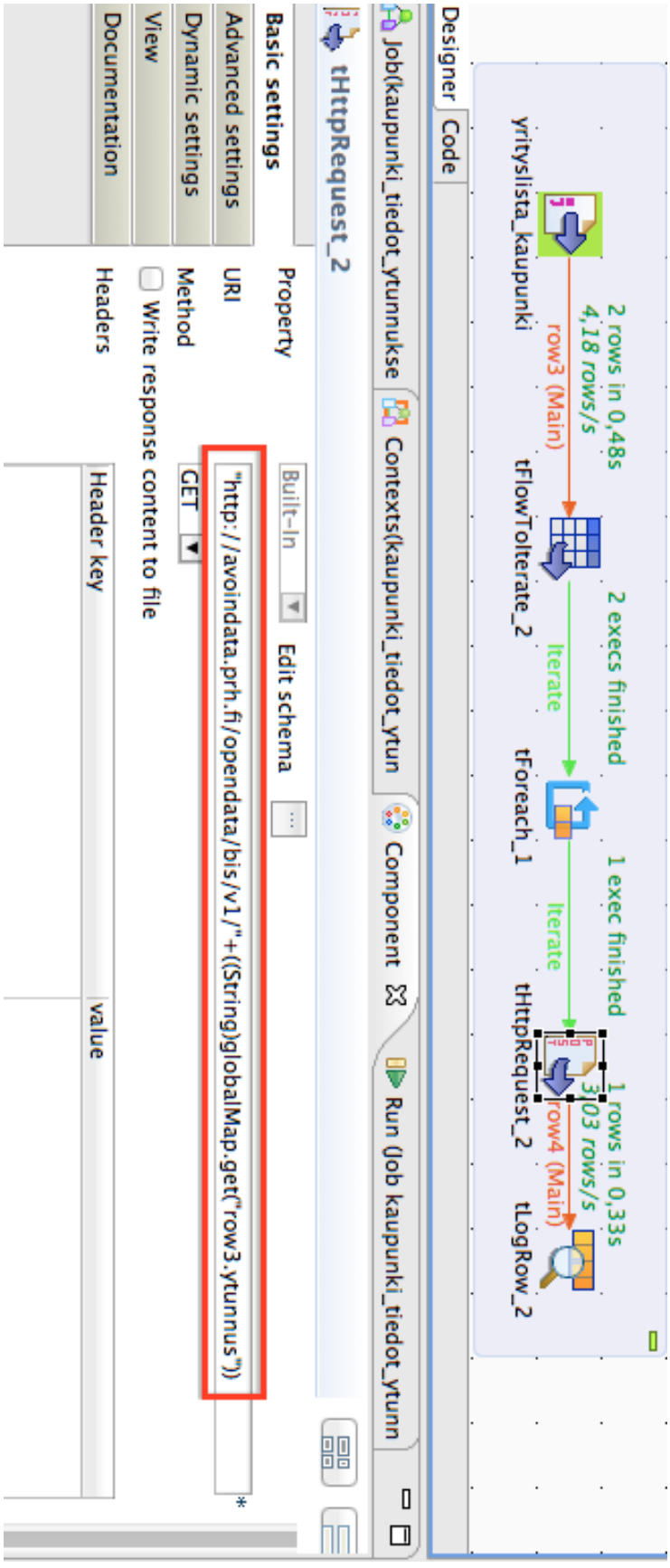
Liite 1: ESB toiminnallisuudet Oraclen mukaan (Kress ym. 2013)



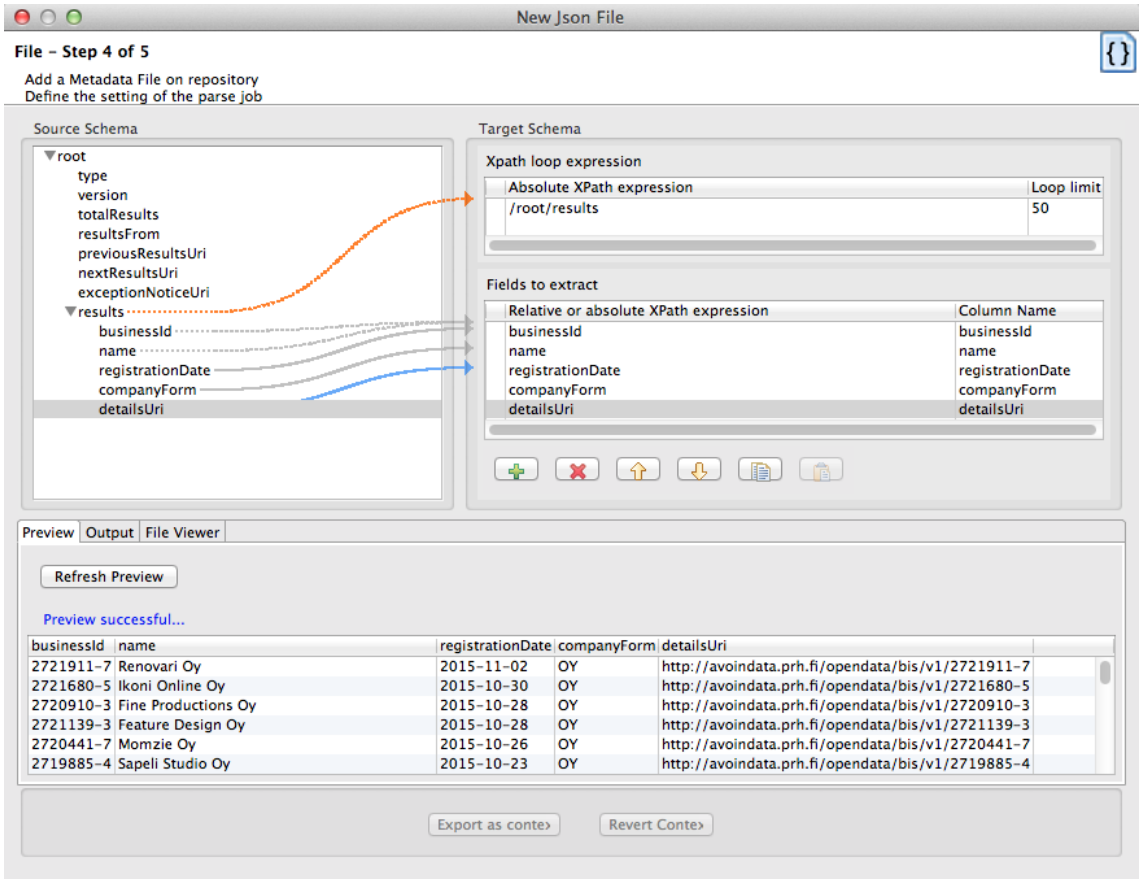
Liite 2: Talend kehitysympäristö, yleiskuva



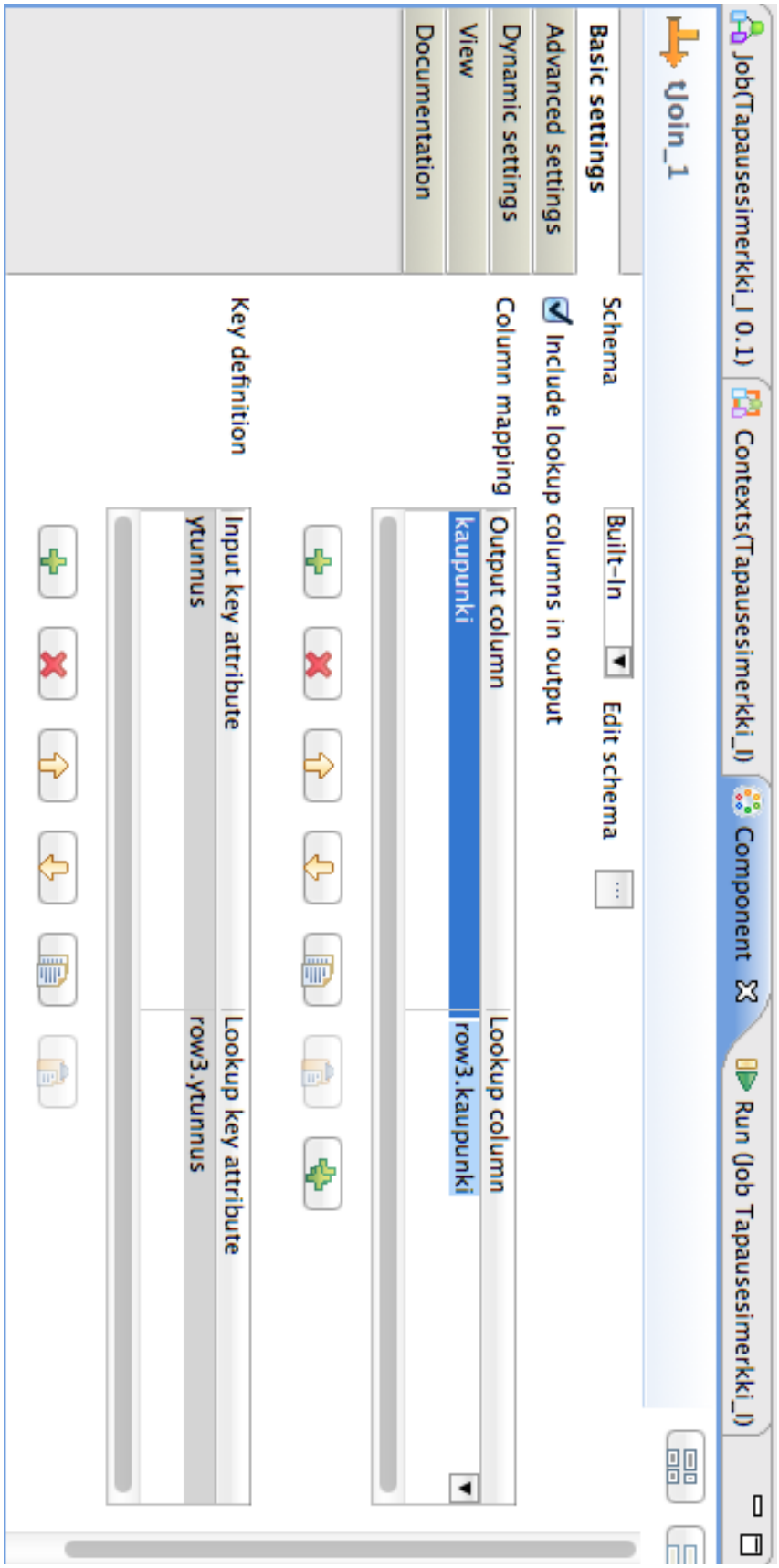
Liite 3: Talend thttprequest-komponentti, iteraation paikallismuuttujan määrittely



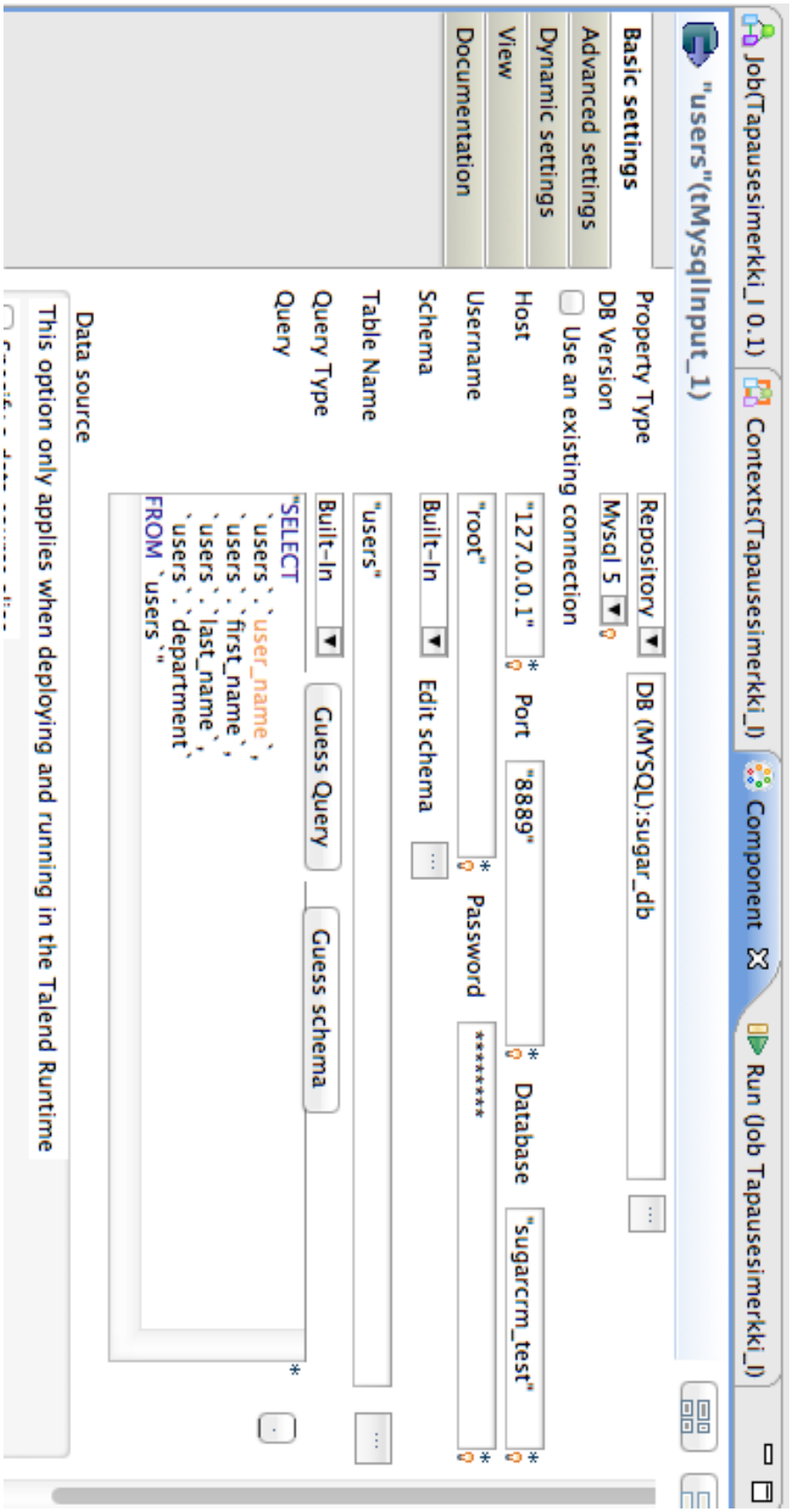
Liite 4: Talend JSON-metadatan määrittely



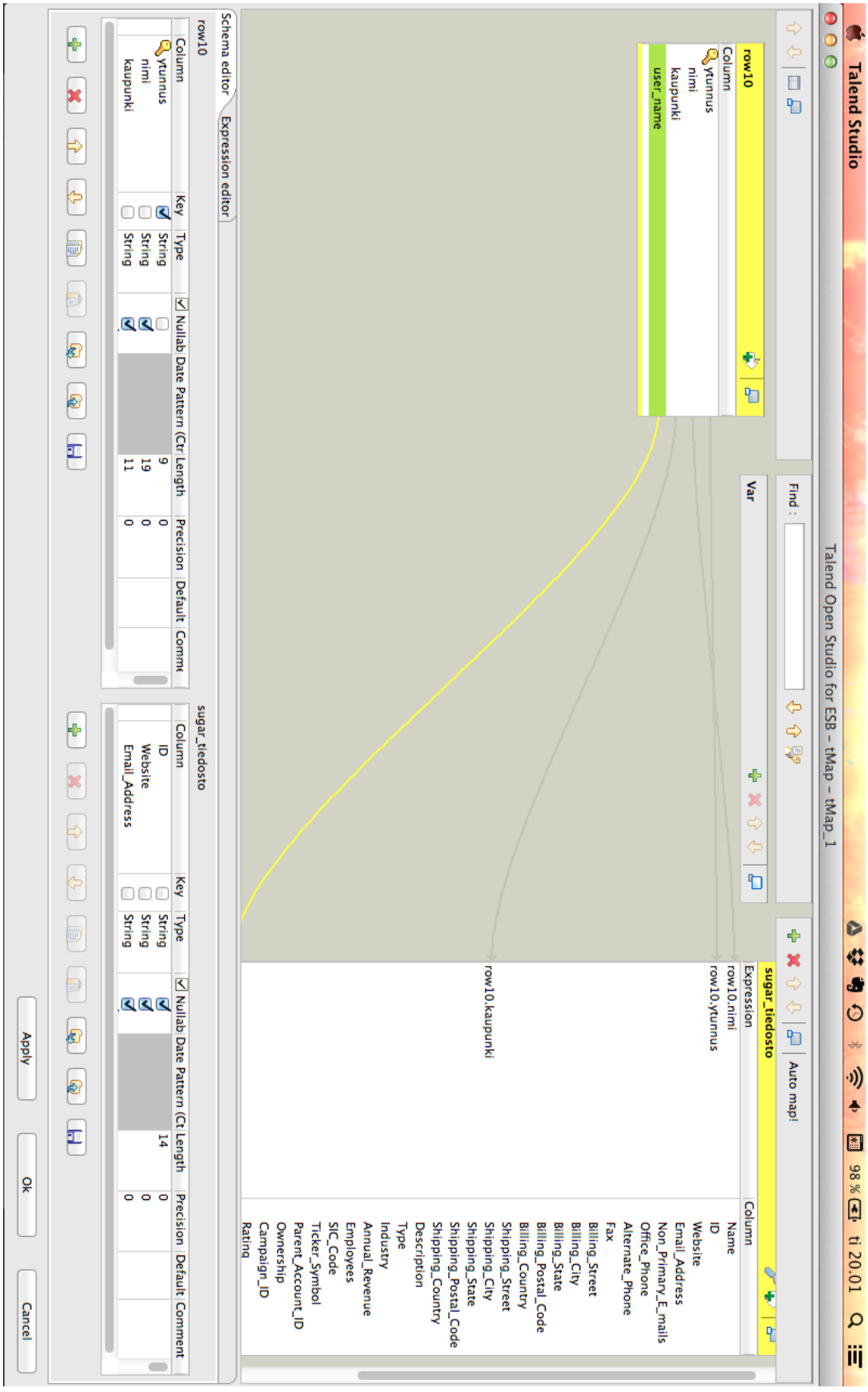
Liite 5: Talend tJoin-komponentin asetukset (kaupunki-tiedolla rikastaminen)



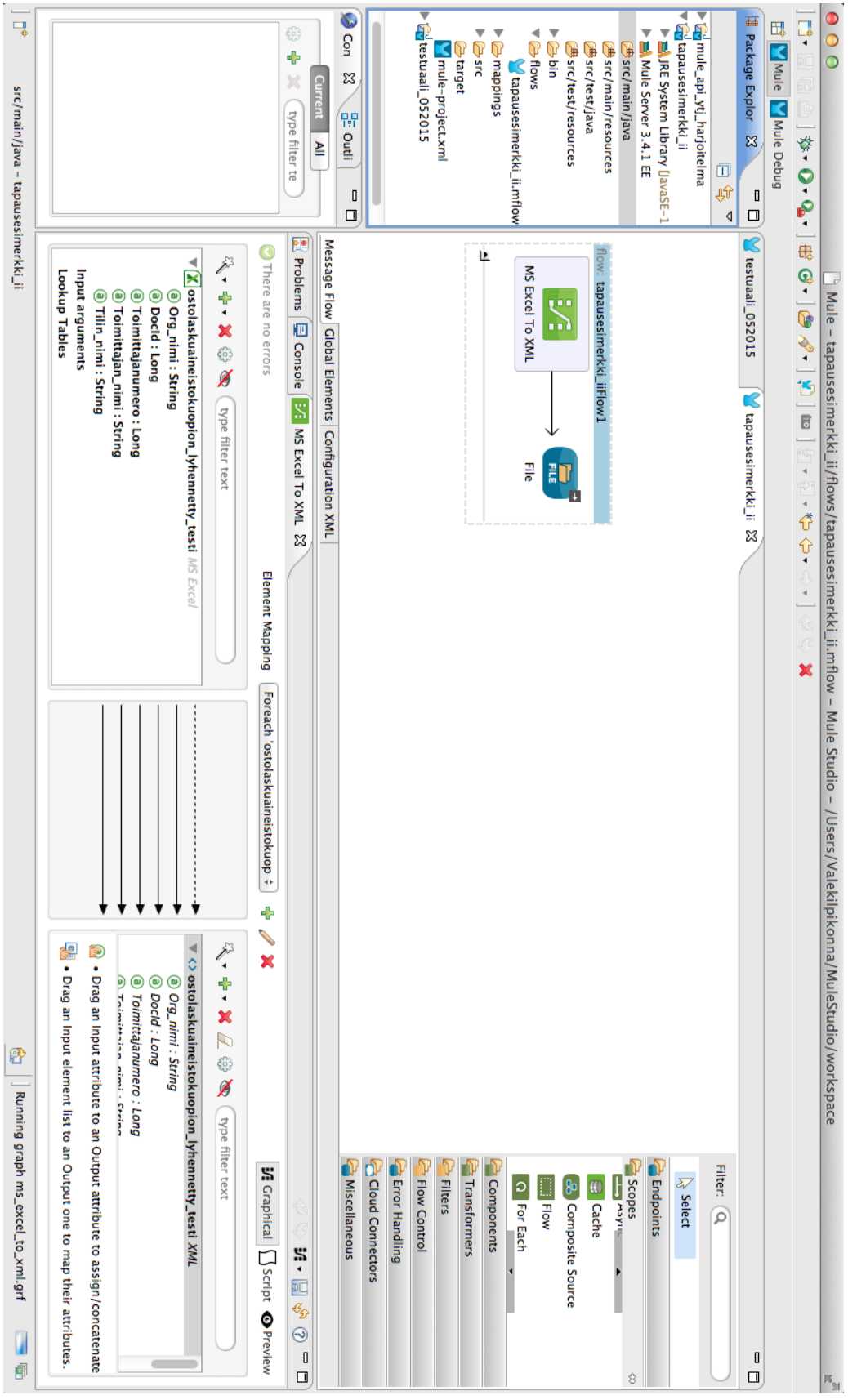
Liite 6: SugarCRM-tietokantakomponentin asetukset Talendissa



Liite 7: Talend Map-expression työkalu



Liite 8: MuleStudio – yleisnäkymä



Liite 9: Matkalaskuprosessikuva Talendin työkalulla luotuna