Tampereen ammattikorkeakoulu

ServiceMix-integraatio Tampereen korkeakouluyhteisölle

Käyttäjätietojen raportointi SAP-järjestelmästä

Hanna Haataja

OPINNÄYTETYÖ
Joulukuu 2019

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

HAATAJA, HANNA:
ServiceMix-integraatio Tampereen korkeakouluyhteisölle
Käyttäjätietojen raportointi SAP-järjestelmästä

Opinnäytetyö 53 sivua, joista liitteitä 6 sivua
Joulukuu 2019

Tässä opinnäytetyössä selvitettiin kattavasti integraatioiden toteutusmalleja ja niiden kehitystä. Digia Finland oy:n toimeksiannosta opinnäytetyössä selvitettiin ServiceMix-integraatioteknologiaa ja sen toimintaperiaatteisiin. Samalla perehdyttiin Tampereen korkeakouluyhteisön ohjeisiin vaatimusmäärittelyn tekemiseksi ja integraatioiden toteuttamiseksi. Lisäksi opinnäytetyön osana tuotettiin vaatimusmäärittelyt kahdelle integraatiolle ja toteutettu määritellyt integraatiot.

Opinnäytetyö toteutettiin konstruktiivisena tutkimuksena, jonka lopputuloksena tuotettiin talouden järjestelmästä käyttäjätietoja tuovat ja niitä raportoivat integraatiot. Aineistona käytettiin sekä alan kirjallisuutta, että Tampereen korkeakouluyhteisön ohjeita ja aiempia integraatiototeutuksia. Näitä aineistoja analysoitiin tekstianalyysin ja havainnoinnin keinoin.

Tässä raportissa esitellään kattavasti integraatiomalleja ja Tampereen korkeakouluyhteisön integraatiopalvelutoteutusta. Lisäksi esitellään kahden integraation määrittely- ja toteutusprosessit.

Opinnäytteessä kerättiin onnistuneesti vaatimusmäärittelyt toteutettaville integraatioille. Opinnäytetyössä tuotetuista integraatioista toinen on tuotantokäytössä ja toinen on vielä testattavana joulukuussa 2019.

Asiasanat: servicemix, integraatio, palveluväylä

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Software Development

HAATAJA, HANNA:
ServiceMix Integration for Tampere Universities
User Information Reporting from SAP

Bachelor's thesis 53 pages, appendices 6 pages
December 2019

This thesis focused on integration patterns, ServiceMix technology, as well as the instructions of the Tampere Universities for defining requirements and implementing integrations. In addition, requirement specifications for two integrations were produced and defined integrations were implemented.

Constructive research was carried out and as a result requirement specification were defined for the two integrations. The material for this thesis included literature as well as instructions from the Tampere Universities and their previous integration implementations. These materials were analyzed by textual analysis and observation.

Two integrations were produced as a part of this thesis. One of those is now in use in the production environment.

Key words: servicemix, integration, enterprise service bus

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä on toteutettu konstruktiivinen tutkimus, jonka tavoitteena on selvittää Tampereen korkeakouluyhteisön integraatioiden määrittely ja toteutustapaa. Tavoitteena on antaa kattava kuvaus malleista, joilla integraatiota voidaan toteuttaa, ja esitellä tarkemmin ServiceMix-teknologiaa, jolla palveluväylämalli voidaan toteuttaa.

Tutkimuksen tarkoituksena on kerätä määrittelyt toteutettaville integraatioille ja dokumentoida ne korkeakouluyhteisön ohjeistuksen mukaan. Tarkoituksena on myös toteuttaa määrittelyjen pohjalta integraatiot ja dokumentoida ne. Tutkimus on toteutettu tekstianalyysin ja havainnoinnin keinoin.

Opinnäytteen toimeksiantajana toimii Digia Finland Oy, joka on kasvava ohjelmisto- ja palveluyritys. Digia toiminnan osana integraatiototeutukset ovat vahvassa kasvussa. Digia on Tampereen korkeakouluyhteisön kumppani, jonka ansiosta tämän opinnäytetyön aiheena on korkeakouluyhteisön integraatioväylätoteutus.

Tampereen yliopisto ja Tampereen ammattikorkeakoulu yhdistyivät vuoden 2019 alussa Tampereen korkeakouluyhteisöksi, jonka seurauksena vanhat organisaatiot kokivat suuria muutoksia. Korkeakouluissa yleensä on hyvin tärkeä saada tieto liikkumaan eri järjestelmien välillä, jotta opiskelijat saavat arvosanansa, unohtamatta taloushallinnon ja henkilöstöpalveluiden tarpeita.

Opinnäytteessä esitellään integraatiomalleja ja Tampereen korkeakouluyhteisön tapoja toteuttaa integraatiopalveluväylä. Samalla esitellään integraatiopalvelu, joka tarjoaa muille korkeakouluyhteisön osa-alueille integraatioiden toteutusta ja ylläpitoa. Samalla perehdytään integraatiopalvelun määrittelyprosesseihin ja integraatiototeutuksentapoihin.

2 INTEGRAATIOMALLIT

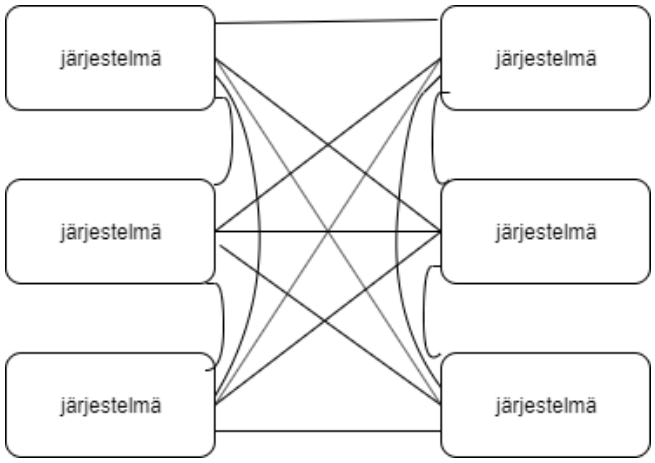
2.1 Erilaiset integraatiotavat ja -mallit

Järjestelmäintegraatiolla tarkoitetaan toimintoja tai teknologioita, joiden tarkoituksena on yhdistää organisaation sisälle hajautettu tieto. Aikaisemmin yhteensopimattomat järjestelmät saadaan näin keskustelemaan keskenään. (Tähtinen 2005, 14.)

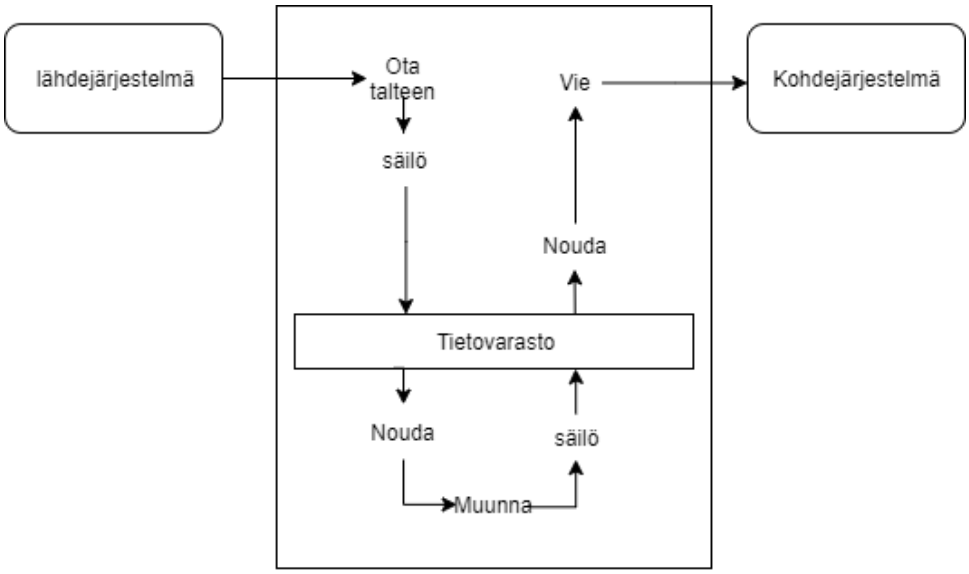
Historiallisesti integraatioiden arkkitehtuuria kuvatessa voidaan puhua neljästä erilaisesta aikakaudesta. Ensimmäinen aikakausi oli ennen järjestelmäintegraatioita, eli eristäytyneiden järjestelmien aikakausi. Seuraava askel oli järjestelmien yhdistäminen toisiinsa suorilla yhteyksillä, eli pisteestä pisteeseen -integraatioiden aikakausi. Kolmantena kautena mukaan otettiin keskitetty ratkaisu, jonka avulla päästiin teollisen tason integraatioratkaisuihin. Neljäntenä ajanjaksona voidaan ajatella palvelukeskeisen arkkitehtuurin periaatteiden käyttöön ottoa. (Tähtinen 2005, 144.) Viime vuosina on kehityksessä otettu jo viidettä askelta, kun integraatioväylistä ollaan siirtymässä mikropalveluarkkitehtuuriin. Mikropalveluarkkitehtuurissa yksinkertaiset, yhteen asiaan keskittyneet sovellukset kommunikoivat keskenään tarkkaan dokumentoiduilla ja avoimilla rajapinnoilla. (Vähimaa 2017.)

2.1.1 Pisteestä pisteeseen -integraatio

Pisteestä pisteeseen -integraatio (point-to-point) käsittelee järjestelmiä pareittain. Järjestelmät liitetään toisiinsa integraatiolla, joka hakee tiedot lähdejärjestelmästä, muuntaa ne kohdejärjestelmän tarvitsemaan muotoon ja vie ne kohdejärjestelmään (kuvio 1). Integraation toiminta voi olla synkronista, eli tietojen haku ja vienti tapahtuu samaan aikaan, tai asynkronista, eli tietojen haku ja vienti eivät ole ajallisesti sidottuna toisiinsa. Asynkronisessa integraatiossa tiedot tallennetaan noutoa odottamaan esimerkiksi tietokantaan tai viestijonoon (kuvio 2). (Bussler 2003, 9.)



KUVIO 1. Pisteestä pisteeseen -integraatiomallin arkkitehtuurikuva (Tähtinen 2005, muokattu)



KUVIO 2. Asynkronisen integraation kaavio (Bussler 2003, muokattu)

Pisteestä pisteeseen -integraatiomalli tarjoaa kyllä integraation perusominaisuudet, mutta sen ongelmat tulevat esiin isoissa ja monimutkaisissa toimintaympäristöissä. Jos isoon ympäristöön lisätään järjestelmä, joka pitäisi integroida jokaiseen ennestään integroituun järjestelmään, täytyy jokaisesta järjestelmästä määritellä ja toteuttaa kaksi integraatiota, yksi, joka tuo uuteen järjestelmään tietoa ja toinen, joka vie tietoa uudesta järjestelmästä. Lisäksi näihin integraatioihin täytyy määritellä tietomuunnokset molempiin suuntiin. (Bussler 2003, 9.)

Jos ajatellaan, että yrityksellä on 10 järjestelmää entuudestaan. Yhden järjestelmän lisääminen, joka vaatii integraation kaikkiin vanhoihin järjestelmiin, tarkoittaisi 20 integraation ja tietomuunnoksen määrittelyä ja toteuttamista.

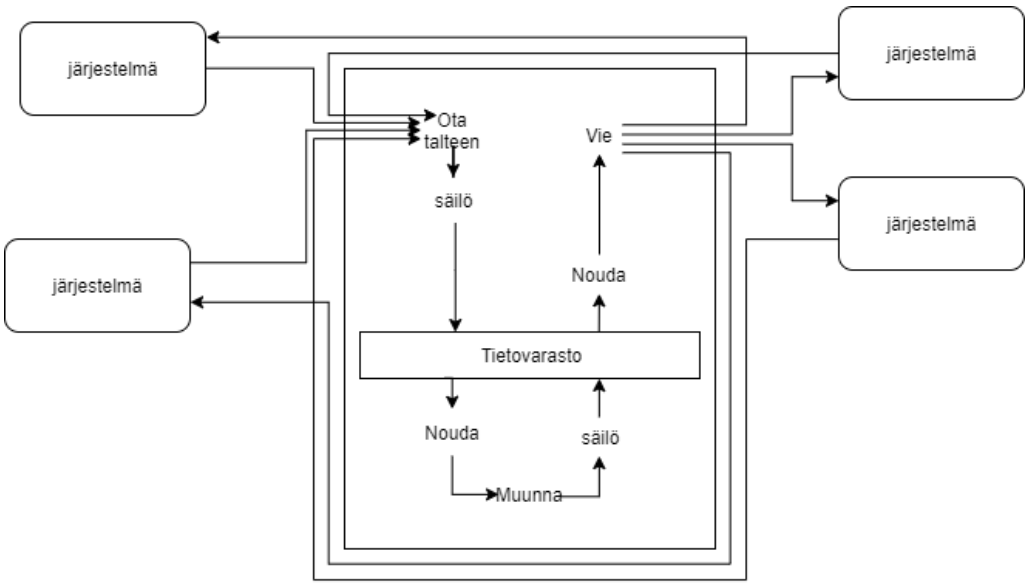
Pisteestä pisteeseen -integraatioiden kömpelyys tulee erityisesti esille tilanteessa, jossa halutaan tietää jokin liiketoiminnan kannalta tärkeä tieto. Esimerkiksi, jos halutaan listata kaikki siirrossa olevat tilaukset, täytyy kyselyt kohdistaa kaikkiin tietovarastoihin integraatioissa, jotka liikuttavat tilauksia. (Bussler 2003, 10.)

Lisäksi tässä mallissa ei ole mahdollisuutta määritellä monimutkaisempia viestiketjuja. Koska integraatiot on määritelty tasan kahden järjestelmän välille, ei ole mahdollista esimerkiksi rikastaa viestiä kolmannesta lähteestä. (Bussler 2003.)

Integraatioita suoraan järjestelmästä toiseen tehdään tilanteen niin vaatiessa. Pisteestä pisteeseen -integraatiomallille on tyypillistä, että se rakentuu yritykseen pitkällä aikavälillä, eri tekniikoilla, eri suunnittelijoiden ja toteuttajien tekemänä ja vaihtelevalla huolellisuudella dokumentoituna. Jos järjestelmästä järjestelmään integraatiota päätetään tehdä, on erityisen tärkeää, että integraatiot ja kaikki siihen liittyvät asiat dokumentoidaan huolella. (Tähtinen 2005, 66.)

2.1.2 Puhelinkeskusmalli

Puhelinkeskusmalli (hub-and-spoke) siirtää tietovaraston yksittäisistä integraatioista kaikille yhteiseksi tietovarastoksi (kuvio 3). Puhelinkeskusmallissa integraatio ei ole enää kahden järjestelmän välillä vaan järjestelmän ja keskuksen välillä. Tässä mallissa keskus siirtää tiedon kohdejärjestelmälle. Kuten pisteestä pisteeseen -integraatiomallissa täytyy tässäkin mallissa tehdä tietomuunnoksia. Tietomuunnokset voivat tapahtua ennen tiedon tallentamista keskukseen, ennen tiedon vientiä kohdejärjestelmään tai erillisessä prosessissa keskuksessa. (Bussler 2003, 11.)



KUVIO 3. Perinteinen puhelinkeskusintegraatiomalli (Bussler 2003, muokattu)

Perinteisessä puhelinkeskusintegraatiossa lähdejärjestelmä määrittelee kohdejärjestelmän. Kohdejärjestelmän tiedot toimitetaan keskukseen viestin otsikossa ja keskus välittää tiedon kohdejärjestelmälle ja hoitaa tietomuunnokset. Tämä tarkoittaa sitä, että jokainen järjestelmä on tietoinen toisistaan. (Bussler 2003, 11.)

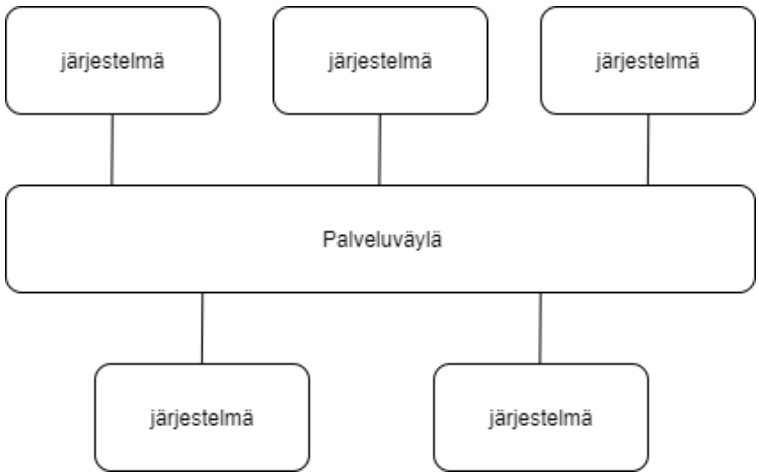
Edistyneemmässä tavassa toteuttaa puhelinkeskusmalli lähdejärjestelmä lähettää tiedon keskukselle ilman tietoa vastaanottajasta. Keskus päättelee kohdejärjestelmän tai -järjestelmät omien sääntöjensä mukaan. Näitä sääntöjä kutsutaan tilauksiksi (*subscriptions*). Tällöin keskus pääsee käsiksi viestin sisältöön eikä vain otsikoihin ja reititystä voidaan tehdä sisällön perusteella. (Bussler 2003, 11.)

Tällainen keskitetty integraatioratkaisu on tehokkain tapa kontrolloida integraatioita, tällöin voidaan helposti myös monitoroida integraatiota tehokkaasti. Siinä missä pisteestä pisteeseen -integraatiomallissa järjestelmien välisten rajapintojen määrä kasvaa toisessa potenssissa, keskitetyssä mallissa rajapintojen määrä kasvaa lineaarisesti. Yksi keskitetyn puhelinkeskusmallin hyöty on arkkitehtuurin yksinkertaisuus. Mitä vähemmän integraatioratkaisussa on järjestelmien välisiä ratkaisuja, sitä helpompi yksittäisiä järjestelmiä on lisätä ja poistaa. (Tähtinen 2005, 66-67.)

Puhelinkeskusmallillakin on kuitenkin heikkoutensa. Keskitetty arkkitehtuuri on tehokas vain tiettyyn pisteeseen asti. Kaikkea tietoa ei voida viedä yhden pisteen kautta. Luotettavuus ja vikasietoisuus muodostuvat puhelinkeskusmallissa myös ongelmalliseksi. Kun kaikki tieto kuljetetaan keskuksen kautta, keskuksen vikaantuminen voi pahimmillaan johtaa koko yrityksen toiminnan lamaantumiseen. Lisäksi yrityksen kasvaessa ja sitä myöden puhelinkeskusintegraatiototeutuksen tehon laskiessa voidaan päätyä ristiriitatilanteeseen, jossa pienemmän järjestelmän on vaara jäädä kokonaan integraatiototeutuksen ulkopuolelle. (Tähtinen 2005, 143.) Vaikka integraatioratkaisu jakaakin kaikille yhteisen tietovaraston, kulkevat tiedot silti vain kahden järjestelmän välillä. Puhelinkeskusmallissa tietojen yhdistäminen useasta lähdejärjestelmästä ei ole mahdollista. (Bussler 2003, 12.)

2.1.3 Palveluväylä

Palveluväylä voidaan ymmärtää monella tavalla ja käsitteenä se voi olla hieman hämmentävä. Palveluväylä-nimitystä käytetään yleensä arkkitehtuurisesta mallista (kuvio 4), joka kuvailee joustavaa ja rakentavaa lähestymistapaa integraatiohaasteisiin. Palveluväylän toteutus voidaan tehdä erilaisilla integraatiotuotteilla, joista ServiceMix on vain yksi esimerkki. (Dirksen & Rademakers 2008.)



KUVIO 4. Palveluväylän arkkitehtuurikuva, joka piilottaa väylän kompleksisuuden

Palveluväylä käyttää hajautettua arkkitehtuuria eli keskitetyn arkkitehtuurin heikkoudet eivät samalla tavalla vaivaa palveluväylätoteutuksia. Lisäksi palveluväylät perustuvat avoimille standardeille, kuten verkkopalvelustandardeille. Palveluväylää tulisi harkita, kun toimintaympäristö on hyvin heterogeeninen, eli varsinkin silloin, kun järjestelmissä on paljon eri teknologioita tai protokollia käytössä. (Dirksen & Rademakers 2008.)

Palveluväylän päärooli on luoda yhteistoimivuutta. Palveluväylän tehtäviin kuuluu yhteyden tarjoaminen, tietojen muunnokset, reititys, turvallisuus, luotettavuus, palveluiden hallinta ja monitorointi sekä lokitus. (Josuttis 2007.)

Tietojen muunnos on yksi palveluväylän tärkeimmistä tehtävistä. Monet järjestelmät käyttävät omaa tietomallia kuvaillessaan tietojaan. Tietomallit poikkeavat toisistaan, vaikka kuvailtava tieto järjestelmien kesken olisi samankaltaista. Palveluväylän ideana on, että jokaiselle palveluväylään liitetylle järjestelmälle on saatavilla muunnospalvelu, joka itsessään sijaitsee väylän sisällä. (Chappell 2005.)

Siinä missä puhelinkeskusmalli jakoi yhteisen tietovaraston, mutta säilytti tiedon kulun kahden järjestelmän välillä, voidaan palveluväylämallissa saavuttaa oikeasti monen lähdejärjestelmän tiedon yhdistäminen. Myös reaaliaikaisuutta ja tapahtumapohjaista viestinvälitystä pidetään palveluväylätoteutusten hyvinä puolina. (Chappell 2005.)

Palveluväylätuotteet nähdään nykyään kuitenkin suurina ja raskaina. Etenkin konfiguraation suuri määrä ja tuotteiden historiallinen painolasti nähdään erityisen haitallisena. (Vähimaa 2017.) Futuricen teknologiajohtaja Ville-Matti Toivonen toteaa: ”Olen havainnut useammassa eri toimialojen isossa firmassa, miten esimerkiksi Oraclen erinäiset ratkaisut nähdään hyvin vanhanaikaisina ja hankalina”. (Vähimaa 2017.) Toivonen jatkaa palveluväylän ongelmista: ”Arkkitehtien kalvoilla esb näyttäytyy kauniin yksinkertaisena laatikkona, jonka kautta yhteydet kulkevat. Todellisuudessa luodaan vain iso pullonkaula integraatioille”. (Vähimaa 2017.)

2.1.4 Mikropalvelut

Mikropalvelut ja niihin perustuva arkkitehtuuri on tämän päivän muotisana. Mikropalvelut ovat sovelluksia, jotka ovat vain yhtä tarkoitusta varten suunniteltu. Mikropalveluarkkitehtuuri nojaa vahvaan ja tarkkaan dokumentaatioon sekä avoimiin ja helppokäyttöisiin rajapintoihin. Arkkitehtuurikuvana mikropalvelut näyttävät paljon pisteestä pisteeseen -integraatiokokonaisuudelta. (Vähimaa 2017)

Mikropalveluarkkitehtuuria ohjaa kolme käsitettä: isojen järjestelmien pilkkominen, päämäärätietoisuus ja vaihdettavuus. Mikropalveluarkkitehtuuri on sopiva nimenomaan isoille järjestelmille, jotka kasvavat liian suuriksi alkuperäiseen tarkoitukseen nähden. Liian suurta järjestelmää on vaikea muuttaa jälkikäteen, siksi olisi fiksumpaa tehdä niistä alun perinkin pienempiä vain tietyn päämäärän tavoittamiseksi. Palveluiden vaihdettavuus tekee mikropalveluarkkitehtuurista erilaisen verrattuna muihin arkkitehtuureihin. Vaihdettavuus tarkoittaa sitä, että komponenttien säilyttämisen tai ylläpitämisen sijaan vaihdetaan ne uuteen, mikäli tarvetta ilmenee. (Amundsen, McLarty, Mitra & Nadareishvili 2016.)

Palveluväyläkokonaisuuden käyttöönottoprojektit ovat raskaita ja valtavia projekteja, kun taas mikropalveluarkkitehtuurissa jokainen palvelu voidaan määritellä ja kehittää omana tarkkarajaisena kokonaisuutenaan. Mikropalvelut eivät myöskään ole kytköksissä lainkaan toisiinsa niin, että jos yksi palvelu vioittuu tai kaatuu, eivät muut palvelut häiriinny siitä. Mikropalvelut tuovat mukanaan paljon standardinmukaisuutta ja helppoutta, kun vanhat xml-pohjaiset rajapinnat vaihdetaan json-pohjaisiin REST-rajapintoihin. Json, *JavaScript Object Notation*, on kevyt ja yksinkertainen dataformaatti, joka on myös ihmisille helppolukuista. (Vähimaa 2017.) REST on rajapintojen arkkitehtuurimalli, joka määrittelee, millaisilla operaatioilla tietoja pyydetään, lisätään ja käsitellään. REST ei kuitenkaan ole standardi, sillä se ei ota kantaa viestien formaattiin, joka voi olla esimerkiksi json tai xml. (Mikkonen 2017.)

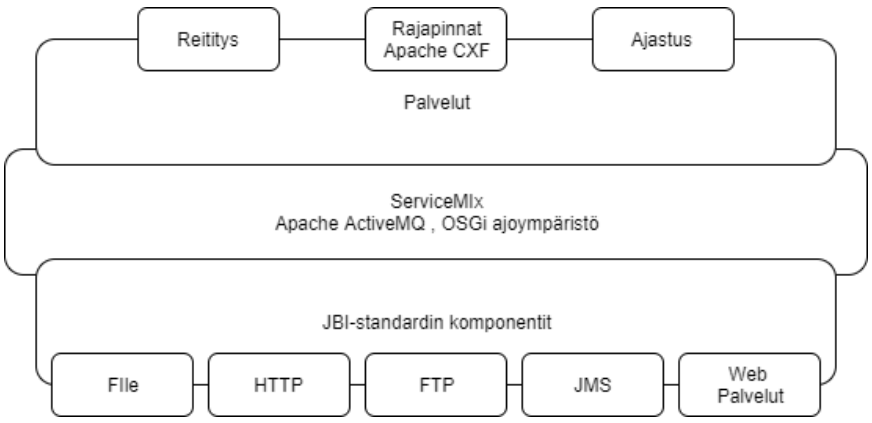
Integraatioiden kontekstissa tämä tarkoittaa menemistä takaisin pisteestä pisteeseen -integraatiototeutuksiin. Vaikka mikropalveluarkkitehtuuri tuo mukanaan paljon hyvää, iso integraatiokokonaisuus kohtaa saman hallitsemisen ja hahmottamisen vaikeuden kuin pisteestä pisteeseen -integraatiokokonaisuus. (Vähimaa 2017.)

Mikropalveluarkkitehtuuri on kuiteinkin vaikuttanut jo paljon integraatioiden toteuttamiseen, esimerkiksi monet palveluväylätuotteet ovat muuttuneet teknologioiltaan modernimpaan ja ketterämpään muotoon. Myös erilaisia iPaaS-palveluita on tullut markkinoille. IPaaS tarkoittaa yritykseltä palveluna ostettavaa integraatioalustaa. (Vähimaa 2017.) Tällöin voidaan esimerkiksi ostaa palveluväylän alusta ja ylläpito yritykseltä pilvipalveluna, jolloin asiakkaalle jää vähemmän huolehdittavaa.

2.2 ServiceMix-teknologia

ServiceMix on Apachen lisenssin alla oleva avoimen lähdekoodin integraatiotuote, joka yhdistää Apachen Karaf-ajoympäristön, Camel-integraatioreitit, CXF-REST rajapinnat ja ActiveMQ-viestijonot (ServiceMix). ServiceMix on yksi tapa toteuttaa integraatioiden palveluväylämalli (Vähimaa 2017). Apache Camel ja ServiceMix ovat johtavia integraatiotekniikoita, jotka tukevat monimutkaista reititystä (Konsek 2013).

ServiceMix toteuttaa *Java Business Integration* (JBI) -standardin. JBI määrittää arkkitehtuuristandardit, joita käytetään Java-pohjaisten integraatiotuotteiden pohjana. Standardi on tarkoitettu varsinkin palveluväylätuotteille. ServiceMixin lisäksi esimerkiksi OpenESB ja PEtALS ovat avoimen lähdekoodin tuotteita, jotka toteuttavat JBI-standardin. Kokonaisuutena ServiceMix piilottaa (kuvio 5) hyvin JBI-standardin luoman mutkikkuuden ja tarjoaa helpomman tavan konfiguroida ja ottaa käyttöön palveluita. (Dirksen & Rademakers 2008.)



KUVIO 5. Kaavio ServiceMixin arkkitehtuurista

ServiceMix on muokattu Apache Karaf -ympäristö. Eli silloin kun ServiceMix ladataan koneelle, asennetaan käytännössä Karaf. (Konsek 2013.) Karaf on alusta, johon yksittäiset sovellukset voidaan käynnistää. (Apache Karaf Container 4.x – Documentation 2019.)

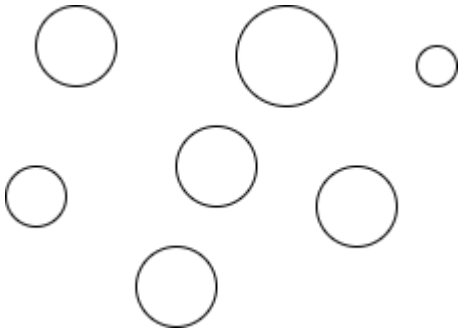
2.2.1 Karaf-ympäristö

Karaf on OSGi-pohjainen ajoympäristö (Cummins & Ward 2013). Jotta ymmärretään, mitä vaatimuksia OSGi-pohjaisuus asettaa toteutettaville sovelluksille, käydään seuraavaksi kevyesti läpi, mitä OSGi tarkoittaa ja mitä sillä saavutetaan.

Nimitys OSGi tulee sanoista *Open Services Gateway iniative*, ja on alun perin osa avoimen standardin organisaation nimeä. Organisaatio nimeltä OSGi Alliance on vuonna 1999 perustettu voittoa tavoittelematon yhdistys, joka pyrkii kehittämään ja ylläpitämään teknologiaa, joka parantaa Javan modulaarisuutta. (OSGi Alliance 2019.)

Nykyaikaisessa ohjelmistosuunnittelussa yksi tärkeimmistä tavoitteista on modulaarisuus. Se vähentää vaivaa vähentämällä kopioidun koodin määrää ja parantaa stabiiliutta. Viime aikoina oliopohjaisuus on vahvasti ohjannut ohjelmointikielten kehitystä. Java on oliopohjainen ohjelmointikieli. Oliot ovat hyvin kapseloituja

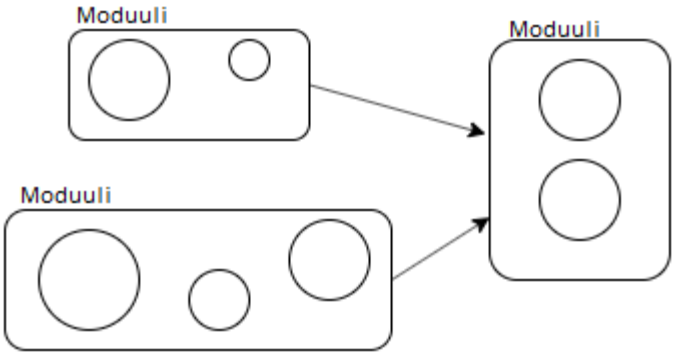
kokonaisuuksia, jotka voivat pitää tietoa itsellään yksityisten metodien kautta. Erillisten olioiden lisäksi Java ei tarjoa minkäänlaista rakennetta. Oliot heitetään kasaan ja jokainen olio voi vuorovaikuttaa toisen olion kanssa (kuvio 6). (Cummins & Ward 2013.)



KUVIO 6. Vapaasti vuorovaikuttavat Java oliot.

Ongelmaa pahentaa se, että harva Java-sovellus on itsenäinen. Useimmiten sovellukset riippuvat muista kirjastoista ja kehyksistä, ja näiden riippuvuuksien selvittäminen voi aiheuttaa ongelmia. Jos riippuvuudet lisätään ohjelman JAR-pakettiin mukaan, voidaan varmistaa, että ohjelman tarvitsemat paketit löytyvät. Riskinä tässä tilanteessa on se, että riippuvuudet ovat niin yleiset, että toinen sovellus tarvitsee saman riippuvuuden eri versiota. Tällöin päädytään tilanteeseen, jossa toinen sovelluksista käyttää väärää versiota riippuvuudesta, eikä siis välttämättä toimi oikein. (Cummins & Ward 2013.)

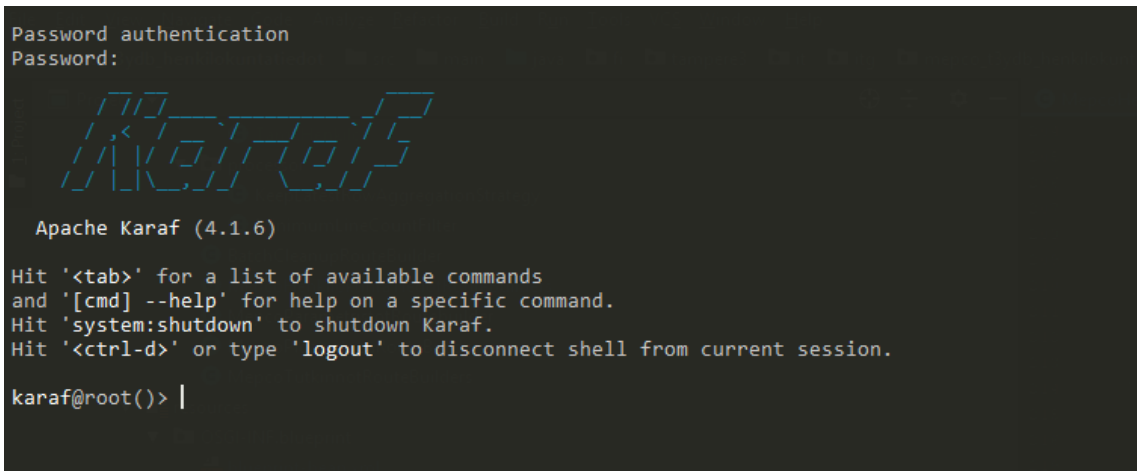
OSGissa usein puhutaan moduuleista, *bundleista,* jotka ovat JAR-tiedostoja, joiden manifesteihin ja otsikoihin, *headereihin,* on kirjoitettu lisätietoja. OSGi-ajoympäristö, kuten Karaf, huolehtii moduulien elinkaaresta ja toiminnoista. Ajoympäristössä moduulin sisällä olevat luokat voivat tavalliseen tapaan käyttää toisiensa metodeja, mutta ne eivät voi käyttää toisessa moduulissa olevaa luokkaa, ellei se ole erikseen sallittua. Tämän avulla luokkien näkyvyys voidaan siis rajoittaa julkisiin ja yksityisiin luokkiin (kuvio 7). (Cummins & Ward 2013.) Samaan tapaan metodit ja muuttujat voidaan jakaa julkisiin ja yksityisiin luokkien sisällä.



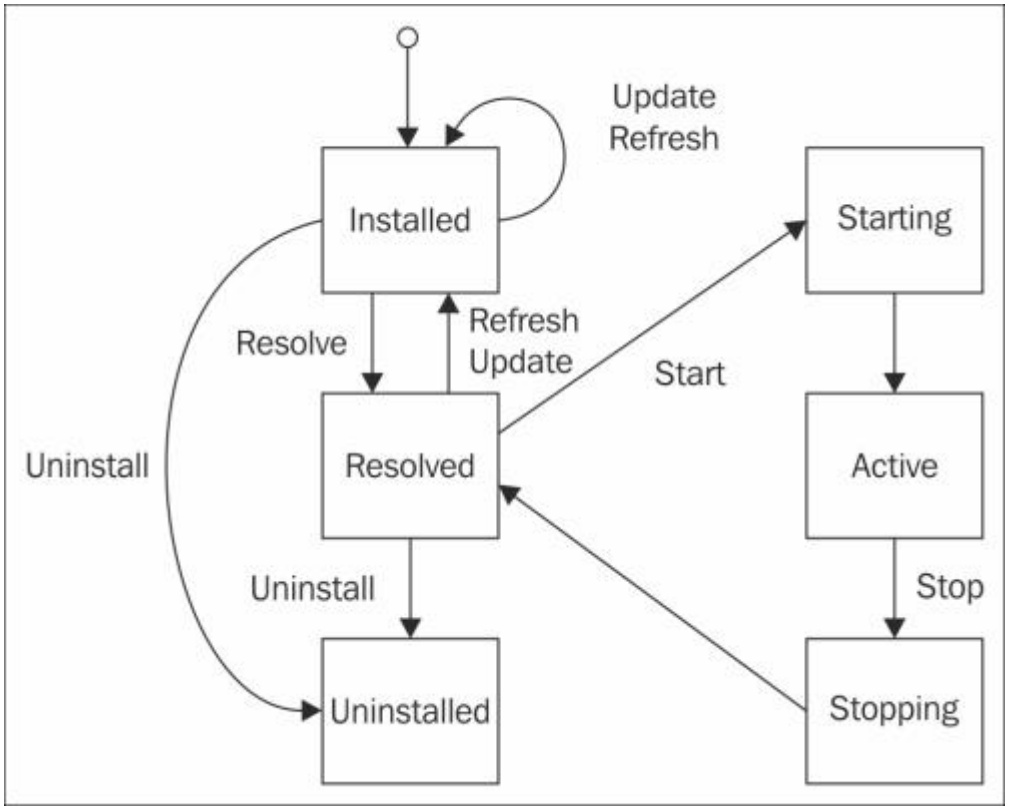
KUVIO 7. Oliot moduulien sisällä. Riippuvuudet moduulien välillä näkyvät selvästi.

Jotta moduulit voivat käyttää toistensa luokkia voidaan tietyt paketit, *packages,* viedä, *export,* muille käytettäväksi. Jotta toinen moduuli voisi käyttää vietyjä paketteja, täytyy sen tuoda, *import*, luokat, joita se käyttäisi. (Cummins & Ward 2013.)

Karaf voidaan käynnistää Karafin kansiossa komennolla **bin\karaf.bat**, joka avaa Karafin konsoli-ikkunan (kuva 1). Kaikki Karafiin ladatut sovellukset noudattavat OSGi-elinkaarta. Kuviossa 8 on esitetty tilat, jossa sovellus voi olla ja komennot, joilla siirrytään tilasta toiseen. (Edstrom, Kesler & Goodyear 2013.) Elinkaarenhallinnan lisäksi Karafin komentoriviltä voi tarkistaa lokit, **log:tail** tai **log:display** -komennoilla.



KUVA 1. Kuvakaappaus Karafin komentoikkunasta



KUVIO 8. OSGi moduulin elinkaaritoiminnot. (Edstrom ym. 2013.)

Karafin voi jättää vahtimaan Maven-projektia **bundle:watch** komennolla. Kun projektiin tehdään muutoksia ja se käännettään **mvn install** komennolla, muutokset ladataan Karafiin ja sovellus käynnistetään uudestaan. (Konsek 2013.) Kun Maven-projekti on käännetty voidaan se asentaa Karafiin käskyllä **bundle:install -s mvn:groupId/artifactId/version**, jossa groupId on Maven projektin ryhmätunnus, joka on usein muotoa com.apache.mave, artifactId on sovelluksen nimi ilman versionumeroa ja version on projektin versionumero. (Edstrom ym. 2013.) Kaikki Karafiin asennetut paketit voi listata **list**-komennolla, joka näyttää myös pakettien tämän hetkisen tilan (kuva 2).

Kuvasta 2 voidaan siis lukea, että tähän Karafiin on asennettu yksi integraatio, jonka nimi on ”T3 in_out_test integration”, jonka versionumero on 1.0.0-SNAPSHOT. ”SNAPSHOT” integraation versionumerossa tarkoittaa sitä, että integraatio on vielä kehityksessä. Kun integraatiosta tehdään julkaisu ”SNAPSHOT” tiputetaan versionumerosta pois, eli tämän integraation seuraava versio tulee olemaan 1.0.0. Integraatio on tilassa *resolved*, joten se on onnistuneesti asennettu, mutta ei käynnissä. Muut asennetut paketit ovat muun muassa Camelin komponentteja ja tietokantayhteyden mahdollistavia kirjastoja.

```
karaf@root()> list
START LEVEL 100 , List Threshold: 50
 ID | State    | Lvl | Version            | Name
----|----------|-----|--------------------|-----------------------------------------------
 29 | Active   |  80 | 4.1.6              | Apache Karaf :: OSGi Services :: Event
202 | Resolved |  80 | 1.0.0.SNAPSHOT     | T3 in_out_test integration
684 | Active   |  80 | 1.9.2.1            | Apache ServiceMix :: Bundles :: jasypt
685 | Active   |  80 | 1.5.0              | OPS4J Base - Service Provider Access
686 | Active   |  80 | 1.1.0              | OPS4J Pax JDBC Generic Driver Extender
687 | Active   |  80 | 1.1.0              | OPS4J Pax JDBC Config
688 | Active   |  80 | 1.1.0              | OPS4J Pax JDBC Pooling Support Base
689 | Active   |  80 | 1.0.0.201505202023 | org.osgi:org.osgi.service.jdbc
694 | Active   |  80 | 2.8.9              | Jackson datatype: JSR310
698 | Active   |  80 | 7.0.0              | Microsoft JDBC Driver for SQL Server
699 | Active   |  50 | 1.5.5              | JavaMail API
700 | Active   |  80 | 1.5.5              | T3 Integrations Parent :: Utilities
702 | Active   |  80 | 3.0.0              | Expression Language 3.0 API
703 | Active   |  80 | 1.2.0              | CDI APIs
704 | Active   |  80 | 1.2                | javax.interceptor API
706 | Active   |  80 | 1.2                | javax.transaction API
708 | Active   |  50 | 5.15.2             | activemq-camel
709 | Active   |  50 | 5.15.2             | activemq-osgi
717 | Active   |  80 | 1.1.1              | Apache Aries Transaction Blueprint
718 | Active   |  80 | 2.1.0              | Apache Aries Transaction Blueprint
719 | Active   |  80 | 1.3.3              | Apache Aries Transaction Manager
720 | Active   |  50 | 2.19.5             | camel-bindy
721 | Active   |  50 | 2.19.5             | camel-blueprint
722 | Active   |  80 | 2.19.5             | camel-commands-core
723 | Active   |  50 | 2.19.5             | camel-core
724 | Active   |  50 | 2.19.5             | camel-csv
725 | Active   |  50 | 2.19.5             | camel-cxf
726 | Active   |  50 | 2.19.5             | camel-cxf-transport
727 | Active   |  50 | 2.19.5             | camel-freemarker
728 | Active   |  50 | 2.19.5             | camel-ftp
729 | Active   |  50 | 2.19.5             | camel-http-common
730 | Active   |  50 | 2.19.5             | camel-http4
731 | Active   |  50 | 2.19.5             | camel-jackson
732 | Active   |  50 | 2.19.5             | camel-jaxb
733 | Active   |  50 | 2.19.5             | camel-jdbc
734 | Active   |  50 | 2.19.5             | camel-jms
735 | Active   |  50 | 2.19.5             | camel-ldap
736 | Active   |  50 | 2.19.5             | camel-mail
737 | Active   |  50 | 2.19.5             | camel-spring
738 | Active   |  50 | 2.19.5             | camel-sql
739 | Active   |  50 | 2.19.5             | camel-zipfile
740 | Active   |  80 | 2.19.5             | camel-karaf-commands
741 | Active   |  80 | 1.11.0             | Apache Commons Codec
742 | Active   |  80 | 4.2.0              | Apache Commons Collections
743 | Active   |  50 | 1.4.0              | Apache Commons CSV
744 | Active   |  80 | 2.1.1              | Apache Commons DBCP
745 | Active   |  50 | 3.6.0              | Apache Commons Net
746 | Active   |  50 | 1.6.0              | Commons Pool
747 | Active   |  50 | 2.4.2              | Apache Commons Pool
764 | Active   |  80 | 1.0.2              | Apache Felix Coordinator Service
767 | Active   |  50 | 1.0.1              | geronimo-j2ee-management_1.1_spec
771 | Active   |  50 | 3.4.6              | ZooKeeper Bundle
772 | Active   |  50 | 4.5.3              | Apache Apache HttpClient OSGi bundle
773 | Active   |  50 | 4.4.6              | Apache Apache HttpCore OSGi bundle
775 | Active   |  80 | 4.1.6              | Apache Karaf :: JDBC :: Core
780 | Active   |  80 | 2.1.2              | Apache XML Security for Java
782 | Active   |  80 | 3.2.4.1            | Apache ServiceMix :: Bundles :: cglib
783 | Active   |  80 | 2.4.0.6            | Apache ServiceMix :: Bundles :: commons-lang
785 | Active   |  50 | 2.3.23.1           | Apache ServiceMix :: Bundles :: freemarker
786 | Active   |  80 | 1.0.0.2            | Apache ServiceMix :: Bundles :: javax.inject
789 | Active   |  50 | 0.1.54.1           | Apache ServiceMix :: Bundles :: jsch
790 | Active   |  80 | 3.17.0.1           | Apache ServiceMix :: Bundles :: poi
853 | Active   |  50 | 0.6.4              | JAXB2 Basics - Runtime
854 | Active   |  80 | 1.5.9.0            | MariaDB JDBC Client
855 | Active   |  80 | 1.1.0              | OPS4J Pax JDBC MariaDB Driver Adapter
856 | Active   |  80 | 1.1.0              | OPS4J Pax JDBC MSSQL Driver Adapter
857 | Active   |  80 | 1.1.0              | OPS4J Pax JDBC Oracle Driver Adapter
```

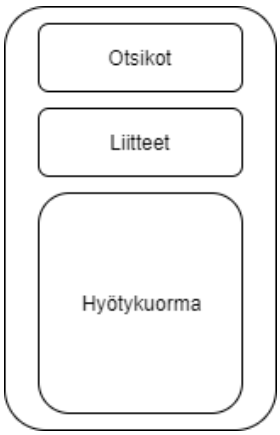
KUVA 2. Kuvakaappaus Karafin konsolista, johon on listattu kaikki Karafiin asennetut paketit.

2.2.2 Camel-reitit

ServiceMixin kanssa viestien reititykseen voidaan käyttää joko Camelia tai EIP Service-engineä. Lisäksi Camelilla reitit voidaan konfiguroida käyttämällä XML- tai Java-syntaksia. (Dirksen & Rademakers 2008.) Tampereen korkeakouluyhteisössä reititys tehdään Camelin Java-syntaksilla.

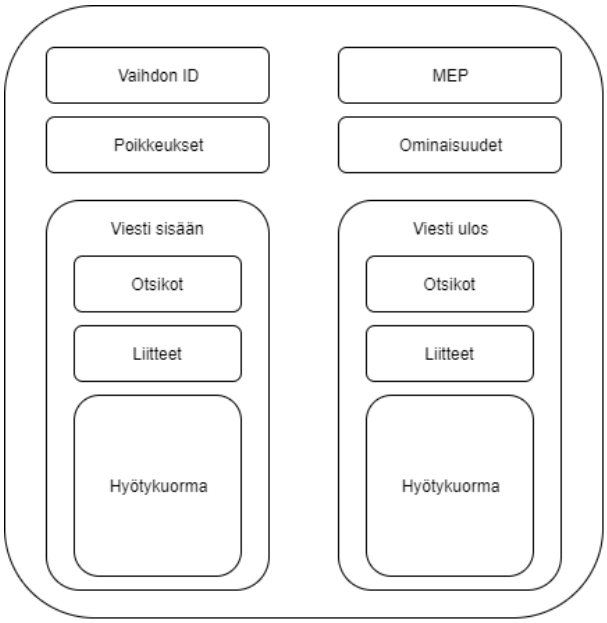
Camel on avoimen lähdekoodin integraatiokehys, joka keskittyy integraatioiden yksinkertaistamiseen. Camel projekti alkoi 2007, se on lisensoitu Apache 2 -lisenssillä ja sillä on aktiivinen kehittäjäyhteisö. (Anstey & Ibsen 2018.) Kun puhutaan integraatioreitityksistä, useimmiten viitataan alan perusteokseen: Gregor Hohpen ja Bobby Woolfin kirjoittamaan *Enterprise Integration Patterns.* Kirjassa esitellään 65 mallia viestien reititykseen (Hohpe & Woolf 2003,10). Tällaisia reititys malleja ovat esimerkiksi sisältöpohjainen reititys, jossa viestin sisältö määrää minne viesti lähetetään, ja jakaja, joka jakaa ison viestin pienempiin, jotka voidaan reitittää edelleen (Hohpe & Woolf 2003, 209). Camelissa suurimmalle osalle reititys malleista löytyy valmiit komponentit niiden toteuttamiseen (Apache Camel).

Integraatioreiteillä kulkevat viestit, *message,* ovat kokonaisuuksia, joita järjestelmät käyttävät keskustellessaan keskenään. Viestit (kuvio 9) koostuvat otsikoista, *headers,* mahdollisista liitteistä, *attachments,* ja tietosisällöstä, *body* tai *payload*. Otsikoihin voidaan kirjoittaa avain-arvo-pareja, joihin kirjoitetaan viestiin liittyviä arvoja kuten lähettäjä tai autentikointi tietoja. Otsikon avain on uniikki ja se on tyypiltään Javan String-muuttuja. Otsikon arvo on Javan Object-tyyppinen, joten monenlaisia arvoja voidaan viedä otsikoissa. Otsikoiden koolle tai määrälle ei Camelissa ole määritelty minkäänlaisia rajoitteita. Viesti voi myös sisältää liitteitä, joita käytetään usein vain nettipalveluissa tai sähköpostikomponenteissa. Camelin hyötykuorma on Object-tyyppinen, joten se voi kantaa lähes millaista tietoa tahansa. Integraation suunnittelijan tai toteuttajan vastuulla on pitää huolta, että vastaanottava järjestelmä ymmärtää välitettävän datamallin. (Anstey & Ibsen 2018.)



KUVIO 9. Camelin käyttämän viestin anatomia (Anstey & Ibsen 2018, muokattu).

Camelin reiteillä viestit on pakattu *exchange*-olion sisälle (kuvio 10). Tässä opinnäytteessä exchange-oliosta käytetään nimeä vaihto-olio. Camel generoi automaattisesti jokaiselle vaihto-oliolle uniikin tunnisteen. MEP tulee sanoista *message exchange pattern* ja se kertoo, millaista prosessointia ollaan tekemässä. Camelissa voidaan tehdä sekä yksisuuntaisia että pyyntö-vastaus-tyyppisiä prosessointeja. Jos prosessointi on yksisuuntaista, vaihto-olio sisältää vain sisään tulevan viestin, muuten se sisältää myös ulos vietävän viestin. Jos reitillä tapahtuu virhe tai poikkeus, se tallennetaan vaihto-olion tietoihin. Vaihto-olioon tallennetaan myös ominaisuuksia, *properties,* jotka ovat samankaltaisia avain-arvo-pareja kuin otsikotkin, mutta ne pysyvät tallessa koko olion elinkaaren ajan. Camel kirjoittaa jotain tietoja automaattisesti ominaisuuksiin, mutta myös kehittäjä voi tallentaa ja hakea tarvittavia attribuutteja ominaisuuksiin. (Anstey & Ibsen 2018.)



KUVIO 10. Kaavio vaihto-oliosta, johon Camel paketoi viestit (Anstey & Ibsen 2018, muokattu).

Reititys on Camelin keskeinen abstraktio. Pellin alla toimii Camelin reititysmoottori, joka pitää huolta, että viestit reititetään oikein. Reititysmoottori ei ole integraatiokehittäjälle näkyvillä, mutta on hyvä olla tietoinen siitä, että yksinkertaisen syntaksin takana voi tapahtua monimutkaisia asioita. (Anstey & Ibsen 2018.)

Yksinkertaisimmillaan reitti voidaan määritellä ketjuksi prosessoreita. Jokaisella reitillä on uniikki tunniste, jota käytetään lokitukseen, virheenkorjaukseen, monitorointiin, reittien käynnistykseen ja sammutukseen. Reiteillä on aina tasan yksi viestien tulolähde, kuitenkin esimerkiksi kolmen viestijonon lukeminen on mahdollista määritellä yhdeksi reitiksi Java-syntaksilla seuraavasti: from("jms:queue:A", "jms:queue:B", "jms:queue:C").to("jms:queue:D");, sillä reititysmoottori hajottaa tässä tapauksessa reitin kolmeksi samanlaiseksi reitiksi. Monen tulolähteen määrittely on sallittua Camelin 2. versiossa, mutta se ei ole suositeltavaa, sillä se todennäköisesti poistetaan Camelin seuraavaan pääversioon. (Anstey & Ibsen 2018.)

Reitti alkaa aina kuluttajalla, *consumer,* joka luo alkuperäisen viestin ja vaihdon. Kuluttaja on palvelu, joka ottaa vastaan ulkoisen järjestelmän tuottaman viestin, paketoi sen vaihdon sisään ja lähettää prosessoitavaksi. Camelissa on tapahtumapohjaisia ja kyselypohjaisia kuluttajia. Tapahtumapohjaiset kuuntelevat tiettyä tulolähdettä viestien varalta ja reagoivat tuleviin viesteihin, kun taas kyselypohjaiset kuluttajat aktiivisesti tarkistavat ja hakevat viestit. (Anstey & Ibsen 2018.)

Reittien muodostamiseen Camelissa käytetään DSL (*Domain-specific language*) syntaksia. Java DSL:ää käytettäessä reitit voidaan kirjoittaa suoraan Java-luokkaan (kuva 3), joka laajentaa, *extend,* Camelin RouteBuilder-luokkaa. Mikään sovellus ei ole vain kasa Java-luokkia vaan ne pitää yhdistää. OSGi ympäristössä Java-luokat yhdistää OSGi Blueprint, mikä on xml-tiedosto, jossa määritellään *beaneja*. Kun integraatio asennetaan OSGi-ympäristöön, Blueprint kertoo ympäristölle, minkälaisia Java-olioita integraatiota varten luodaan. Näin voidaan samasta Java-luokasta luoda useita reittejä, jotka kuitenkin toimivat toisistaan riippumatta. Jotta reitit käynnistyvät, reittibeanit lisätään CamelContextin sisälle (kuva 4). (Anstey & Ibsen 2018.)

```
import org.apache.camel.builder.RouteBuilder;

public class TestRouteBuilder extends RouteBuilder {

    public void configure() throws Exception {
        from("scheduler:foo?scheduler=spring&scheduler.cron=*/10 * * * * *")
            .routeId("in-out-test-rt")
            .log("I'm logging every ten seconds!");
    }
}
```

KUVA 3. Kuvakaappaus Java-reitistä, joka käyttää kyselypohjaista kuluttajaa ja lokittaa 10 sekunnin välein lauseen ”I’m logging every ten seconds!”

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0"
           xsi:schemaLocation="
         http://www.osgi.org/xmlns/blueprint/v1.0.0 https://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
         http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-blueprint.xsd
         http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0 http://aries.apache.org/schemas/blueprint-cm/blueprint-cm-1.1.0.xsd">

    <bean id="test" class="fi.tampere3.it.itg.TestRouteBuilder"/>

    <camelContext id="camelContext" xmlns="http://camel.apache.org/schema/blueprint">
        <routeBuilder ref="test" />
    </camelContext>
</blueprint>
```

KUVA 4. Kuvakaappaus blueprint.xml- tiedostosta, jossa on määritelty reittibean nimeltä test ja se on asetettu aktiiviseksi reitiksi CamelContextin sisälle.

3 TAMPEREEN KORKAKOULUYHTEISÖN INTEGRAATIOPALVELU

3.1 Palvelun toiminta

Tampereen korkeakouluyhteisön integraatiopalvelun tehtävänä on ylläpitää ja kehittää Tampereen korkeakouluyhteisön palveluväylää ja integraatioalustaa, joiden välityksellä voidaan keskitetysti jakaa korkeakouluyhteisön ydintietoja. Integraatiopalvelut myös vastaavat integraatioiden toteutuksesta. Samalla pyritään keskittämään integraatio-osaamista yhden palvelun taakse.

Integraatiopalveluissa toimii projektipäällikön ja tuoteomistajan lisäksi seitsemän integraatiokehittäjää, joista osa on korkeakouluyhteisön ja osa korkeakouluyhteisön kumppaneiden henkilöstöä. Lisäksi osa integraatiopalveluiden henkilöistä toimii integraatioasiantuntijana, joiden tehtäviin kuuluu muun muassa määrittelyyn osallistuminen, määrittelyn kuvauksen tekeminen ja testauksen suunnittelu ja toteuttaminen (Rönkä 2019d). Integraatioasiantuntijan tilalla voi myös olla integraatiokehittäjä, jos nähdään, että kehittäjällä on tarpeellista osa-alueen osaamista.

Integraatiopalvelu voidaan jakaa karkeasti kehitykseen, testaukseen ja ylläpitoon. Ylläpidon työnä on ylläpitää tuotettuja integraatioita ja kehittää integraatioalustaa. Testaus vastaa integraatioiden testauksesta aina uuden integraation tai integraatiomuutosten valmistuessa. Kehityksen työt koostuvat integraatioiden ja muutosten toteuttamisesta ja virheiden korjaamisesta.

Kehitystä johdetaan ketterän kehityksen periaattein Scrumilla, niin että työ on jaettu kahden viikon sprintteihin, joiden välissä pidetään Scrum-menetelmän peruskokoukset eli sprintin katselmus, retrospektiivi ja seuraavan sprintin suunnittelu (The Scrum Guide™ 2018). Kehityksen tehtävät sitoutuvat vahvasti asiakkaiden toteutus- tai muutostilauksiin ja kehityksen kehitysjono syntyy tilauskirjan perusteella. Kiireelliset virheiden korjaustehtävät voidaan tuoda kehitykselle korjattavaksi myös kesken sprintin, jotta ne saataisiin mahdollisimman vähillä vaikutuksilla korjatuksi.

3.2 Integraatiopalveluväylä

Tampereen korkeakouluyhteisön integraatiopalveluväylä on ServiceMix-teknologiaan pohjautuva integraatioväylä, joka on tarkoitettu korkeakouluyhteisön integraatiotarpeita varten. Palveluväylällä käytetään ajoympäristönä Karafia ja reititykseen Camelia.

Palveluväylästä on kolme vain hieman toisistaan eroavaa ympäristöä, jotka palvelevat integraatiopalvelun eri osa-alueita. Kehitysympäristö on nimenmukaisesti tarkoitettu kehitystä varten. Kehitysympäristöissä voidaan testata integraation käynnistymistä ja perusominaisuuksia tuotantoympäristön kaltaisessa ympäristössä ilman pelkoa siitä, että virheet tai muut kehityksen aikaiset kokeilut häiritsevät oikeita käynnissä olevia integraatioita. Testiympäristö on testausta ja varsinkin hyväksymistestausta varten. Integraatiopalvelun testiympäristössä testit tehdään aina anonymisoidulla tai tekaistulla datalla. Tuotantoympäristössä on asennettuna ne integraatiot, jotka välittävät oikeita tietoja järjestelmältä toiselle. Tuotantoympäristöön on rajatuimmat pääsyoikeudet vahinkojen ja muiden virheiden minimoimiseksi.

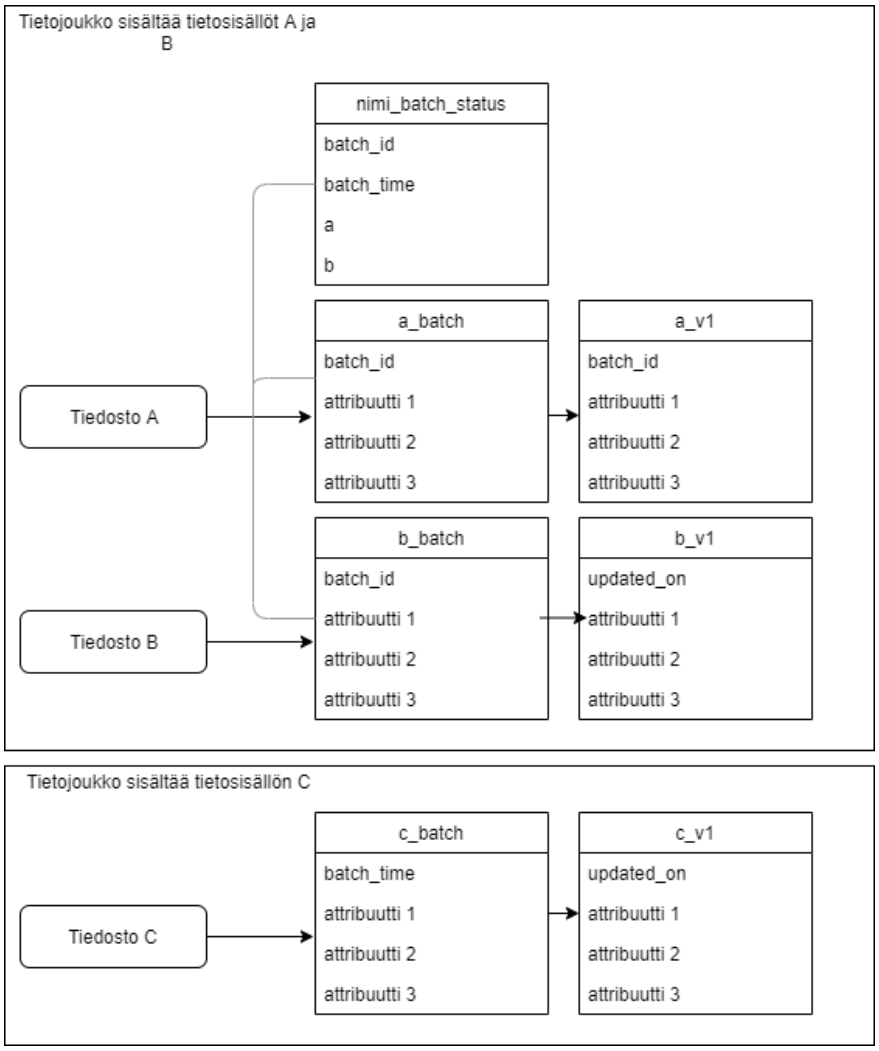
Integraatioväylä myös piilottaa sisäänsä välimuistitietokannan ja SFTP-palvelimen, minne toiset järjestelmät voivat tuoda tiedostoja, tai integraatiot voivat viedä tiedostoja järjestelmille luettavaksi. Myös välimuistitietokannasta on testi- ja tuotantoversiot.

3.2.1 Välimuistitietokanta

Välimuistitietokannan tarkoituksena on tehostaa tiedostopohjaisten tiedonsiirtojen käsittelyä. Jos välimuistitietokantaa ei käytettäisi, jouduttaisiin tiedot parsimaan tiedostoista jokaisessa integraatiossa erikseen. Välimuistitietokantaa käytettäessä tiedostoina tuodut tiedot luetaan kerran vuorokaudessa tietokantaan, missä ne ovat yhtenäisessä tietomallissa integraatioiden käytettävissä. (Borodulin 2019.)

Välimuistitietokanta on tarkoitettu sellaisenaan vain integraatioväylän sisäiseen käyttöön ja siksi sinne ei saa avata pääsyä muille järjestelmille. Tietoja voidaan tarjota integraatioväylän kautta muille järjestelmille käyttöön esimerkiksi REST-rajapintoina. Välimuistitietokantaan ei pidä lukea mitään sellaista tietoa, jota ei voida uudelleen tuottaa joistain lähdejärjestelmästä. Tällä pyritään minimoimaan haitta, joka koituisi koko välimuistitietokannan menetyksestä. (Borodulin 2019.)

Jokaista sisään luettavaa tietojoukkoa kohtaan tehdään oma integraatio, joka vie tiedot välimuistitietokantaan niille määriteltyyn tietomalliin. Mikäli samasta liittymästä saadaan useita eri tietoja, niin kunkin taulukokonaisuuden sisään lukua seurataan tilannekuvataululla, jonka nimen tulee päättyä ”_batch_status” (kuvio 11). Tiedot luetaan tauluihin, joiden nimi päättyy ”_batch”. Jos yhdestä tiedostosta tulee vain yhden taulun tietoja, tulee sen tietokanta tauluun kirjoittaa aikaleima, jolloin tiedot on tuotu. (Borodulin 2019.)



KUVIO 11. Kaavio tietosisältöön liittyvästä taulurakenteesta.

Tietokantatauluille määritellään tietokantanäkymät, joita tietoja vievät integraatiot voivat käyttää. Näkymät nimetään niin, että niiden perään tulee versionumero, joka on aluksi 1 (kuvio 11). Tietotauluihin voidaan tallentaa useampi versio tiedoista, joista vain uusin viedään näkymään. Tietoja vievät integraatiot voivat käyttää näkymiä vapaasti yhdistelläkseen niistä tiedot integraatiota varten, mutta toteutuksessa täytyy muistaa, että jokainen sarake on yksilöitävä hakulausekkeessa. (Borodulin 2019.) Näin vältytään turhilta virheiltä, jos näkymään esimerkiksi lisätään sarakkeita myöhemmin.

3.2.2 Integraatioille yhteiset piirteet

Niin kuin muutkin sovellukset, integraatiot monesti riippuvat usein ulkopuolisista kirjastoista. Maven-projekteissa riippuvuudet kirjataan pom.xml-tiedostoon, jolloin Maven käy lataamassa riippuvuudet ja projektissa voidaan käyttää riippuvuuden tarjoamia luokkia. Palveluväylällä on monta integraatiota yhtä aikaa, jotka kaikki riippuvat osakseen samoista paketeista. Jotta vältyttäisiin versioristiriidoilta, pitäisi olla helppo tapa hallita riippuvuuksien versioita.

Tampereen korkeakouluyhteisössä tämä ongelma on ratkaistu sillä, että jokainen integraatio perii riippuvuutensa ja niiden versiot niin kutsutulta *Parent*-projektilta (kuva 5). Riippuvuuksien lisäksi Parentissa ylläpidetään Karaf-ympäristöön asennettavat ominaisuuspaketit. (Rönkä 2019c.) Kun Parenttiin tehdään muutoksia, siitä tehdään uusi julkaisu. Kun integraatio, joka tarvitsee Parentista uudemman version, viedään tuotantoon, asennetaan samalla myös Parent-projektista uudempi versio.

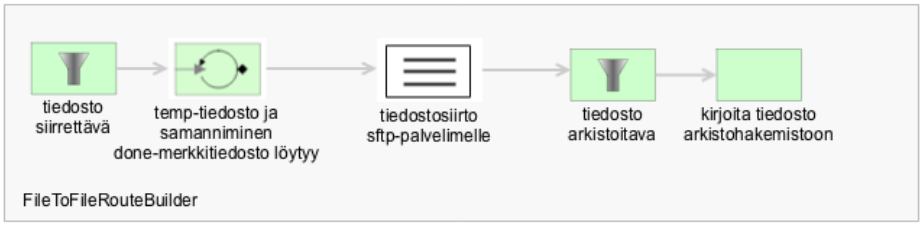
```
<parent>
    <groupId>fi.tampere3.it.itg</groupId>
    <artifactId>t3-itg-parent</artifactId>
    <version>1.4.2</version>
</parent>

<artifactId>sap_ydb_kayttooikeudet</artifactId>
<version>1.0.2-SNAPSHOT</version>
<packaging>jar</packaging>

<name>T3 sap_ydb_kayttooikeudet integration</name>
```

KUVA 5. Kuvakaappaus integraation POM.xml projektitiedostossa, josta näkyy, että integraatiolle on asetettu parent-projekti.

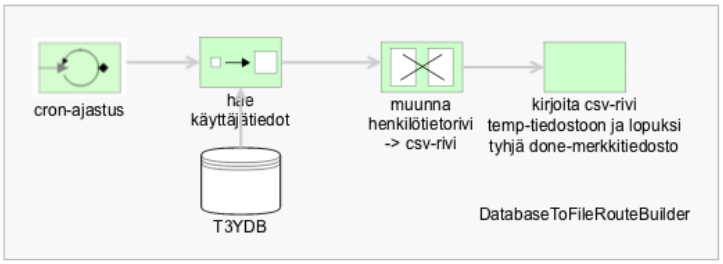
Kun projektirunko luodaan Tampereen korkeakouluyhteisön Maven-arkkityypin avulla, projekti automaattisesti saa Parent-projektiksi oikean projektin. Kehittäjälle jää tarkistettavaksi, että Parent-projektin versio on uusin versio, joka on tuotantoon asennettu. Tietysti, jos toteutettava integraation toteutus vaatii muutoksia tai lisäyksiä Parent-projektiin, niin täytyy Parentin versio nostaa myös integraatioon.

Parenttiin on myös toteutettu uudelleenkäytettäviä ominaisuuksia ja työkaluja. Parentissa on muun muassa validointityökaluja suomalaisen henkilötunnuksen validointiin ja työkaluja resursseihin kirjoitettujen SQL-tiedostojen lukemiseen. Lisäksi Parenttiin on toteutettu uudelleenkäytettäviä Java-reittejä, jotka on todettu toistuvan useissa integraatioissa. Reitit voidaan ottaa käyttöön yksittäisessä integraatiossa määrittelemällä integraation Blueprinttiin reittibean, joka käyttää Parentin Java-reittiä.

Valmisreitteinä on tällä hetkellä toteutettuna tiedostosta tiedostoksi -siirto (kuvio 12), jolla tiedosto voidaan siirtää paikallisella koneella kansiosta toiseen, tuoda tiedosto SFTP-palvelimelta, viedä tiedosto SFTP-palvelimelle tai siirtää tiedosto SFTP-palvelimelta toiselle. Blueprintissä määritellään, millainen siirto tehdään reitille annetuilla parametreilla. Valmisreitteinä löytyy myös tiedostosta tietokantaan luku, jolla voidaan viedä tiedoston sisältöä suoraan tietokantaan, sekä tietokannasta tiedostoon (kuvio 13).



KUVIO 12. Kaavio tiedon kulusta tiedostosta tiedostoon -valmisreitillä



KUVIO 13. Kaavio tiedon kulusta tietokannasta tiedostoksi -valmisreitillä

4 SAP-INTEGRAATIOT

4.1 Määrittelyn kokoaminen

Vaatimusmäärittely on usein iso kokonaisuus, jossa pyritään kartoittamaan ohjelmiston vaatimukset mahdollisimman perusteellisesti. Uuden ohjelmisto kokonaisuuden vaatimusmäärittely saatetaan lohkaista omaksi kokonaisuudekseen. Tuotettavan ohjelmiston vaatimukset muodostuvat liiketoiminnan tarpeista, ja usein niiden tärkeimmät lähteet ovat asiakas ja ohjelmiston tulevat käyttäjät. (Haikala & Mikkonen 2011, 66.)

Vaatimusmäärittely voidaan jakaa vaiheisiin vaatimusten kartoittamisesta validointiin. Vaatimusmäärittely aloitetaan yleensä vaatimusten kartoittamisesta, minkä jälkeen vaatimukset analysoidaan. Tässä kohtaa vaatimuksia voidaan vielä tarkentaa ja niiden prioriteettia ja keskinäisiä suhteita voidaan pohtia. Näin kerätyt vaatimukset dokumentoidaan sovitulla tavalla ja validoidaan. Usein validointi tarkoittaa sitä, että määrittelydokumentti katselmoidaan asiakkaan kanssa. (Haikala & Mikkonen 2011, 66.)

Hyviä vaatimusten kartoittamistapoja ovat muun muassa tulevien käyttäjien haastatteleminen, aivoriihet ja työpajat (Haikala & Mikkonen 2011, 66). Vaatimuksia voidaan tunnistaa monista eri lähteistä, joista tärkeimmät esitellään seuraavaksi. Liiketoiminnan nykytilan ymmärrys voi poikia erilaisia vaatimuksia tuotettavalle ohjelmistolle tai palvelulle. Lisäksi on hyvä tutustua ympärillä toimiviin samankaltaisiin palveluihin tai ohjelmistoihin, sillä nyt käytössä olevien ohjelmien määrittelydokumentaatio voi paljastaa uudelleen käytettävää materiaalia. Lisäksi kannattaa tutustua nykyisten toimijoiden ongelmiin ja parannusehdotuksiin. Markkinoinnin keinot, kuten markkinointitutkimukset ja käyttäjille suunnatut kyselyt, helpottavat järjestelmän määrittelyä. Välillä määrittelyä voi myös helpottaa käyttäjän seuraaminen tai jopa käyttäjän asemaan asettuminen. (Miller 2009, 32.)

Jotta haastatteluista voidaan saada kaikki hyöty irti, kannattaa muutamaan seikkaan kiinnittää erityistä huomiota. Ensinnäkin kannattaa tarkistaa, että haastateltavalla on tarpeeksi aikaa haastattelulle, jottei hänen tarvitse rientää seuraavaan

kokoukseen suoraan haastattelusta. Ensimmäinen tai viimeinen työtunti tai päivä ennen tai jälkeen loman, ei ole tyypillisesti paras haastatteluaika, sillä usein ihmiset ovat väsyneitä tai heidän on vaikea keskittyä näinä aikoina. Haastattelu on aina eräänlainen tapaaminen, joten haastattelua varten olisi hyvä valmistella esityslista, johon kirjataan tapaamisen tarkoitus, tapaamispaikka ja -aika, osallistujat, haastatteluformaatti, esitiedot, jotka osallistujien olisi hyvä tietää ennen haastattelua ja tiedot siitä, miten prosessi jatkuu haastattelun jälkeen. (Miller 2009, 59.)

Jotta käyttäjää voidaan haastatella, täytyy tunnistaa, kuka on integraation käyttäjä. Integraatio on loppujen lopuksi sovellus, jonka on tarkoitus helpottaa työntekijän työtehtäviä ja näin ollen tehostaa yrityksen liiketoimintaa (Tähtinen 2005, 37). Integraatio ei kuitenkaan ole tekninen työväline, jolle voitaisiin nimetä yksittäinen käyttäjä. Mikäli integraatio on hyvin otettu osaksi yrityksen tietoteknistä arkkitehtuuria, jokainen integroidun järjestelmän käyttäjä on myös välillisesti integraation käyttäjä. Tyypillisesti käyttäjät huomaavat integraation olemassa olon vain silloin, kun se jostakin syystä ei toimi, mutta käyttäjät voivat havainnoida integraatioratkaisut myös raportoinnin edistymisenä ja kehittymisenä. (Tähtinen 2005, 38.)

4.1.1 Määrittelyn ohjeistus

Tampereen korkeakouluyhteisössä integraatiot usein linkittyvät korkeakouluyhteisön sisäisiin projekteihin, esimerkiksi järjestelmän käyttöönottoprojektissa on hyvin tyypillistä, että ilmenee integraatiotarpeita. Integraatioasiantuntia Miika Haverinen (2019) on laatinut integraatiopalveluille ohjeet määrittelyn tekemiseksi ja toimittamiseksi. Tampereen korkeakouluyhteisössä jokaiselle projektille nimetään tietohallinnosta IT-järjestelmäarkkitehti, jonka kanssa määrittely tavallisesti tehdään. Määrittelyyn voi osallistua muita henkilöitä järjestelmäarkkitehdin ja integraatiopalveluiden asiantuntijan lisäksi, esimerkiksi järjestelmän pääkäyttäjiä tai kehittäjiä, jos tämä nähdään tarpeelliseksi. (Haverinen 2019.)

Yleisesti on hyvä ensiksi sopia aloituspalaveri asiakkaan edustajan kanssa, jossa käsitellään integraation toiminnallisuutta. Määrittelyyn kootaan tiedot tiedonsiirtotavasta ja kenttätason kuvaukset tiedosta, joka integraatiossa liikkuu. Lisäksi kehitystä helpottaa, jos on jo valmiiksi olemassa testidataa tai jonkinlaisia testi- tai kehitysympäristöjä, joita käyttäen integraatiota voidaan testata tai kehittää. (Haverinen, 2019.)

Lähde- ja kohdejärjestelmät määrittävät usein pitkälle tiedonsiirtotavan. Tiedonsiirtotapoja ovat esimerkiksi tiedostonsiirto, tietojen tuominen kohdejärjestelmän rajapintaan, datan kirjoitus suoraan kohdejärjestelmän tietokantaan tai integraatiopalvelun toteuttama rajapinta. (Haverinen 2019.) Sen lisäksi, että määritellään tapa, jolla tiedot siirretään, on Haverinen (2019) kirjannut kysymyksiä, joita on hyvä pohtia asiakkaan edustajan kanssa:

Yleisiä kysymyksiä:

- Mistä kaikista järjestelmistä tietoa pitää hakea?
- Miten usein tieto pitää siirtää?
- Millaiset transaktiorajat operaatioissa on, voidaanko epäonnistunut transaktio perua jotenkin?
- Millaisia tietosuojavaatimuksia dataan liittyy? Sisältääkö data (arkaluontoisia) henkilötietoja? Muita salassa pidettäviä tietoja?
- Millaisia laatuvaatimuksia dataan liittyy? Onko datan laadussa ongelmia - toistoa, puuttuvia tietoja? Pitääkö dataa rikastaa?
- Siirretäänkö yksi vai useampia kokonaisuuksia?
    - Useita HTTP-endpointteja?
    - Useita tietokantatauluja?
    - Useita tiedostoja?
    - Suoritetaanko erilliset aineistot samaan vai eri aikaan? Esim. kirjoitetaanko kaikki tiedostot klo 05:00 aamulla, vai yksi tiedosto aamulla ja toinen tiedosto tunnin välein klo 7-19?

Kenttätason kuvauksessa kerrotaan yksityiskohtaisesti, mitä tietoa integraatiossa siirretään. Kuvaukseen kirjataan myös tieto, miten kyseinen tieto poimitaan tai tuotetaan lähdejärjestelmästä. Kentät esitetään taulukkona, johon kirjataan muun muassa kentän nimi, sen pakollisuus, mistä tieto poimitaan ja muita huomioita. Huomioihin olisi hyvä kirjata, jos kentällä on esimerkiksi maksimipituus, päivämäärillä esitysmuoto tai jos kentän arvo tulee koodistosta, niin millaisia muita arvoja koodistolla on. (Haverinen 2019.)

Jos siirretään csv-tiedostoja, on hyvä selvittää:

- mitä erotinmerkkiä arvojen välillä käytetään
- osaako kohdejärjestelmä käsitellä lainausmerkit oikein
- millä merkistökoodauksella tiedosto on kirjoitettu tai tulee kirjoittaa
- käytetäänkö Windows- vai Linux-tyylistä rivinvaihtoa
- onko tiedostossa tai tuleeko tiedostoon kirjoittaa otsakerivi.

Jos kohdejärjestelmä ei käsittele lainausmerkkejä oikein, on erotinmerkit korvattava siirrettävästä tiedosta jollain muulla merkillä. Siirrettäessä XML-tyyppistä viestiä, tiedostoa tai esimerkiksi SOAP-rajapintakutsua, on hyvä selvittää, onko saatavilla tietomallin määrittävä XML Schema -tiedosto, joka on usein xsd-tiedosto. (Haverinen 2019.)

Tampereen korkeakouluyhteisön integraatiopalveluissa määrittelyiden keräämiseen sovelletaan eräänlaista haastattelun ja aivoriihen sekoitusta. Kun integraatiopalvelu aloittaa määrittelyn, ollaan Haikalan ja Mikkosen (2011) esittelemissä määrittelyn vaiheissa jo vaatimusten analysoinnissa. Voidaankin perustellusti olettaa, että asiakkaalla on valmiudet vaatimusten keräämiseen. Asiakkaan oletetaan osaavan vastata ainakin kysymyksiin, mitä tietoja integraatiossa tuodaan ja mistä lähdejärjestelmästä ne löytyvät. Integraatiopalvelun asiantuntija voi keskittyä teknisempiin aiheisiin, kuten tuodaanko tieto jo integraatiopalveluihin vai pitääkö se vielä hakea lähdejärjestelmästä. Jos integraatio pystytään määrittelemään suoraviivaisesti, voidaan aloituspalaverin yhteydessä päätyä myös validoimaan määrittelyt.

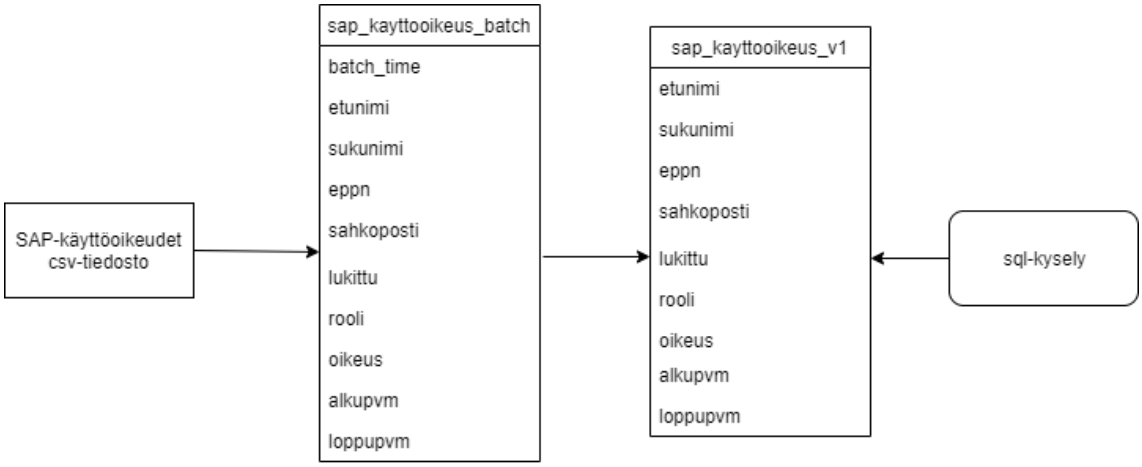
4.1.2 Määrittely asiakkaan kanssa

Ennen kolmen korkeakoulun tietojärjestelmien yhdistymistä Tampereen yliopisto oli käyttänyt palvelua, joka raportoi SAPin käyttöoikeuksista verkkolevylle. Samanlainen palvelu haluttiin säilyttää, mikä käynnisti määrittelyprosessin SAP-integraatiolle. Ennen yhdistymistä yliopistolla ei ollut käytössä nykyisen kaltaista integraatiopalvelua; vanha palvelu oli toteutettu Linux-palvelimella pyörivällä

Perl-koodilla ja raportointi Ingres Report Writer -kielellä. SAP tuottaa viikoittain csv-tiedoston käyttöoikeuksista palvelimelleen.

Määrittelyä tehtiin yhdessä Tampereen korkeakouluyhteisön taloushallinnon integraatioasiantuntijan Maija Kosken kanssa sekä itsenäisesti vanhojen toteutusten koodien perusteella. Koski myös toimitti anonyymin testiaineiston kehitystä varten, josta selvisi esimerkiksi tiedoston merkistökoodaus, erotinmerkki ja tiedot, jotka tiedostossa tuodaan. Järjestelmien muutosten takia myös Miika Haveriselta pyydettiin apua vanhojen toteutusten tulkinnassa. TAMKissa ei tiettävästi ole ollut aikaisemmin vastaavaa palvelua, joten määrittely toteutettiin yliopiston palvelun pohjalta ja laajennettiin koskemaan myös TAMKin SAPia.

Tampereen korkeakouluyhteisön integraatioväylän yleinen arkkitehtuuri määrää, että tiedostopohjaisesti vastaanotettavat tiedot viedään integraatiopalvelun sisäiseen välimuistitietokantaan. Tätä varten SAP-käyttöoikeuksia koskeva kokonaisuus jaettiin kahteen integraatioon, joista toinen tuo tiedot välimuistitietokantaan ja toinen muodostaa ja vie raportit verkkolevylle. Tietojen tuontia varten tietokantaan luodaan arkkitehtuurin mukaiset taulut ja näkymät tiedoille (kuvio 14), joita ulospäin vievä integraatio käyttää.



KUVIO 14. Tietojen tuonti välimuistitietokantaan sovellettuna SAPin käyttöoikeuksiin

Lisäksi määrittelyvaiheessa tuli pohdittavaksi, viedäänkö yliopiston ja TAMKin tiedot samaan vai eri tietokantatauluun. Tiedot päätettiin viedä erillisiin tauluihin, sillä siihen ratkaisuun on päädytty aikaisemmissakin integraatioissa, jotka tuovat tietoja SAP-järjestelmistä. Tätä päätöstä tuki myös se, että yliopistolla ja TAMKilla on omat instanssit SAPista ja raportit viedään erikseen yliopistolle ja TAMKille. Päätös myös hyväksytettiin Miika Haverisella.

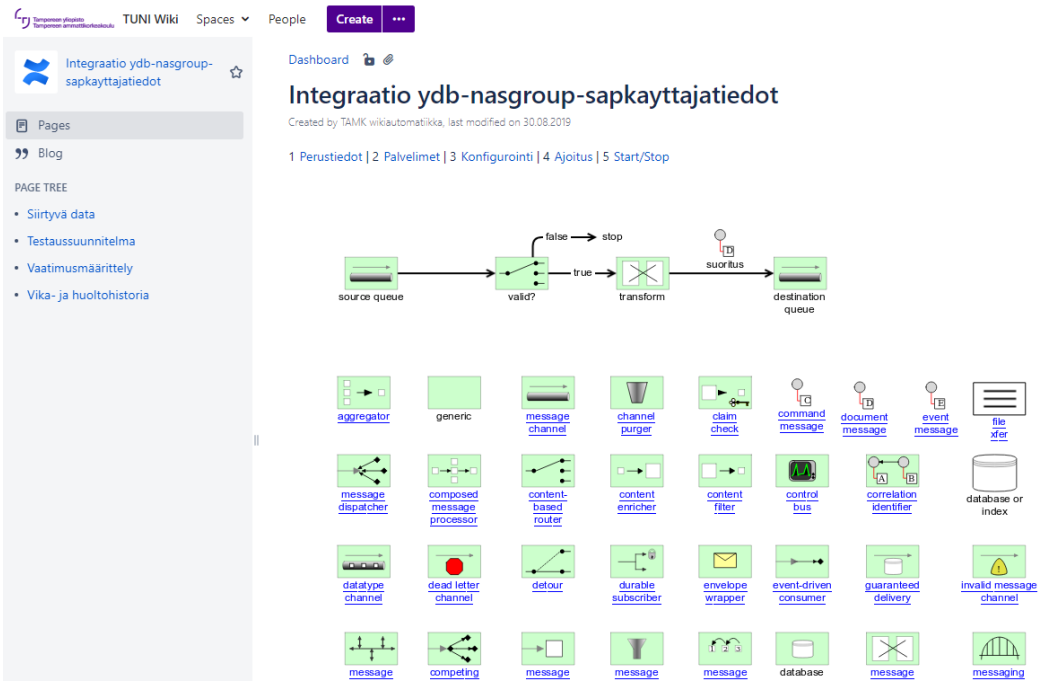
Määrittelyn aikana päätettiin myös, että reitit parametrisoidaan niin, että sekä yliopisto että TAMK käyttävät samaa reititystä ja reitille annettu parametri kertoo, viedäänkö yliopiston vai TAMKin tietoja. Parametrisointia käytetään, jottei samaa koodia tarvitse kirjoittaa kahteen kertaan. Tähän päädyttiin myös siksi, että aiemmin toteutetuissa SAP-integraatioissa on tehty samanlainen ratkaisu.

Vuodenvaihteessa 2019-2020 TAMKissa otetaan käyttöön yhteiset järjestelmät ja samalla TAMKin SAP-instanssi vaihtuu, joten TAMKin SAP alkaa vasta vuoden vaihteen jälkeen tuottamaan integraation vaatimaa käyttöoikeusraporttia. Tämän takia reitit, jotka kuljettavat TAMKin tietoja, täytyy olla konfiguroitavissa päälle ja pois riippumatta yliopiston reiteistä. Konfiguraatiotiedostoon määritellään sellaisia arvoja, jotka voivat vaihtua ympäristöstä toiseen, esimerkiksi ajastus ja tiedostojen polut ovat usein tietoja, joiden arvot muuttuvat, kun integraatio viedään kehityksestä testiin ja testistä tuotantoon.

SAPin käyttöoikeudet hallinnoidaan SAPin sisältä, eikä Tampereen korkeakouluyhteisön yhteisestä käyttövaltuushallinnasta, joten on myös tärkeää pitää huolta oikeuksien ajantasaisuudesta sekä SAPissa että käyttövaltuushallinnossa. Yliopiston vanhassa palvelussa oli tätä tarkoitusta varten myös käyttäjätietojen ristiin vertailu käyttövaltuushallinnon kanssa ja vertailun tulosten raportointi. Tarkistuslista on ollut hieman kömpelö, tämän ongelman ratkaisu on osaltaan kehityksessä, joten se päädyttiin tässä kohtaa rajaamaan ensimmäisen version ulkopuolelle. Vertailu ja raportointi toteutetaan mahdollisesti myöhemmin, jos sille vielä nähdään tarvetta.

4.1.3 Määrittelyt toteutettaville integraatioille

Jokaisella integraatiolla on oma wikisivustonsa Tampereen korkeakouluyhteisön Confluencessa (kuva 6), jonka etusivulle on kuvattuna integraation kulku ja konfiguraatio. Tälle wikisivustolle kootaan kaikki tärkeät tiedot integraatiosta. Sivustolla on omat välilehtensä siirtyvälle datalle, testaussuunnitelmalle, vaatimusmäärittelylle ja vika- ja huoltohistorialle. Määrittelyvaiheessa täytetään vaatimusmäärittely sivulle tärkeät tiedot niin tarkasti, että kuka tahansa kehitystiimistä voi sen pohjalta toteuttaa integraation. Samalla kuvataan siirtyvä data sen omalle välilehdelle. Tyypillisesti tiedosta muodostetaan taulukko, josta selviää, mitä tietoa integraatiossa kulkee.



KUVA 6. Kuvakaappaus integraation wikisivustosta.

Integraatio, joka tuo SAP käyttöoikeudet välimuistitietokantaan, nimettiin sap-ydb-kayttooikeus-integraatioksi ja raportit muodostava ja vievä integraatio nimettiin ydb-nasgroup-sapkayttajatiedot-integraatioksi. Tuovan integraation määrittelyihin kerättiin vaatimusmäärittely-sivulle (liite 1) ja siirtyvä data -välilehdelle kuvattiin saapuvan tiedon rakenne ja yhteys tietokantatauluun taulukkoon (taulukko 1).

TAULUKKO 1. csv-tiedoston sisältö ja kenttätasonkuvaus

CSV sarake nro	Sarake CSV:ssä	Tieto	Esimerkki	Tallennetaan YDB:hen	Sarake YDB:ssa
1	USERNAME	käyttäjätunnus, ei vastaa identiteetin käyttäjätunnusta			
2	FIRSTNAME	Etunimi		x	etunimi
3	LASTNAME	sukunimi		x	sukunimi
4	EPPN	käyttäjätunnus@tuni.fi (otetaan vain käyttäjätunnus mukaan)	tunnus@tuni.fi	x	eppn
5	EMAIL	sähköposti	etunimi.sukunimi@tuni.fi	x	sahkoposti
6	HOMEORGANIZATION	kotiorganisaatio	tuni.fi		
7	LOCKED		false	x	lukittu
8	BUSINESSROLE	rooli		x	rooli
9	PRIVILEGE	oikeus		x	oikeus
10	VALIDFROM	alkupäivämäärä		x	alkupvm
11	VALIDTO	loppupäivämäärä		x	loppupvm
12	FIRSTAPPROVER				
13	SECONDAPPROVER				

Raportit muodostavan integraation määrittelyt koottiin suurimmaksi osaksi vanhan toteutuksen koodeista ja ne on kuvattu integraation wikisivulle (liite 2). Siirtyvä data -sivulle on kuvattu taulukkoon kenttätason tiedot, siitä mistä tarvittavat tiedot poimitaan (taulukko 2) ja raporttien mallipohjat (liite 3).

TAULUKKO 2. Tietojen poiminta välimuistotietokannasta kenttätason kuvaus

tieto	kenttä ydb:ssa
henknro	TAU: identiteetti_v1.henkilonumero TAMK: identiteetti_v1.tamk_henkilonumero
ktunnus	sap_kayttooikeus.eppn tamk_sap_kayttooikeus.eppn
sukunimi	sap_kayttooikeus.sukunimi tamk_sap_kayttooikeus.sukunimi
etunimi	sap_kayttooikeus.etunimi tamk_sap_kayttooikeus.etunimi
rooli	sap_kayttooikeus.rooli tamk_sap_kayttooikeus.rooli
oikeus	sap_kayttooikeus.oikeus tamk_sap_kayttooikeus.oikeus

4.2 Integraatioiden toteutus

Integraatiototeutus alkaa sillä, että Maven-projekti luodaan käyttäen Maven-arkkityyppiä. Komento luo projektirungon kansiorakenteineen, jossa on valmiina pom.xml-tiedosto, eli Maven-projektitiedosto, README.md-tiedosto, johon kuvataan tärkeät asiat, joista muutosten tekijän tulee olla tietoinen, esimerkkikonfiguraatiotiedosto. Nämä tiedostot on löydyttävä, jokaisesta integraatiosta, lisäksi projektin juureen voidaan lisätä kansio, johon tallennetaan testidata kehitystä ja testausta varten (Rönkä 2019c).

Kun projektirunko luodaan, pitää integraatio samalla nimetä. Perussääntönä integraatiot nimetään lähde-kohde-tieto, jossa lähde on lähde järjestelmän nimi tai lyhenne, kohde on kohde järjestelmän nimi tai lyhenne ja tieto kohtaan, kirjataan mitä tietoa integraatiossa kulkee. Usein nimeämien tapahtuu, jo ennen toteutuksen alkamista. Tampereen korkeakouluyhteisössä integraatioiden toteutusta ohjaa integraatiokäsikirja, johon on kerätty sovitut käytännöt integraatioiden toteuttamiseksi, esimerkiksi riippuvuuksien hallinnasta, lokituspisteistä ja nimeämiskäytännöistä (Rönkä 2019c).

4.2.1 Toteutetut integraatiot

Integraation toteutus on hyvä aloittaa kartoittamalla tiedonsiirtotapa ja edetä kenttätason kuvauksiin vasta sen jälkeen. Tietoja tuovan integraation määrittelyssä (liite 1) on todettu, että tiedot tulee hakea palvelimelta. Tiedoston siirron palvelimelta työhakemistoon voidaan toteuttaa monella eri tavalla, mutta koska projektissa on huomioitu, että tietynlaisia reitityksiä tarvitaan usein, voidaan suoraan hyödyntää yleiskäyttöistä valmisreittiä.

Kun tiedosto on siirretty työhakemistoon, se pitää lukea ja viedä välimuistitietokantaan. Tällaista siirtoa varten on myös valmiina valmisreitti, mutta koska välimuistitietokannan merkistökoodaus on utf-8 ja tiedosto on windows-1252 merkistössä, eikä valmisreittiin ollut lisätty valmiutta sille, että merkistökoodaukset eroavat toisistaan, ei sitä voitu suoraan käyttää. Tilanteeseen jäi kaksi vaihtoehtoa: joko kirjoittaa integraatiolle reitti, joka on käytännössä kopio valmisreitistä, lisättynä merkistökoodauksen muunto, tai lisätä merkistökoodauksen muunto valmisreittiin.

Koska aikaisemmin merkistön muunnolle ei ole ollut tarvetta, eikä ainakaan seuraavissa kehitykseen tulevissa integraatiossa nähty tarvetta, niin päädyttiin siihen, että tässä integraatiossa reitti toteutetaan integraatioon itseensä, muokkaamatta valmisreittiä. Tulevaisuudessa muutos valmisreittiin toteutetaan, jos lisätarvetta esiintyy. Samalla täytyy muuttaa kyseinen integraatio käyttämään myös valmisreittiä.

Camelissa on sisäänrakennettuja dataformaatteja (Onofré 2015, 18), joista ainakin kahta on aikaisemmin käytetty Tampereen korkeakouluyhteisön integraatiopalveluissa csv-tiedostojen käsittelyyn. Dataformaatit muuntavat tietoa binääristä tai tekstistä Java-luokiksi ja päinvastoin, helpottaen kehitystä (Data Format). Koska muiden muassa Karaf-ajoympäristöön asennettuja dataformaatteja ylläpidetään Parent-projektissa (Rönkä 2019), on tärkeää käyttää jo valmiiksi asennettuja toimintoja, jottei ympäristöön tule turhia riippuvuuksia. Valinta tehtiin CSV- ja Bindy-dataformaatin välillä.

Vaikka CSV-dataformaatilla olisi saanut tehtyä kaiken tarvittavan, päädyttiin käyttämään Bindyä, sen monipuolisuuden takia. Bindy on myös hieman uudempi komponentti kuin CSV. CSV oli ensimmäisen kerran mukana Camelin versiossa 1.3 ja Bindy versiossa 2.0. (Bindy Dataformat; CSV Dataformat.)

Toteutuksen yhteydessä huomattiin, ettei Bindy käytä Java-luokan aksessoreita, eli set-metodia, silloin kun se muuntaa tiedon Java-luokaksi. Koska SAPin csv-tiedostossa oli erikoisesti käytetty lainausmerkkejä (liite 1), tuli ne poistaa datasta ennen tietokantaan vientiä, lisäksi esimerkiksi eppn-sarakkeesta tuli poimia vain tunnuksen osuus. Koska näitä prosessointeja ei voitu tehdä tiedoston lukemisen yhteydessä, siirrettiin prosessointi siihen kohtaan, kun tiedot viedään tietokantaan, eli Java-luokan get-metodeihin.

Tietojen toteutuksen yhteydessä myös havahduttiin siihen, että tietokantaan viedään tietoa kerran viikossa. Tiedot viedään yhtenä joukkona niin, että kaikille joukon tietueille annetaan sama aikaleima. Aikaleiman perusteella näkymä näyttää vain uusimman joukon. Näillä määrittelyillä taulusta ei kuitenkaan koskaan mitään poisteta, mikä tarkoittaisi sitä, että taulu vain täyttyy ajan myötä. Tämän ratkaisemiseksi päädyttiin integraatioon toteuttamaan tietokannan puhdistus -reitti, joka jättää vähintään yhden joukon tietokantaan, mutta poistaa muut. Tähän otettiin mallia jo aikaisemmin toteutetuista integraatioista, jotta toteutus olisi yhteneväinen vallitsevan linjan kanssa.

Tietoja raportoiva integraation toteutus aloitettiin kartoittamalla tiedonsiirtotapa. Tietokannasta tuotettavien tiedostojen luontiin on olemassa valmisreitti, mutta se rajoittuu perus tietomalleihin, kuten csv-tiedostojen luontiin, ei se sovellu raporttien kirjoittamiseen. Tiedostojen luontia varten päädyttiin siis kirjoittamaan oma reitti, joka hakee tiedot tietokannasta ja kirjoittaa ne tiedostoon mallin mukaan.

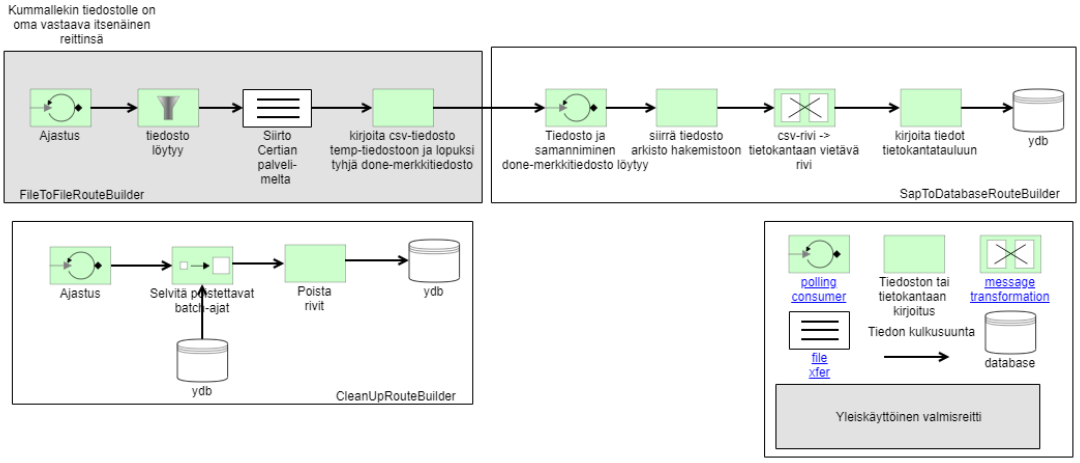
Tiedostojen kirjoituksessa valittiin käytettäväksi FreeMarkeria, joka on Apachen Java-kirjasto tekstin tuottamiseen. FreeMarkerissa määritellään mallit, joissa käyttäen FreeMarkerin notaatiota kerrotaan, miten teksti halutaan kirjoittaa. Mallit ovat ftl-muotoisia tiedostoja. FreeMarker toimii todella hyvin Camelin kanssa yhteen. Camelin versiosta 2.10 lähtien Camelissa on ollut oma komponentti FreeMarkerin käyttöön (FreeMarker Component).

Verkkolevylle vienti tapahtuu käyttäen File-komponenttia, sillä ympäristö näkee verkkolevyt, kuin ne olisivat tavallisia kansioita, mikäli ympäristöstä on pääsy verkkolevylle. Koska raportoidaan ulos ympäristöstä, niin toteutuksen aikana päätettiin lisätä tiedoston siirtävä valmisreitti integraatioon, sitä varten, että kirjoitettavat tiedostot halutaan mahdollisesti arkistoida niin, että vanhoihin versioihin päästään käsiksi myöhemmin esimerkiksi virhetilanteita selvitettäessä. Jos olisi tyydytty siihen, että kirjoitetaan suoraan verkkolevylle ja kukaan ei kopioi vanhoja versioita talteen menetettäisiin historia, kun uusi tiedosto kirjotetaan. Arkistoinnin olisi voinut lisätä myös tiedoston luovan reitin loppuun, mutta valmisreitin käyttö oli huomattavasti nopeampaa, sillä siihen oli jo valmiiksi rakennettu konfiguroitavissa oleva arkistointi. Tätä päätöstä tuki myös se, että jos tiedostoja ei jostain syystä haluttaisikaan arkistoida, voi yksittäisen reitin sammuttaa asettamalla sen automaattisen käynnistyksen epätodeksi, sen vaikuttamatta muuten reittien toimintaan.

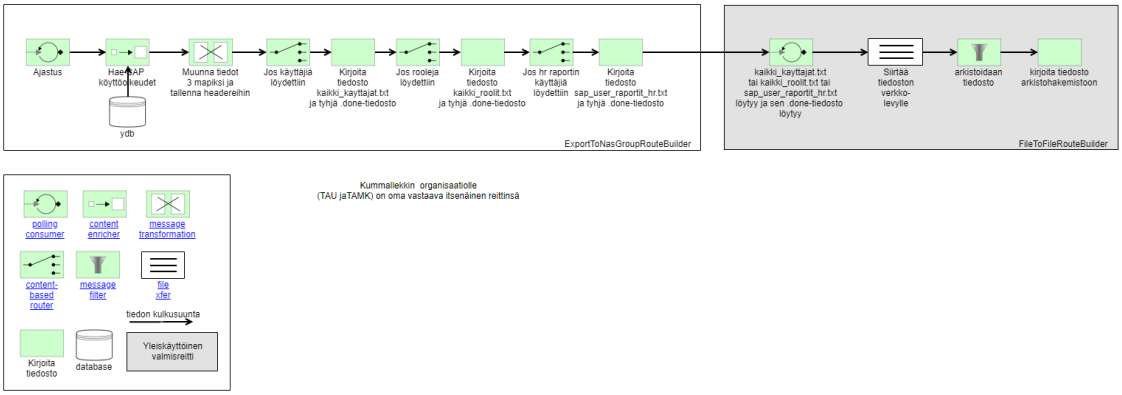
4.2.2 Integraation dokumentaatio

Integraation dokumentaatio tehdään integraation wikisivulle, tässä kohtaa päivitetään mahdollisesti siirtyvä data -välilehden tiedot ja piirretään etusivulle kuvio integraation kulusta. Lisäksi etusivulle lisätään, mikäli on tietoja integraation ajastukseen vaikuttavista tekijöistä, esimerkiksi tietoja tuovan integraation tietoihin kirjattiin, että SAP muodostaa tiedoston, joka sunnuntai kello 6.35, joten ajastus olisi hyvä asettaa tämän ajan jälkeiseksi.

Tietoja tuovan integraation kulku on kuvattu kuvioon 15 ja raportit muodostava ja kirjottava integraation kulku kuvioon 16. Kuvioista käy ilmi myös erilliset reitit ja se, että TAMKin ja yliopiston tiedot kulkevat itsenäisesti samaa reittiä pitkin.



KUVIO 15. Tiedon kulku tietoja tuovassa integraatiossa.



KUVIO 16. Tiedon kulku raportit muodostavassa ja kirjoittavassa integraatiossa.

Tärkeä osa integraation dokumentaatiota on myös konfiguraation esimerkkitiedosto. Tampereen korkeakouluyhteisön ympäristöissä integraation konfiguraatio on cfg-tiedosto, ja jokaisesta integraatiosta projektin juuresta on löydyttävä esimerkki konfiguraatiosta. Esimerkkitiedosto täytetään niin kuin se olisi käytössä oleva konfiguraatio tiedosto, lisäksi sitä voi kommentoida ja täydentää. Jos määrittelyssä ei erikseen sanota konfiguraatiossa olevia asioita, olisi sinne hyvä pistää ainakin jokaisen reitin automaattikäynnistyksen valinta, ajastukset, osoitteet ja polut ja mahdollisesti vaihtuvat suureet, kuten roolien nimet, lisäksi tulee huomioida, että jos konfiguraatiotiedostoon kirjoitetaan suoraan salassa pidettäviä tietoja, tulee ne suojata (Rönkä 2019c). Toteutettujen integraatioiden esimerkkikonfiguraatiot koostuvat juuri tällaisista parametreista (liite 4; liite 5), lisäksi konfiguraatioon on huomioitu TAMKin ja yliopiston konfiguraatioparametrit erikseen.

4.2.3 Katselmoinnin tulokset

Haikala ja Mikkonen (2011, 200) määrittelevät katselmoinnin, tai teknisen katselmoinnin, tietyn vaiheen päättymispisteeksi. Katselmoinnit ovat keino luoda läpinäkyvyyttä etenemiseen (Haikala & Mikkonen 2011, 200). Katselmointi on myös osa laadunvarmistusta. Jotta katselmointi toimii laadunvarmistusvälineenä, sitä täytyy tehdä järjestelmällisesti, suunnitelmallisesti ja kattavasti. Katselmointiin tulisi aina osallistua muitakin tuloksen tuottaneita henkilöitä. Lisäksi katselmoinnissa olisi hyvä toteuttaa jatkuvan katselmoinnin periaatetta, eli että jokaisen vaiheen laatu tarkistetaan katselmoinnilla, ennen kuin työtä käytetään jatkotyön perustana. (JUHTA-julkisen hallinnon tietohallinnon neuvottelukunta 2011.)

Tampereen korkeakouluyhteisön integraatiopalveluissa uuden integraation kehitysprosessi päättyy siihen, että integraatio menee tekniselle arkkitehdille katselmoitavaksi. Arkkitehti tarkastaa, että vaaditut dokumentit ovat olemassa, dokumentaatio on ymmärrettävää ja yksiselitteistä ja, että asennusohjeesta käy ilmi muu muassa paketin asennusohjeet, vaaditut konfiguraatiot ja lokitus- ja monitorointiohjeet (Rönkä, 2019). Katselmoinnista integraatio voi tulla takaisin kehitykseen, jos arkkitehti löytää puutteita tai muuta korjattavaa integraatiosta. Jos huomautettavaa ei löydy, integraatio siirtyy katselmoinnista testattavaksi. Arkkitehdit ovat usein huomattavasti kokeneempia myös teknisistä lähtökohdista, joten, varsinkin juniortasoiselle kehittäjälle, katselmoinnit ovat myös tärkeitä oppimisen paikkoja.

Tietoja tuova integraatio siirrettiin kehityksestä katselmointiin 24.9. Katselmoinnin kommentit tulivat 1.10. ilman korjattavaa, jolloin integraatio siirrettiin testattavaksi (Rönkä 2019a). Raportit kirjoittava integraatio siirrettiin katselmointiin 1.10. Katselmoinnin tulokset tulivat 9.10. Katselmoinnissa ei löytynyt korjattavaa, mutta dokumentointia ja toteutustapaa oli kommentoitu. Rönkä (2019b) oli kiinnittänyt huomiota siihen, että dokumentointi kuvaa hieman niukasti integraation toiminnallisuudet. Toteutustavasta Rönkä (2019b) oli kommentoinut, että ”Nykytavan mukainen. Koodi helppoa tulkita, ei turhaa, ei kommentteja”, mikä viittaisi siihen, että koodia tulisi kommentoida enemmän.

5 POHDINTA

Opinnäytteen aikana:

- tutustuttiin kattavasti erilaisiin integraatiomalleihin
- selvitettiin ServiceMix-integraatioteknologiaan
- perehdyttiin Tampereen korkeakouluyhteisön määrittelyohjeisiin
- tuotettiin vaatimusmäärittelyt kahdelle toteutettavalle integraatiolle
- perehdyttiin Tampereen korkeakouluyhteisön integraatiototeutuksen ohjeistukseen
- toteutettiin määritellyt integraatiot
- dokumentoitiin integraatiot

Tässä raportissa on esitelty integraatiomalleja ja ServiceMix-teknologiaa, verrattu Tampereen korkeakouluyhteisön vaatimusmäärittely ohjeita kirjallisuuteen vaatimusmäärittelyistä. Lisäksi on avattu integraatioiden määrittely- ja toteutusprosessia.

Tämän raportin perusteella opinnäytetyön tavoitteet ovat täyttyneet. Opinnäytetyön aikana toteutettiin integraatiot ja toinen niistä on jo tuotantokäytössä, joten siltä pohjalta voidaan osoittaa, että myös tarkoitus on näytetty täyttyvän.

Kehityksen vastuu loppuu siihen, että integraatio siirtyy testattavaksi, mutta oikea palaute integraation toteutuksesta tulee testauksesta tai tuotannosta. Tämän opinnäytetyön aikana toteutetut integraatiot siirtyivät testattavaksi. Monesti integraatioihin tulee muutospyyntöjä vielä tuotantoon viennin jälkeen, sillä kaikkia tarvittavia ominaisuuksia ei välttämättä ole tiedossa tai tarpeet muuttuvat.

Tietojen tuovan integraation osalta testauksessa ei löydetty korjattavaa ja se vietiin tuotantoon elokuussa 2019. Tuotannossa kuitenkin ilmeni ongelma, kun tiedostossa tulevassa tiedossa oli käytetty pilkkua itse tietosisällössä. Sillä pilkku on myös csv-tiedoston erotin merkki, ei integraatio osannut lukea sitä oikein. Ongelmaa selvitettiin ja selvisi, ettei määrittelyn aikana toimitettu esimerkkiaineisto pitänytkään paikkaansa. Esimerkkiaineistossa riveillä oli erikoisesti käytetty lainausmerkkejä (liite 1), kun tuotannossa rivit olivat normaalisti lainatut, eli jokainen

tietue omien yksittäisien lainausmerkkien sisällä. Tästä tehtiin uusi tehtävä kehitykselle ja korjausten jälkeen integraatiosta tehtiin uusi julkaisu tuotantoon. Tietoja raportoiva integraatio on vielä testauksessa, eikä sen osalta ole opinnäytteen aikana ilmennyt korjattavaa.

Määrittelyiden osalta prosessi oli aika itsenäinen, eikä siis hyvin kuvaa tyypillistä määrittelyprosessia. Toisaalta vanhan toteutuksen olemassaolo helpotti huomattavasti prosessia, sillä asiakkaan kanssa ei oltu yhteyksissä prosessin aikana, jolloin ei tietenkään tullut tilanteita, jossa omaa aikataulua joutuisi sovittamaan yhteen asiakkaan kanssa tai joutuisi odottamaan vastauksia. Toisaalta tämä myös johtaa siihen, ettei määrittelyiden validointia kunnolla suoritettu, joten testauksesta tai tuotannosta oli odotettavissa muutospyyntöjä.

Määrittelijän kokemattomuuden huomaa tässä prosessissa siitä, että määrittelyn aikana ei tullut ilmi sitä, että tietoja pitää myös siivota pois tauluista. Se olisi kannattanut huomata määrittelyn aikana ja kirjata siitä muutama rivi tietoja tuovan integraation määrittelyyn, jottei tietojen siivous olisi integraation toteuttajan muistin varassa.

Toteutus onnistui hyvin ja soveltuvin osin toteutuksessa käytettiin valmiita reittejä. Eikä katselmoinnissa löytynyt sellaisia puutteita, joita olisi tarvinnut korjata. Katselmoinnissa kommentoitiin, että kommentteja tulisi olla enemmän koodissa, mutta kommentoinnista ei ole vielä tehty ohjeistusta, joten on vaikea arvioida, millaisia kommentteja koodiin kaivattaisiin. Ohjeistus on kyllä projektin työlistalla, joten tulevaisuudessa katselmointikin varmasti helpottaa kommentoinnin ohjeistuksen myötä.

Raportteja kirjoittavan integraation toteutuksessa tuli eteen tilanne, jossa piti valita toteuttaako integraatioon oma reitti vai muokatako Parent-projektin valmisreittiä. Jälkikäteen ajateltuna olisi ollut parempi toteuttaa muutokset Parent-projektiin, sillä siinä olisi säästänyt toteutukseen tarvittavan koodin määrää ja tulevaa työtä. Parent-projekti on kuitenkin jatkuvasti muutoksien kohteena, joten muutosten tekemättömyys oli vain pelkkää turhaa ujostelua. Toki on aina ikävämpi, jos tekee virheitä komponentteihin, joita kaikki kehittäjät käyttävät. Kuten yleensäkin

ohjelmistokehityksessä virheitä ei tulisi pelätä, sillä ne ovat usein helposti korjattavissa, eikä kehitysympäristöön tehdyt virheet ole liiketoimintakriittisiä.

Toiminnallisuuksien kannalta integraatioiden toteutuksesta jäi pois vertailu Tampereen korkeakouluyhteisön käyttäjähallinnan ja SAP-järjestelmän käyttäjätietojen kesken. Tämä toiminnallisuus tultaneen määrittelemään ja toteuttamaan vielä vuoden 2019 aikana. Tekstitiedostojen kirjoitus raportteina ei ehkä ole enää tätä päivää, joten jatkokehitysehdotuksena raportit voisi kirjoittaa helpommin käsiteltäviksi pdf-dokumenteiksi. Camelilla on valmiudet kirjoittaa ja lukea pdf-dokumentteja (PDF component 2019), joten tämän toiminnallisuuden toteutus olisi mahdollinen.

Henkilökohtaisen kehityksen kannalta opinnäyte on parantanut valmiuksia määrittelyn tekemiselle. Vaikka tässä prosessissa asiakaskontakti oli hyvin niukkaa, tarjosi se kuitenkin tilanteita, joissa täytyi kysyä neuvoa, ja näin poisti eräänlaisia jännitystä, joka liittyy epätietoisuuden kommunikoimiseen. Lisäksi määrittely kokonaisuuden tavoitteena oli tuoda kokemusta määrittelyn tekemisestä ja tässä prosessissa sen sai toteuttaa hieman turvallisemmassa kontekstissa. Lisäksi opinnäyte on harjaannuttanut ja tuonut rutiinia integraatioiden toteuttamiseen ja siten parantanut kirjoittajan osaamista.

LÄHTEET

Amundsen, M. & McLarty, M. & Mitra, R. & Nadareishvili, I. 2016. Microservice Architecture. 1. painos. O'Reilly Media, Inc.

Anstey, J.& Ibsen, C. 2018. Camel in action. 2. painos. Manning Publications.

Apache Camel. N.d. Apache Software Foundation. Luettu 28.8.2019. https://camel.apache.org/

Apache Karaf Container 4.x – Documentation. 2019. Apache Software Foundation. Päivitetty 21.5.2019. Luettu 11.10.2019. http://karaf.apache.org/manual/latest/

Bindy Dataformat. N.d. Apache Software Foundation. Luettu 2.10.2019. https://camel.apache.org/components/latest/bindy-dataformat.html

Borodulin, P. 2019. Välimuistitietokanta. Julkaisematon. Integraatiopalvelut. Tampereen korkeakouluyhteisö.

Bussler, C. 2003. B2B Integration. 1. painos. Springer, Berlin, Heidelberg

CSV Dataformat. N.d. Apache Software Foundation. Luettu 2.10.2019. https://camel.apache.org/components/latest/csv-dataformat.html

Cummins, H. & Ward, T. 2013. Enterprise OSGi in Action. 1.painos. Manning Publications.

Data Format. N.d. Apache Software Foundation. Luettu 2.10.2019. https://camel.apache.org/manual/latest/data-format.html

Dirksen, J. & Rademakers, T. 2008. Open-Source ESBs in Action: Example Implementations in Mule and ServiceMix. 1. painos. Manning

Edstrom, J., Kesler, H. & Goodyear, J. 2013. Learning Apache Karaf. Packt Publishing

FreeMarker Component. N.d. Apache Software Foundation. Luettu 2.10.2019. https://camel.apache.org/components/latest/freemarker-component.html

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. 12., uudistettu painos. Helsinki: Talentum.

Haverinen, M. 2019. Ohje määrittelyn tekemiseksi ja toimittamiseksi integraatiopalvelulle. Julkaisematon. Integraatiopalvelut. Tampereen korkeakouluyhteisö

Hohpe, G. & Woolf, B. 2003. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. 1. Painos. Addison-Wesley Professional 2003.

JUHTA-julkisen hallinnon tietohallinnon neuvottelukunta. 2011. JHS 182 ICT-palvelujen kehittäminen: Laadunvarmistus. JHS-suositus. Julkaistu 15.12.2011. Luettu 30.09.2019. http://docs.jhs-suositukset.fi/jhs-suositukset/JHS182_liite6/JHS182_liite6.html

Konsek, H. 2013. Instant Apache ServiceMix How-to. 1.painos. Packt Publishing

Mikkonen, J. 2017. Rest on nettipalveluiden yhteinen kieli. Tivi. Julkaistu 27.5.2017. Luettu 2.11.2019. https://www.tivi.fi/uutiset/rest-on-nettipalveluiden-yhteinen-kieli/23703ab5-dd19-383e-a422-ebfc3d910583

Miller, R. 2009.The Quest for Software Requirements. 1. painos. MavenMark Books cop.

Onofré, J. 2015. Mastering Apache Camel. 1. painos. Packt Publishing Ltd.

OSGi Alliance. 2019. About Us. Luettu 11.10.2019. https://www.osgi.org/about-us/

PDF component. 2019. Apache Software Foundation. Luettu 20.10.2019. https://camel.apache.org/components/latest/pdf-component.html

Rönkä, T. 2019a. Integraatio sap-ydb-kayttooikeudet katselmointi. Julkaisematon. Integraatiopalvelut. Tampereen korkeakouluyhteisö.

Rönkä, T. 2019b. Integraatio ydb-nasgroup-sapkayttajatiedot katselmointi. Julkaisematon. Integraatiopalvelut. Tampereen korkeakouluyhteisö.

Rönkä, T. 2019c. Integraatiokäsikirja. Julkaisematon. Integraatiopalvelut. Tampereen korkeakouluyhteisö.

Rönkä, T. 2019d. Toimintamalli. Julkaisematon. Integraatiopalvelut. Tampereen korkeakouluyhteisö.

The Scrum Guide™. 2018. ScrumGuides.org. Luettu 19.10.2019 https://scrumguides.org/scrum-guide.html

ServiceMix. N.d. Apache Software Foundation. Luettu 28.8.2019. https://servicemix.apache.org/

Tähtinen, S. 2005. Järjestelmäintegraatio - Tarve, vaihtoehdot, toteutus. 1.painos. Helsinki: Talentum.

Vähimaa, A. 2017. Integraatioiden uusi aika. Tivi 6/2017, 26-29.

LIITTEET

Liite 1. sap-ydb-kayttooikeudet-integraation määrittely

Haetaan Certian palvelimelta ajastuksella csv-tiedosto ja luetaan se tietokantaan.

Tiedoston merkistö on windows-1252 , erotinmerkki on pilkku (,).
Tiedoston nimi on muotoa: tuni.fi_USERS_REPORT_dd-mm-yyyy.csv
Tiedosto sisältää otsikko rivin:

"USERNAME,""FIRSTNAME"",""LASTNAME"",""EPPN"",""EMAIL"",""HOMEORGANIZATION"",""LOCKED"",""BUSINESSROLE"",""PRIVILEGE"",""VALIDFROM"",""VALIDTO"",""FIRSTAPPROVER"",""SECONDAPPROVER"""

Tiedostot haetaan sekä TAU:n SAP:sta että TAMK:n SAP:sta (info) TAMK:ssa tuotantoon alkaa muodostua tiedostot vasta alkuvuodesta 2020:

TAU: /tuotanto/3100/ID1002
TAMK: /tuotanto/4100/ID1002
Parametrisoidaan samaan tapaan, kuin sap-ydb-perustiedot

Liite 2. ydb-nasgroup-sapkayttajatiedot-integraation määrittely

Haetaan ydb:sta SAP-käyttöroolit ja muodostetaan niistä raportit verkkolevylle (ent. S-asema).
TAU:n ja TAMK:n aineistoista kirjoitetaan omat raportit ja ne viedään eri verkkolevyille.

Raportit
Tiedoista muodostetaan neljä tekstitiedostoa:

sap_users_tarklst.lst: tarkistuslista
sap_user_raportit_hr.txt : lista palkkaraporttikäyttäjistä
kaikki_kayttajat.txt : lista käyttäjistä ja heidän rooleistaan
kaikki_roolit.txt :Lista kaikista rooleista ja niiden käyttäjistä

Liitteet numeroidaan juoksevasti omalla numerollaan ja siinä järjestyksessä, missä niihin viitataan tekstissä. Liitteissä on oltava otsikko ja lähdemerkintä, ellei liiteaineisto ole kirjoittajan laatima. Jos liitteessä on useita sivuja, niin esimerkiksi kolmisivuisen liitteen ensimmäisen sivun oikeaan yläreunaan kirjoitetaan 1 (3) ja seuraavalle sivulle 2 (3) ja viimeiselle sivulle 3 (3).

Liite 3. Raporttipohjat

sap_user_raportit_hr: 1(2)

Tampereen yliopisto, tietohallinto 30.08.2019

Henkilöt, joilla on SAP-rooli: SAP_RAPORTIT_LUKU_JA_KIRJOITUS

Nimi (Henknro)

Testaaja, Teppo (13314)

oikeus1, oikeus2, oikeus3,

Sukunimi, etunimi (henkilonumero)

oikeudet listattuna

kaikki_kayttajat:

Tampereen yliopisto, tietohallinto 30.08.2019

Henkilöt, joilla on jokin SAPin ERP-rooli

Testaaja. Teppo (13314, xktete)

KAIKKI KÄYTTÄJÄT
(directly assigned privilege)

Sukunimi, Etunimi (henkilonumero, kayttajatunnus)

Rooli1
Rooli2

(jatkuu)

Tampereen yliopisto, tietohallinto 30.08.2019
Kaikki roolit ja niihin liitetyt henkilöt

TUNI.FI KAIKKI KÄYTTÄJÄT

Henkilokunta, Heini (13315, gthehe)
Testaaja, Teppo (13314, xktete)

Rooli

Sukunimi, Etuninimi (henkilonumero, kayttajatunnus)

Liite 4. sap-ydb-kayttooikeudet-integraation esimerkkikonfiguraatio

```
# Example of the cfg file

#-----------------------------
# TAU
#-----------------------------
tau.sftp.download.enabled=true
# aineisto muodostuu, joka sunnuntai 6.35
tau.download.cron=0 40 6 * * SUN
tau.sftp.download.hostname=download.sftp.host.fi
tau.sftp.download.username=username
tau.sftp.download.dir=/testi/dir
tau.sftp.download.filename=sap _REPORT_.*\\.CSV
tau.sftp.download.privateKeyFile=
tau.sftp.download.knownHostsFile=
tau.file.download.path=/SAP/TAU
tau.file.download.name=sap_users.csv
tau.file.archive.after.download=false

tau.db.update.enabled=true
tau.file.archive.after.update=true
tau.file.charset=windows-1252

tau.cleanup.enabled=true
tau.ydb.cleanup.cron=0 45 6 * * SUN
tau.ydb.cleanup.keep.batches=1

#-----------------------------
# TAMK
# TAMKin aineistot alkavat muodostua vasta alkuvuodesta 2020
#-----------------------------
tamk.sftp.download.enabled=false
tamk.download.cron=0 40 6 * * SUN
tamk.sftp.download.hostname=different.download.sftp.host.fi
tamk.sftp.download.username=
tamk.sftp.download.dir=/testi/dir
tamk.sftp.download.filename=sap _REPORT_.*\\.CSV
tamk.sftp.download.privateKeyFile=
tamk.sftp.download.knownHostsFile=
tamk.file.download.path=/SAP/TAMK
tamk.file.download.name=sap_users.csv
tamk.file.archive.after.download=false

tamk.db.update.enabled=false
tamk.file.archive.after.update=true
tamk.file.charset=windows-1252

tamk.cleanup.enabled=false
tamk.ydb.cleanup.cron=0 45 6 * * SUN
tamk.ydb.cleanup.keep.batches=1
```

Liite 5. ydb-nasgroup-sapkayttajatiedot-integraation esimerkkikonfiguraatiotiedosto

```
# Example of the cfg file
tau.cron=0 50 6 * * SUN
tau.enabled=true
# laita tahan temp-kansio, arkistointi tapahtuu taman alle kansioon archive
tau.file.temp.toDir=/files/sap/tau
tau.hr.report.role= SAP_RAPORTIT_LUKU_JA_KIRJOITUS
# laita tahan verkkolevyn osoite
tau.nas.group.directory=

# TAMKin aineisto alkaa muodostumaan vasta 2020 alkuvuodesta
tamk.cron=0 50 6 * * SUN
tamk.enabled=false
tamk.file.temp.toDir=/files/sap/tamk
tamk.hr.report.role= SAP_RAPORTIT_LUKU_JA_KIRJOITUS
# laita tahan verkkolevyn osoite
tamk.nas.group.directory=
```