Sasu Mikonranta

Keskitetyn integraatiojärjestelmän valvonnan kehittäminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

23.12.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Sasu Mikonranta Keskitetyn integraatiojärjestelmän valvonnan kehittäminen 46 sivua + 1 liite 23.12.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Software Specialist Sami Pikulinsky Lehtori Juha Kämäri

Tämän insinöörityön tarkoituksena oli selvittää kehitysmahdollisuuksia keskitetyssä integraatioissa käytettävän sovellusadapterin valvontaan. Selvityksen esille nostamista ehdotuksista valittaisiin yksi jatkokehitykseen ja tämä toteutettaisiin prototyyppinä ensin testiympäristöön ja myöhemmin tuotantoon. Työn tavoite oli parantaa sovellusadapterin nykyistä valvontaa ja tällä havaita virhetilanteita nykyhetkeä tehokkaammin. Työssä pohjustetaan myös järjestelmäintegraation periaatteita, sekä kuvataan integraatioissa käytettäviä komponentteja.

Työssä keskityttiin ongelmaan, kuinka kyseiseen toimintaympäristöön saataisiin luotua mahdollisimman riippumaton, yksinkertaisesti implementoitava, ongelmia korjaava valvontaratkaisu. Ratkaisumahdollisuuksia rajoittivat käytettävissä olevat resurssit, sekä ympäristön rajoitukset. Ratkaisuvaihtoehtoja käytiin kaupallisista vaihtoehdoista omakehittämiin ratkaisuihin ja arvioitiin näiden toiminnallisuutta.

Ratkaisu toteutettiin hyödyntämällä integraatiokeskuksessa olevien valvontaohjelmistojen ominaisuuksia, sekä luomalla automatisoitu valvontaproseduuri ympäristöön. Nykytoiminnallisuuteen lisättiin ajastetulla skriptillä toteutettu valvonta, jonka tehtävänä on tarkastaa määritettyjen sovellusadapterien tilaa, hyödyntäen adapterin tuottamaa lokidataa.

Ratkaisua käytettiin työn loppuvaiheessa testiympäristön valvojana. Toteutus laajennetaan myöhemmin adapteri kerrallaan integraatiokeskuksen tuotantoympäristöön.

Avainsanat	järjestelmäintegraatio, valvonta, sovellusadapteri

Author(s) Title Number of Pages Date	Sasu Mikonranta Development of Surveillance for a Centralized Integration System 46 pages + 1 appendix 23 December 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Sami Pikulinsky, Software Specialist Juha Kämäri, Senior Lecturer

The aim of this engineering thesis was to investigate the possibilities of developing the surveillance of an application adapter used in a centralized system integration environment. One of the proposals would be selected for further development and this chosen solution would be produced as a prototype first to a test environment and later into production. The goal of this thesis was to improve the existing monitoring of the application adapter and to detect error situations more effectively. This work also introduces the basic principles of system integration, and describes the components used in it.

The study was focused on how to create an independent surveillance solution which could be implemented in to the existing environment in the simplest way possible. The solution possibilities were restricted by available resources and environment restrictions. Both commercial and self-developed solutions were investigated and evaluated.

The chosen solution was created with the help of the existing features of the integration center, and by adding a new automated monitoring procedure. A timed monitoring script was added in to the environment, whose task was to check the status of designated integration adapters using the adapter's log information.

The solution was used in a test environment at the final stage of this thesis. Later it will be expanded one adapter at a time to the production environment.

Keywords	system integration, surveillance, application adapter

Sisällys

Lyhenteet ja käsitteet

EAI	Enterprise Application Integration – Organisaation järjestelmäintegraatio. Termiä käytetään puhekielessä myös integraatiokeskuksesta.
MQ	Message Queue – Sanomajono. Objekti jonka kautta välitetään JMS-sanomia. Käytetään myös viitatessa IBM-välittäjäohjelmistoon IBM MQ:hun.
QM	Queue Manager – Jonomanageri. Termiä käytetään IBM:n jononhallintaohjelmistossa.
JMS	Java Messaging Service – Java EE:n sisältämä viestinvälitysrajapinta.
ESB	Enterprise Service Bus – Palveluväylä.
SOA	Service Oriented Architecture – Palvelukeskeinen arkkitehtuuri.
API	Application Programming interface – Ohjelmointirajapinta.
MOM	Message Oriented Middleware – Sanomakeskeinen välitysohjelmisto.
SaaS	Software as a Service – Ohjelmisto palveluna.
Legacy	Vanhentunut käytössä oleva järjestelmä. Mahdollisesti vanha, vielä toiminnassa oleva järjestelmä, joka on toteutettu käyttäen vanhempia menetelmiä tai ohjelmointikieliä.
HTTP	Hypertext Transfer Protocol - Hypertekstin siirtoprotokolla.
HTTPS	Hypertext Transfer Protocol Secure – Suojattu hypertekstin siirtoprotokolla.
FTP	File transfer protocol – Tiedonsiirtoprotokolla.
SFTP	Secure file transfer protocol – Suojattu tiedonsiirtoprotokolla.
XML	Extensible Markup Language – Standardoitu rakenteellinen kuvauskieli.

REST	Representational State Transfer – Arkkitehtuurimalli, jonka toiminta perustuu HTTP-protokollaan.
IIB	IBM Integration Bus – IBM:n tuottama välittäjäohjelmisto.
Crontab	Cron-ajastuspalvelua ohjaava ohjelmisto, Unix-pohjaisissa järjestelmissä.
Python	Ohjelmointikieli.
Regex	Regular expression. Säännöllinen lauseke. Käytetään esimerkiksi määritetyn merkkijonon tunnistamisessa toisen merkkijonon sisältä.

1 Johdanto

Yritysten jatkuvasti kasvava järjestelmien määrä on luonut tarpeen saada yksittäisen järjestelmän informaatio hyödynnettäväksi yhä useampien järjestelmien välillä. Järjestelmien lisääntyessä näiden keskinäinen tiedonjakotarve kasvaa, mutta datan hyötyä ei saada enää maksimoitua käyttämällä sitä vain järjestelmässä, johon se on alun perin luotu. Yritysfuusiot ja uudet järjestelmät luovat vaatimukset uudistumiselle, mutta kuinka tämä käytännössä voidaan toteuttaa?

Näihin järjestelmien yhteistoiminnan haasteisiin on vastaus löydetty järjestelmien välisistä integraatioista. Monesti kuitenkaan järjestelmäintegraation termi ei sano paljoa tavalliselle kuluttajalle, tai välttämättä edes sovelluskehittäjälle. Integraation toimiessa täydellisesti se on parhaimmillaan lähes huomaamaton. Integraatiot kuten muukin tekniikka ovat kuitenkin alttiina virhetilanteille, joten toimintaympäristön aktiivinen valvonta on tärkeä osa nykyaikaisten järjestelmäympäristöjen kanssa työskentelyä. Integraatioiden toiminnan valvonnalla varmistetaan integraatiota hyödyntävien järjestelmien optimaalinen toiminta ja näin ollen tekniseltä osalta turvataan ratkaisua käyttävän yrityksen liiketoimintaa.

Tässä insinöörityössä keskitytään löytämään kehitysmahdollisuuksia järjestelmäintegraatiossa käytettävän rajapintakomponentin valvonnalle. Rajapintakomponentti eli sovellusadapteri toimii keskitetyssä integraatiomallissa integraatiokeskuksen ja tämän ulkopuolisten järjestelmien välisenä tiedonsiirtorajapintana. Työn tarkoitus on selvittää mahdollisuuksia laajentaa sovellusadapterin olemassa olevaa virhevalvontaa, pyrkimyksenä luoda lisävalvontakerros nykyiseen toimintaympäristöön. Työn tavoitteena on löytää ratkaisumalli, jonka tuloksena virhetilanteet saataisiin nopeammin ja automatisoidusti selville. Työssä arvioidaan eri ratkaisumalleja ja näiden implementointimahdollisuuksia nykyiseen toimintaympäristöön, rajoitteet ja resurssit huomioiden. Työ toteutetaan Digia Finland Oy:lle selvitystyönä mahdollisista valvonnan kehittämiskeinoista. Työssä tutustutaan tuotteistettuihin, sekä kehitystyön aikaansaamiin valvontaratkaisuihin. Valitusta valvontaratkaisusta toteutetaan testiympäristöön soveltuvuusselvityksenä prototyyppi.

Valvonnan kehittämisen kohteena oleva sovellusadapteri on komponentti keskitetyssä integraatiojärjestelmässä. Sovellusadapterin tehtävänä on toimia rajapintana palvelimen ja integraatiokeskuksen välillä, osana keskitettyä integraatiomallia. Sovellusadapteria

käyttävät yksittäiset palvelimet, joilla on tarve tiedonsiirtoon integraatiokeskuksen ja tähän liittyneiden muiden osapuolten välillä. Työssä tutustutaan myös järjestelmäintegraation perusteisiin, jotta näkemys sovellusadapterin sijoittumisesta ympäristöön ja tämän roolista osana integraatiota selkeytyisi.

Työ alkaa tutustumisella järjestelmäintegraation perusteisiin ja tämän yhteydessä käytettäviin teknologioihin. Tämän jälkeen työssä lähdetään syventymään tarkemmin integraatiokeskuksen, sekä sovellusadapterin toimintaan. Pohjustuksen jälkeen kuvaamme nykytilannetta sekä määrittelemme ympäristön ongelmia. Selvitys jatkuu ongelman määrittelyn jälkeen ratkaisuvaihtoehtojen kartoituksella. Työssä tutkinnan alle otetaan kaksi tuotteistettua ratkaisuvaihtoehtoa sekä pohditaan mahdollisia kehitystyön tuottamia ratkaisuja. Selvityksen lopputuloksena toteutamme testiympäristöön prototyypin valitusta ratkaisusta ja arvioimme tämän toteutusta. Loppukappaleessa arvioidaan yhteenvetona insinöörityön tekoa, työn opettamia asioita, valitun ratkaisun toimivuutta, sekä tälle asetettujen tavoitteiden täyttymistä.

2 Järjestelmäintegraatio

Järjestelmäintegraatio voidaan yleistoimintaperiaatteiltaan kuvailla kahden tai useamman järjestelmän väliseksi yhteydeksi toisiinsa. Järjestelmäintegraatiota ei voi rajata tiettyihin tuotteisiin tai teknologioihin, vaan se tulisi mieltää ennemmin järjestelmäkokonaisuuksien yhteen saattavaksi hallintamalliksi [1, s.13]. Hyödyntämällä yksittäisen järjestelmän tiedot yhä useammassa järjestelmässä saavutetaan näiden järjestelmien tuottamalle informaatiolle maksimaalinen hyöty.

Informaatio ei kuitenkaan ole usein suoraan käytettävissä jaettaessa sitä eri järjestelmien välillä. Järjestelmäintegraatiolla on pyritty löytämään tähän ratkaisu saattamalla tieto sopivaan muotoon, osana tiedonvälitystä. Loppukäyttäjien käyttämät palvelut, kuten verkkopankit tai -kaupat, saattavat sisältää useita, jopa kymmeniä eri taustajärjestelmiä. Näiden taustajärjestelmien sulava keskinäinen toiminta voi luoda palvelulle tärkeän kilpailuedun, muiden palveluntarjoajien toimintaan verrattuna. Integraatiot ovatkin yrityksille tärkeä osa tietojärjestelmien tehostamista ja automatisointia. Prosessien kehitys kohti automatisoidumpaa järjestelmäympäristöä tuo yrityksen liiketoiminnalle suurta lisäarvoa.

Tässä luvussa käsitellään järjestelmäintegraation yleisiä periaatteita ja malleja, syventymättä vielä tiettyjen mallien tuomiin tuotteistettuihin yksityiskohtiin.

2.1 Integraatiomallit

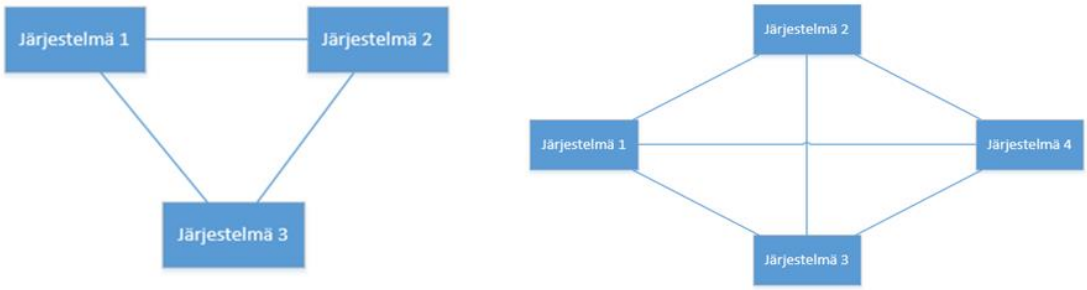
Integraatiomallit ovat arkkitehtuuritason ratkaisuja siitä, kuinka järjestelmien välinen integraatio on toteutettu. Tässä yhteydessä esittelemme kolme erilaista integraatioarkkitehtuuriratkaisua sekä pohjustamme näiden mallien soveltuvuutta eri tilanteisiin. Luvussa käydään myös läpi, kuinka erilaiset integraatiomallit ovat kehittyneet ajan myötä ja mihin ratkaisuihin eri mallit pohjautuvat sekä millaiset ovat tulevaisuuden integraatioratkaisujen kehitysnäkymät.

2.1.1 Point-to-point

Yksinkertaisin ja varhaisin integraation muoto on niin kutsuttu point-to-point-integraatio, jossa kaksi järjestelmää integroidaan suoraan toisiinsa (kuva 1). Tämän kaltainen integraatiotarve saattaa esiintyä yrityksen ensimmäisenä tarpeena integraatioratkaisulle. Esimerkkinä kivijalkamyymälää pyörittävä yritys perustaa verkkokaupan ja tästä syntyy tarve kivijalkamyymälän varastojärjestelmän integroimiseen verkkokaupan tuotesaatavuuden hallintaa varten. Pienen skaalan järjestelmäkokonaisuuksissa point-to-point-integraatio voi olla järkevä ja kustannustehokas vaihtoehto.

Kuva 1 - Kahden järjestelmän välinen Point-to-Point-integraatio

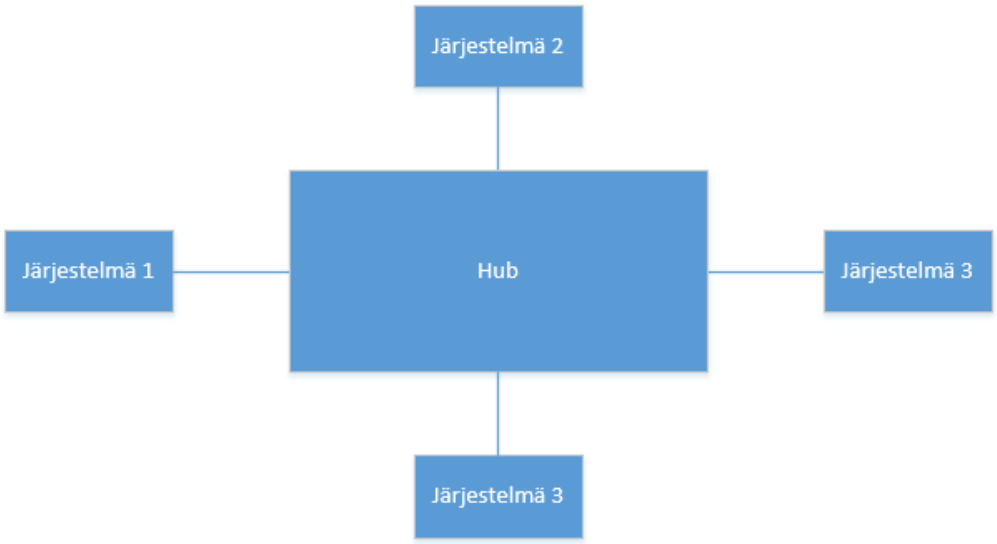
Osapuolten määrän kasvaessa, point-to-point-integraatioiden ylläpidettävyys sekä kustannustehokkuus laskevat [1, s.66]. Laskiessamme yksittäisten yhteyksien määrän järjestelmien välillä, kolmen järjestelmän integraatiossa näitä on yhteensä kolme. Mikäli neljäs järjestelmä lisätään kokonaisuuteen, syntyy järjestelmien välisiä integraatioita jo kuusi (kuva 2). Yli kolmen järjestelmän point-to-point-integraatio menettää jo tehokkuutensa, mikäli järjestelmien integraatiotarve on hyödyntää kaikkien integroituneiden osapuolten informaatiota. Suurempia integraatiotarpeita varten point-to-point-integraatiot eivät ole kannattavia huonon skaalautuvuutensa vuoksi. Osapuolten ollessa hajautettuna, myös integraation valvonta vaikeutuu tämän kasvaessa. [2.] Tästä johtuen kehityssuunta laajempien kokonaisuuksien hallintaan löydettiin keskittämällä integraatioita kulkemaan yhdyspisteen, eli hubin kautta.



Kuva 2 - Kolmen ja neljän järjestelmän välinen integraatio

2.1.2 Keskitetty integraatio

Kuten edellisessä kappaleessa totesimme, point-to-point-integraation kannattavuus laskee merkittävästi järjestelmien määrän kasvaessa yli kolmen. Kasvavien järjestelmäkokonaisuuksien hallintaan kehitettiin keskitetyn integraation hallintamalli. Keskitetyssä integraatiossa keskenään yhteydessä olevat järjestelmät ovat integroitu keskitetyn hubin kautta (kuva 3). Näin integraatiota hyödyntävät ulkoiset osapuolet tarvitsevat luoda vain yhteen osapuoleen yhteydet, keskitetyn hubin toiminnallisuus hoitaa ohjaukset oikeille osapuolille.



Kuva 3 - Integraatiokeskus järjestelmien yhdyspisteenä

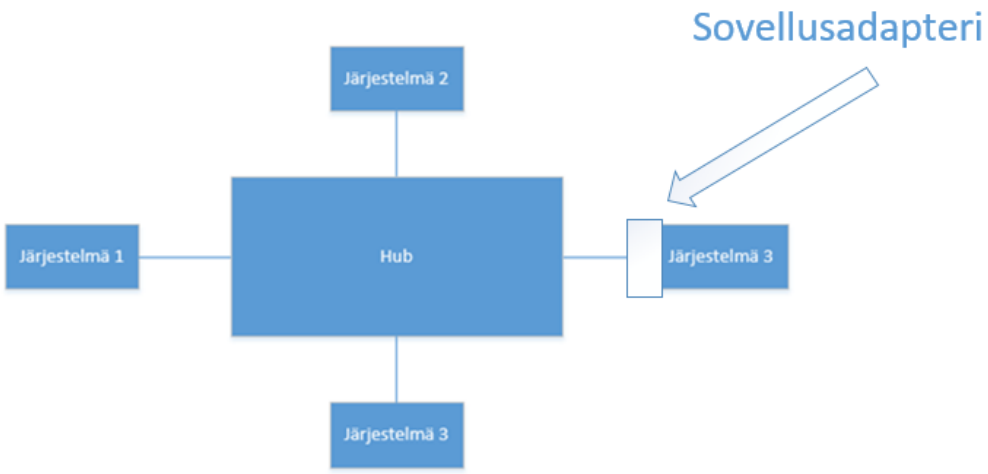
Yritysten järjestelmäintegraation keskitettyä hubia kutsuaan yleiskielessä integraatiokeskukseksi. Puhekielessä näistä saatetaan käyttää myös termiä EAI, vaikka termi yleisesti viittaakin yrityksen sisäisten järjestelmien integraatioon, eikä pelkästään integraatiokeskukseen. Mallista puhutaan myös hubina ja adaptereina (hub and spoke) [3]. Integraatiokeskus sisältää useita sanomavälityksen kannalta tärkeitä väliohjelmistoja (MOM, message oritented middleware), joilla on erinäisiä tehtäviä osana integraatiokeskusta. Väliohjelmistot toimivat informaation välityskerroksena, käyttöjärjestelmätason ja sovellustason välissä [4.]. Näitä väliohjelmistoja tuottavat useat eri toimittajat kuten esimerkiksi IBM, Microsoft, Apache, Oracle, SAP, Red Hat sekä monet muut. Komponentteja osana hubia on usein ainakin viestinvälittäjä (Message broker), tietokanta sekä mahdolliset jonojenhallintatyökalut JMS. Integraatiokeskuksen ulkopuoliset osapuolet ovat yhteydessä integraatiokeskukseen käyttäen eri teknologioita, joihin voi kuulua esimerkiksi JMS/MQ, HTTP/HTTPS, FTP/SFTP sekä sähköpostisanomat.

Keskitetyn integraation etuina ovat luonnollisesti uusien osapuolien helppo liitettävyys, laajennettavuus sekä sanomaliikenteen valvonnan keskittäminen. Koska kaikki liikenne tapahtuu yhden keskuksen kautta, voidaan sanomaliikennettä kontrolloida ja valvoa keskitetysti. Uusien järjestelmien lisäys ja poisto ovat keskitetyssä ratkaisussa helposti to-

teutettavissa, koska joissakin tilanteissa uusia integraatioita määrittäessä, toinen integraation osapuoli saattaa olla jo liittynyt integraatiokeskukseen. Yrityksen liiketoiminnan näkökulmasta sanomien reaaliaikainen valvontamahdollisuus antaa nopean mahdollisuuden reagoida ongelmiin ja täten välttää katkoista aiheutuvia liiketoiminnan tappioita. Aineistolle voidaan toteuttaa myös monipuolisempia käsittelyitä keskitetyssä palvelinkokonaisuudessa. Integraatiokeskus sisältää ominaisuuksiltaan usein aineiston muunnos ja reititystyökalut, joiden avulla aineistot saadaan vastaanottajalle sopivaan muotoon. Usein sanomat noudattavat sisällöltään määritettyjä standardeja kuten XML, CSV tai JSON.

Toisaalta keskitetty ratkaisu tuo myös riskin liikenteen ollessa yhden järjestelmäkokonaisuuden varassa. Tätä riskistä käytetään nimitystä ”Single point of failure” [1, s.68]. Koska riski on yleisellä tasolla hyvin tiedostettu, integraatiokeskuksen järjestelmillä on useita varajärjestelmiä, mikäli ongelmia tiedonsiirrossa esiintyisi. Sanomaliikenteen keskittäminen tuo myös riskin hubin toimia pullonkaulana kasvavassa tiedonsiirrossa. Jokainen sanoma kulkee integraatiokeskuksen kautta ja käyttää samoja resursseja välityksessä. Tämä voi aiheuttaa point-to-point-integraatioon verrattuna viivettä, koska sanomat joutuvat kulkemaan useamman yhteyspisteen sekä hubin ohjausten läpi. Mikäli joku hubin yhteinen yhteyspiste olisi katki tai jumissa, aiheuttaisi tämä viivettä kaikelle sanomaliikenteelle, jotka kyseistä pistettä käyttävät. Näin ollen integraatiokeskuksen tehokkuus on hyvin paljon sidoksissa tässä käytettävän raudan (hardware) tehoon.

Integraatiokeskuksen tiedonsiirrossa hyödynnettäviä teknologioita on monia, mutta tässä yhteydessä perehdymme tarkemmin sanomajonoihin (Message queue) perustuvaan JMS (Java Messaging service) -sanomavälitykseen ja tähän liittyvään sovellusadapterin valvontaan. Keskitetyssä integraatioratkaisussa sovellusadapteri sijoittuisi ulkoisen järjestelmäosapuolen palvelimelle ja toimisi palvelimen rajapintana integraatiokeskuksen (hub) kanssa (kuva 4).



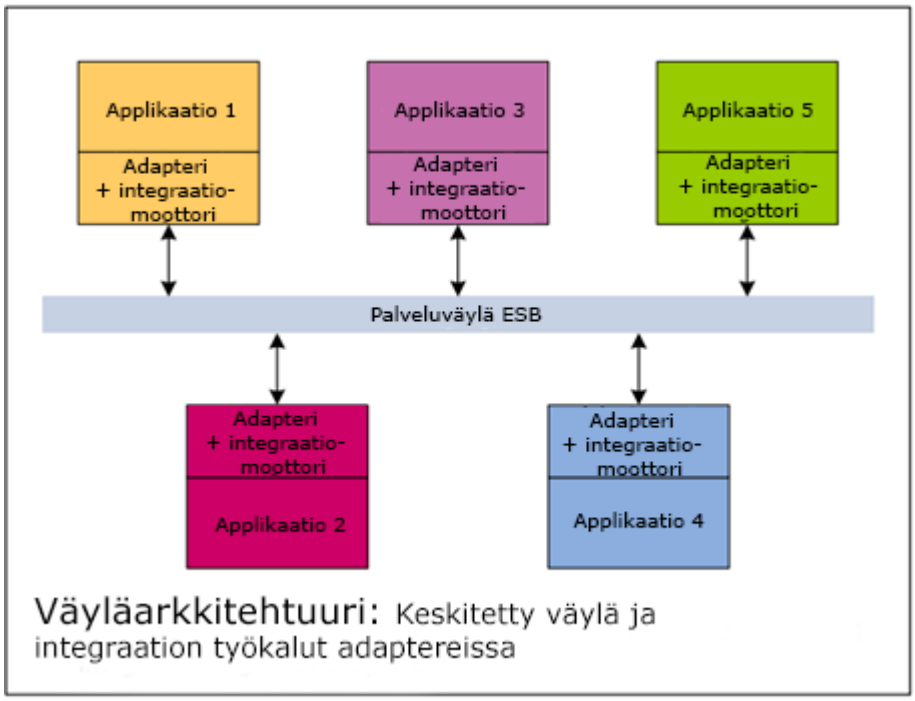
Kuva 4 - Sovellusadapterin sijoittuminen keskitettyyn integraatioympäristöön

2.1.3 Integraatioiden kehityssuunta

Näkökulma moderneissa integraatioissa on siirtynyt 2000-luvulla palvelukeskeisempään ajattelutapaan (kuva 5). Palvelukeskeisessä mallissa liiketoiminnan prosesseja on suunniteltu toimimaan itsenäisinä palveluina. Näistä palvelukeskeisistä ratkaisumalleista ryhdyttiin käyttämään nimitystä palvelukeskeinen arkkitehtuuri SOA (Service Oriented Architechture). Kaiken suunnittelun keskipisteenä on näin ollen palvelut, joiden toiminta mahdollistetaan teknisten ratkaisujen avulla. [5.]

Kuva 5 - Integraatioiden kehityssuunta [5.]

Ilmentymänä kohti palvelukeskeisempää näkökulmaa syntyi keskitetystä hub-mallista kehittynyt palveluväylä ESB (Enterprise Service Bus) (kuva 6). Malli perustuu keskitettyyn palveluväylään, johon applikaatiot kytkeytyvät adaptereilla. Nämä applikaatiot voivat välittää sanomia palveluväylälle käyttäen standardoituja yhteensopivia sanomaformaatteja. Suurin osa aineistokäsittelystä suoritetaan palveluväylän ulkopuolella, jolloin sanomien kuormitus ei keskittyisi palveluväylälle. Palveluväylän rooli on pysyä kevyenä, mutta mahdollistaa samoja ominaisuuksia kuin raskaammat integraatioratkaisut. [6.] Viemällä raskas käsittely palveluväylän ulkopuolelle erottuu ESB skaalautuvuudeltaan edukseen verrattuna perinteiseen keskitetyn integraation integraatiokeskus-malliin.



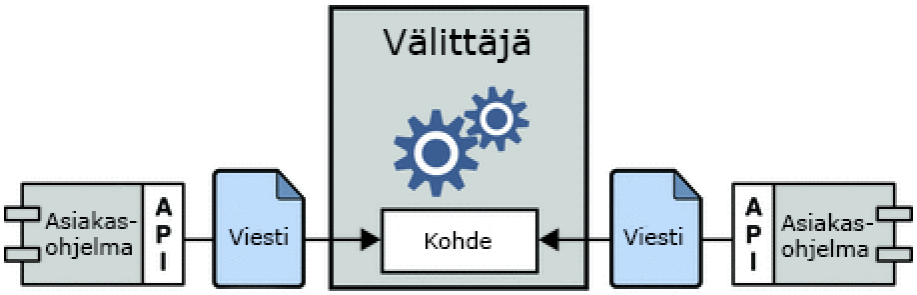
Kuva 6 - Enterprise Service Bus esimerkki [6.]

Nykyhetken ja tulevaisuuden näkymät integraatioiden parissa ovat kuitenkin mikropalvelukonseptiin pohjautuvia itsenäisiä ja tilattomia palveluita [7]. Palveluilta haetaan yhä kevyempiä ratkaisumalleja, joita haetaan esimerkiksi REST API (Application Programming Interface) -rajapintojen avulla. Mikropalvelut ovat ns. pilkottuja palveluita pienempinä ja itsenäisinä osina. Kun palvelukeskeinen arkkitehtuuri tarjoaa esimerkiksi henkilöstöjärjestelmän kokonaisena yhtenäisenä palveluna, mikropalveluiden arkkitehtuurimallissa tämä voitaisiin jakaa pienempiin osapalveluihin, kuten henkilötieto-, palkka-, rekrytointihallinta.

Mikropalveluiden lisäksi myös useat ohjelmistotoimittajat ovat siirtyneet tarjoamaan omia ratkaisujaan pilvipohjaisina palveluluina. Tämä mahdollistaa yrityksille uusien järjestelmien käyttöönoton, ilman omien konesalien tai lisenssien vaatimuksia. Tätä ohjelmistojen palveluna tarjottavaa pilvitoimitusmallia kutsutaan termillä SaaS (Software as a Service). Rahoitusmallina näillä pilvipalveluilla usein käytetään käyttöön pohjautuvaa mallia. Täten pienemmillä yrityksillä on helpompi ottaa jokin ennen hyvin kallis ohjelmisto käyttöön jo yrityksen alkuvaiheissa. Luomalla toisistaan riippumattomia palveluita, järjestelmien välinen monoliittisuus purkautuu ajan myötä ja järjestelmien päivitys ainakin teoriassa on paljon yksinkertaisempaa tulevaisuudessa. Integraation rooli näissä malleissa voisi keskittyä rajapintojen määrittämiseen ja hallintaan sekä legacy-järjestelmien tukemiseen [8.].

2.2 JMS-sanomavälitys

Java Message Service on Java Enterprise Editionin sisältämä viestinvälitysrajapinta, sanomien välitykseen asiakasohjelmien (Client) välillä. JMS sanomavälitys perustuu löyhästi kytkettyjen järjestelmien (loosely coupled systems) väliseen, asynkroniseen sanomavälitykseen. Asynkronisella sanomavälityksellä tarkoitetaan, että lähettäjä voi kirjoittaa sanoman kohteeseen (jono tai aihe) ja sanoma säilyy kohteessa, kunnes sanoma on käsitelty tai se erääntyy. Sanomavälitys voi olla jonoihin perustuvaa suoraa välitystä (point-to-point) tai julkaisu- ja tilauspohjaista (publish - subscribe). JMS sanomavälityksessä keskeisiä osapuolia ovat lähettäjä, välittäjä ja vastaanottaja (kuva 7). Välittäjän rooli voi yksinkertaisimmillaan olla passiivinen jono, tai laajemmin jopa integraatiokeskus. Lähtökohtaisesti lähettävä osapuoli ei tunne vastaanottavaa osapuolta, vaan perustuen lähettävän aineiston metadataan tai välitettävään kohteeseen, sanoma ohjautuu oikealle vastaanottajalle.



Kuva 7 – JMS-sanomavälitys [9.]

Jonopohjaisessa asynkronisessa sanomavälityksessä lähettäjän kirjoitettua sanoman jonoon sanoma ei poistu jonosta ennen kuin vastaanottava osapuoli lukee sen, tai sanoma erääntyy. Jonopohjainen yhteys myös varmistaa, että sanoma toimitetaan vastaanottajalle vain kerran. [9.]

Usealle kohteelle sanoma voidaan välittää julkaisu- ja tilauspohjaisella jakelumekanismilla, joka perustuu julkaisija- ja tilaajaosapuoliin. Lähetyksessä sanoma siirretään jonon sijaan julkaisuun, josta sanoma lähetetään aktiivisille tilaajaosapuolille. [10.]

2.2.1 JMS-sanoma

JMS-sanoma on objekti, joka sisältää sanoman metatiedot, sekä itse kuljetettavan datan. Sanoma rakentuu kolmesta osasta, jotka ovat otsake (header), asetukset (properties) sekä kuorma (payload) (kuva 8).

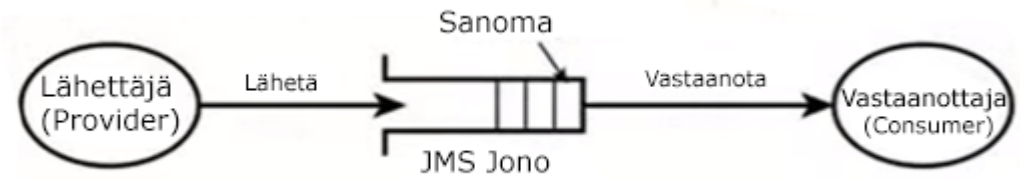
Kuva 8 – JMS-sanoman rakenne [11.]

Tavanomaisesti otsake sisältää JMS-yhteyden kannalta olennaisimmat tiedot, jotka asetetaan automaattisesti. Näitä otsikkotietoja voidaan laajentaa käyttäjän asettamilla asetustiedoilla, jotka ovat valinnaisia, mutta olennaisia varsinkin keskitetyssä integraatiossa käytettävien reititysten osalta. Asetustiedoissa voi esimerkiksi määrittää tiedon lähettäjästä, vastaanottajasta tai aineiston tyypistä. Sanoman otsake ja asetukset rakentuvat XML-formaatin elementeistä ja näiden sisältämästä datasta, kun taas kuorma voi olla

formaatiltaan vapaampaa. XML ja muut aineiston rakennetta kuvaavat kielet, kuten JSON ja CSV, ovat suosittuja integraatioissa käytettäviä sanomien kuorman formaatteja. Näiden helpon hallittavuuden vuoksi. Integraatiotyökaluihin onkin oletuksena usein rakennettu muuntimia yleisimmille formaateille keskenään. Mikäli sanoman kuorma on standardoitujen formaattien mukaista, on tämän muunnokset eri muotoihin helpommin suoritettavissa ja aineiston validointi yksinkertaisempaa.

2.2.2 Sanomajono

Sanomajonot (Message Queue) ovat objekteja, joka säilövät sanomia niiden välityksen ajan, kahden osapuolen välillä (kuva 9). Lähettävä osapuoli kirjoittaa sanoman jonoon, jonka jälkeen sanomaa säilytetään jonossa, kunnes se luetaan jonosta pois tai se erääntyy. Sanomajonot eivät tee aineistolle muuta kuin säilövät tätä välityksen ajan. Sanomajonot voivat olla määritettyinä keskusmuistiin tai kovalevylle, vaikka jonoihin saapuvat sanomat jäävät odottamaan jonoon niiden poislukemista, ideaalissa tilanteessa jono on aina tyhjä. Sanomia ei ole yleisesti tarkoitus varastoida jonoihin. Olennaista jonoa käyttävässä sanomavälityksessä on, että lähettäjän välitettyä sanoman jonoon, joutuu vastaanottava osapuoli noutamaan sanoman jonosta. Jono itsessään ei lähetä sanomaa vastaanottajalle. Jonon rooli on toimia staattisena objektina, jolla itsellään ei ole varsinaisesti aktiivista toiminnallisuutta. [12.]



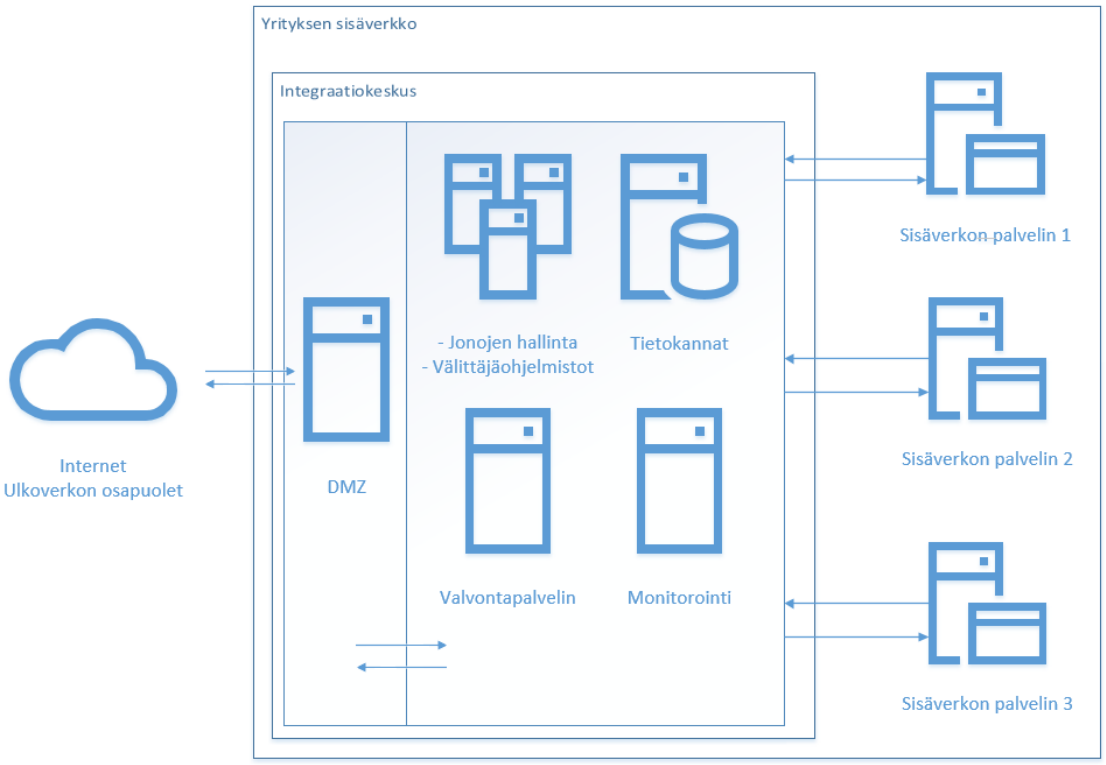
Kuva 9 - JMS Sanomajono [12.]

3 Integraatiokeskus

Olennaisena osana keskitettyä integraatiota on itse integraatiokeskus. Integraatiokeskus on keskitetyn integraation yhdyspiste, johon osapuolet ovat sovellusadaptereiden sekä muiden tekniikoiden avulla yhteydessä. Tässä luvussa kuvataan integraatiokeskuksen rakennetta ja tämän sisältämiä komponentteja.

3.1 Yleiskuvaus

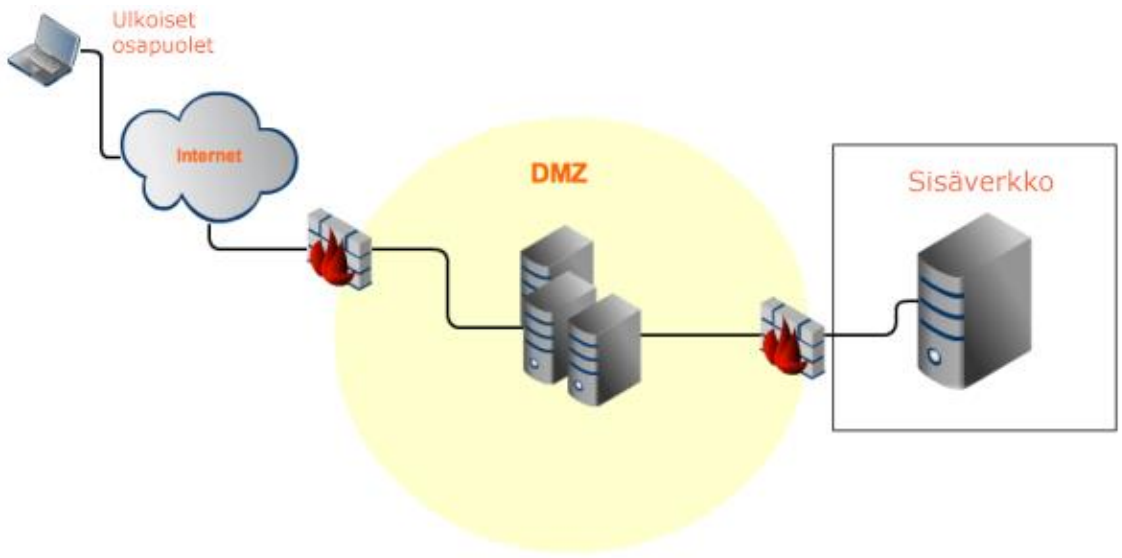
Integraatiokeskus on palvelinkokonaisuus, joka toimii ympäristön keskitettynä hubina ja joka sisältää useita väliohjelmistoja (Middleware). Integraatiokeskus käsitteenä on kuitenkin hyvin yleisen tason käsite, eivätkä sen yksityiskohdat ole tarkasti määriteltyjä. Olennaista kuitenkin on, että integraatiokeskus on keskitetyn integraation hubi. Yksinkertaisimmillaan JMS-sanomia käsittelevä integraatiokeskus voi sisältää yhden palvelimen, joka sisältää sanomia käsittelevän välittäjän (Message broker) sekä jonojenhallintaohjelmiston. Välittäjän rooli on toimia sanomien reitittäjänä ja mahdollisten aineistomuunnosten toteuttajana sekä hyödyntää jonohallintaohjelmiston ylläpitämien jonojen aineistoja. Kuitenkin monipuolisempi integraatiokeskus rakentuu lukuisista eri ohjelmistoista sijoittuen useille eri palvelimille (kuva 10). Vaikka periaatetasolla lähes kaikki integraatiokeskuksen toiminnallisuus pystyttäisiin sijoittamaan yhdelle palvelimelle. Tämä kuormittaisi palvelinta huomattavasti ja kasvattaisi yksittäisen vikaantumispisteen riskiä [1, s.105].



Kuva 10 – Esimerkki integraatiokeskuksen rakenteesta

Muut verkon palvelimet voivat olla yhteydessä integraatiokeskukseen käyttäen useita eri siirtoprotokollia riippuen valituista integraatio-ohjelmistoista. Tietoturva integraatiokeskuksen ja tähän liittyvien osapuolten välillä on usein turvattu useiden palomuurien sekä varmenteiden avulla. Oletuksena lähes kaikki liikenne palvelinten välillä on suojattua. Välitettävä aineisto saattaa olla sisällöstä riippuen myös kryptattua, joka kasvattaa tietovirran tietoturvaa.

Laaja integraatiokeskus voi pitää sisällään myös sanomien valvontaan käytettäviä käyttöliittymiä, sanoma-arkistoja tai käytettävästä integraatioalustasta riippuvia muita työkaluja. Integraatiokeskuksen ulkopuoliset järjestelmät voivat olla ohjelmistosta riippuen usealla eri protokolalla yhteydessä integraatiokeskukseen. Integraatiokeskus sijaitsee yleisesti yrityksen sisäverkossa ja yhteydet ulkoverkon (internet) osapuoliin on toteutettu DMZ-verkkoalueen (demilitarized zone) avulla. DMZ on tietoturvaa parantava verkkoalue yrityksen sisäverkon ja internetin välissä, joka lisää tietoturvatason palomuurien avulla ulkoverkon ja sisäverkon välille (kuva 11). [13.]



Kuva 11 - DMZ verkkoalue [13.]

3.2 Tietovirta

Tietovirraksi kutsutaan integraatiokeskuksen läpi kulkevaa yhteyttä, joka on yksilöity lähettäjä-aineistotyyppi-vastaanottaja-kohtaisesti. Aineistotyyppi on aineiston yksilöivä kuvaus, joka on uniikki jokaisella erityyppisellä aineistolla. Koska yhdellä aineistotyypillä

voi olla useampi lähettäjä tai vastaanottaja, ei tietovirtaa voida yksilöidä pelkästään tämän perusteella. Usein sanoman ohjaus perustuu vain tietoon lähettäjästä ja aineistotyypistä. Tämän tiedon avulla integraatiokeskuksen välitysohjelmisto osaa etsiä aineistolle oikeat tietovirrat ja tätä kautta vastaanottajat. Aineiston lähettäjän ei tarvitse tietää, kenelle aineisto lähetetään asynkronisessa tiedonvälityksessä. Integraatiokeskuksen rooli on pitää yllä tietoa tietovirroista, jolloin lähettävä järjestelmä vain huolehtii aineiston toimittamisesta integraatiokeskukselle. Uudet yhteydet määritetään aina uutena tietovirtana, olivat osapuolet integraatiokeskukselle ennestään tunnettuja tai tuntemattomia.

3.3 Integraatiokeskuksen väliohjelmistot

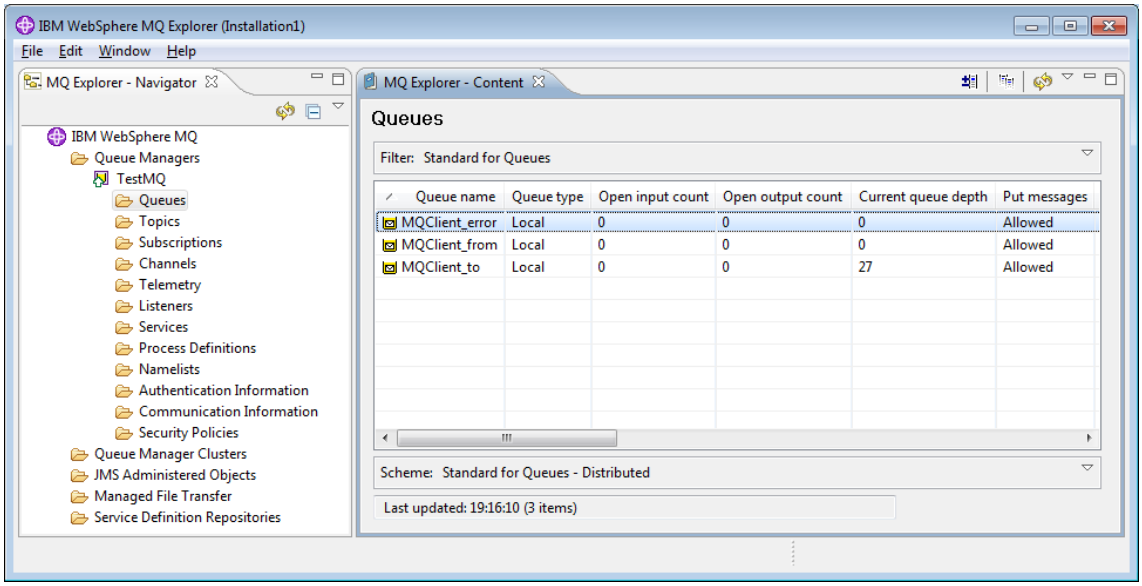
Integraatiokeskuksen teknisen toiminnallisuuden mahdollistavat integraatiokeskuksen palvelimille sijoitetut sanomavälitykseen tarkoitetut väliohjelmistot (Message oriented middleware). Väliohjelmistot ovat eri toimittajien valmistamia integraatio-ohjelmistoja, jotka voivat ajaa yksittäistä integraatioratkaisun osa-aluetta tai toimia kokonaisvaltaisena integraatioratkaisuna. Väliohjelmistoja tuottavat useat eri yritykset, joihin kuuluvat esimerkiksi kansainvälisistä IBM, Microsoft, SAP, Oracle, Apache ja suomalaisista Digia. Väliohjelmistojen ominaisuudet vaihtelevat tuotteesta riippuen ja saattavat vaatia muita väliohjelmistoja toimiakseen. Usein väliohjelmistot ovat kuitenkin hyvin keskenään yhteensopivia, koska ohjelmistot ovat luotuja erilaisten järjestelmien integroimiseen ja yleistasolla ajavat samankaltaista roolia. [1, s.149.]

3.3.1 Jonojen hallinta

Jonojen hallinta on yksi tärkeimmistä integraatiokeskuksen väliohjelmistojen toiminnoista. Tätä tehtävää varten on olemassa useita väliohjelmistoja, joita ovat esimerkiksi Apache Active MQ ja IBM MQ. Näistä esimerkkinä on IBM WebSphere tuoteperheen jonomanageriohjelmisto IBM MQ.

IBM MQ käyttää jonojen hallinnassa jonomanagereiden konseptia. Jonomanagerit (QM) ovat MQ:n sisältämiä jonojen yläobjekteja, joiden avulla hallinnoidaan jonoja ja osapuolten yhteyksiä jonoihin. JMS-sanomavälityksessä tämä tarkoittaa käytännössä sitä, että osapuolet ottavat yhteyden jonomanageriin ja lähettävät sanomia jonomanagerin alaisuudessa oleviin jonoihin. Jonomanagereita voi olla integraatiokeskuksessa useita ja

näiden välinen keskustelu tapahtuu kanavien avulla. Jonoja voidaan tarkkailla komentoriviltä tai visuaalisesti IBM WebSphere Message Explorerin avulla. (kuva 12). IBM MQ on määritellyt kahdeksan erilaista jonotyyppiä. Nämä voidaan kuitenkin yksinkertaistetulla tasolla luokitella paikallisiin ja viitejonoihin. Paikallinen jono on jonomanagerilla sijaitseva fyysinen objekti, johon sanomat voidaan kirjoittaa. Viitejonoja voivat olla paikalliseen tai toisen jonomanagerin jonoon viittaavat viitejonot. Sanomia voidaan lukea vain paikalliselle jonomanagerille määritetystä paikallisesta jonosta, mutta sanoma voidaan kirjoittaa myös toisella jonomanagerilla sijaitsevaan jonoon hyödyntäen viitejonoa.



Kuva 12 – IBM Websphere MQ Explorer - jonojen tarkkailunäkymä [14.]

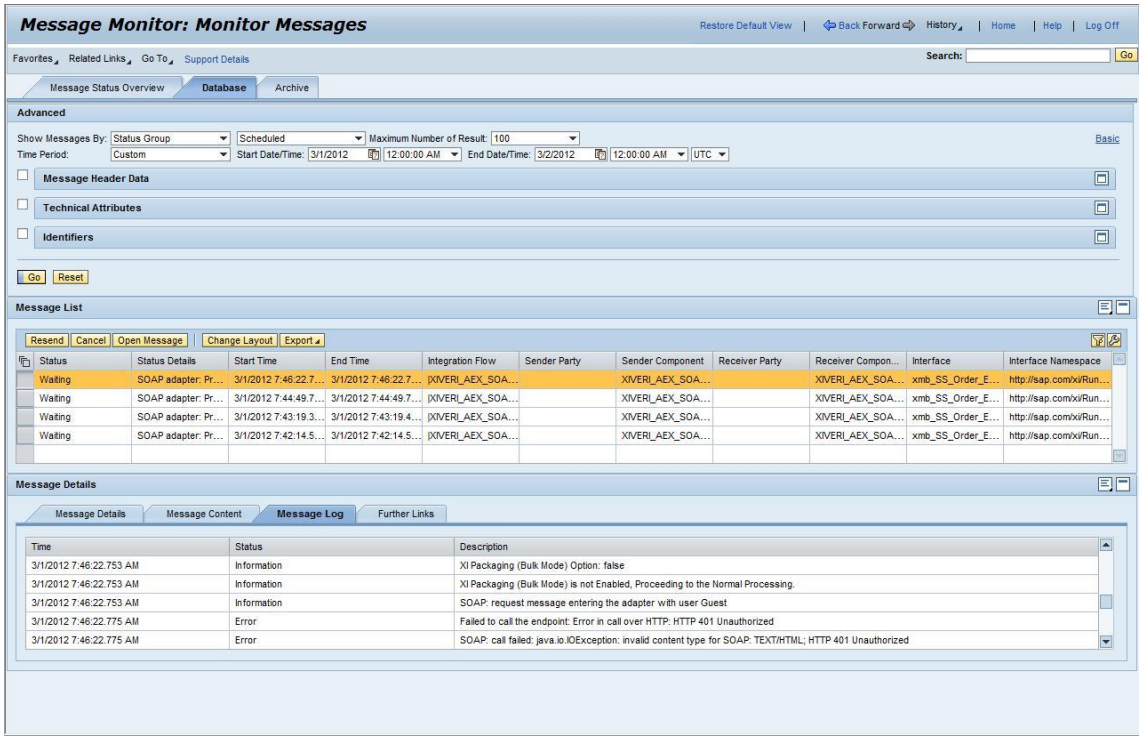
Pääasiallisesti JMS-sanomia lähettävät osapuolet eivät ota suoraan jonoon yhteyttä, vaan yhteys tehdään jonohallintasovelluksen kautta ja sanoma ohjataan kanavayhteyksien avulla hallintaohjelmiston alaisuuteen määritettyyn jono-objektiin.

3.3.2 Välittäjäohjelmistot

Välittäjäohjelmistoksi (Message broker) kutsutaan osapuolta, jonka tehtävänä on ohjata sanoma oikealle vastaanottajalle, sekä tehdä tälle mahdollisesti tarvittaessa sisältömuunnoksia tai validointeja. Välittäjäohjelmistot voivat ominaisuuksiltaan olla laajasti vaihtelevia valitun ohjelmiston ratkaisusta riippuen. Välittäjäohjelmisto on esimerkiksi IBM Integration Bus (IIB).

3.3.3 Monitorointi

Integraatiokeskuksen läpi kulkevaa sanomaliikennettä on tarve aktiivisesti monitoroida virhetilanteiden varalta. Monitorointia varten useat väliohjelmistot sisältävät jonkinlaisen valvontamahdollisuuden. Valvontamahdollisuus lokien kautta löytyy lähes jokaisesta sovelluksesta. Lokia visuaalisempaan monitorointiin tarvitaan kuitenkin käyttöliittymä. Integraatiotyökaluille löytyykin useita tuotteistettuja sanomaseurantaratkaisuja. Osa ratkaisuista voi olla väliohjelmistoon sisäänrakennettuja, kuten esimerkiksi SAP-ohjelmistolla (kuva 13). Sanomaseurannan avulla sanomien kulkua voidaan seurata käyttäjäystävällisemmässä muodossa, sekä virheet voidaan havaita reaaliaikaisesti. Sanomaseurannan ominaisuuksista riippuen sanomia voidaan tarkastella jopa sisällöltään. Monitorointia suorittavat usein palvelua ylläpitävän yrityksen palvelupisteiden valvontahenkilöstö.



Kuva 13 – Esimerkki sanomamonitoroinnista SAP-ohjelmistolla [15.]

4 Sovellusadapterin toiminta ja ympäristö

Sovellusadapterin paikallinen toimintaympäristö on palvelin, jolle adapteri on asennettu. Kuitenkin adapterin tehtävä on toimia sovelluksen ja integraatiokeskuksen välillä, joten myös integraatiokeskus on olennainen osa adapterin toimintaa. Tässä esimerkkitapauksessa kuvattu integraatiokeskus, jonka osana sovellusadapteri toimii sisältää IBM:n sekä Digian valmistamia integraatioväliohjelmistoja.

4.1 Sovellusadapteri

Sovellusadapteri on rajapintakomponentti integraatiokeskuksen ja tämän ulkopuolisten palvelinten välillä. Sovellusadapteri on Javalla toteutettu applikaatio, joka toimii palvelimella ympäristöstä riippuen Windows-palveluna tai Unix-pohjaisissa käyttöjärjestelmissä palveluprosessina (daemon). Sovellusadapterit ovat osapuolikohtaisia, jolloin jokainen sovellusadapteri vaatii uniikit konfiguraatiot. Sovellusadaptereiden tarkoitus on saada yksittäiset palvelimet ja näiden järjestelmät osaksi integraatiokeskusta ilman, että itse aineiston tuottavaan järjestelmään tai sovellukseen tarvitsee tehdä suuria muutoksia. Sovellusadapteri on myös hyvin monipuolisesti muokattavissa, joka tuo tälle mahdollisuuden räätälöityihin lisätoiminnallisuuksiin. Sovellusadapterin pääasiallinen toiminta on kuitenkin tämän muodostama tiedonsiirto ja raskaammat aineistokäsittelyt pyritään aina suorittamaan integraatiokeskuksen välittäjäohjelmistojen avulla.

Sovellusadapteri on vaihtoehtoinen sanomavälityskeino, jolle vaihtoehtoisia siirtotapoja ovat esimerkiksi FTP-, SFTP-lähetykset, tai HTTP-protokollan sanomavälitys. Sovellusadapteri hyödyntää tekniikkana JMS-sanomavälitystä jonojen avulla. Etuina sovellusadapterilla verrattuna suoriin tiedonsiirtoprotokoliin on mahdollisuus käsitellä aineiston sisältöä, automatisoida tiedonsiirto sekä parantaa sanomien kulkuvalvontaa adapterin sisältämien omien valvontatoimintojen avulla. Sovellusadapterilla pystytään suorittamaan aineistolle myös toimenpiteitä, kuten salauksen muodostus ja purku tai aineistomuunnoksia. Sovellusadapteri hyödyntää toiminnassaan sille luotuja moduuleja, jotka ovat tehty suorittamaan eri tehtäviä. Moduulien yksinkertainen konfigurointi mahdollistaa monimuotoisia yksilöllisiä sanomien käsittelyprosesseja.

4.2 Sanomavälitys sovellusadaptereiden välillä

Sovellusadapteri on integraatiokeskuksen ulkopuolisille palvelimille asennettava rajapintana toimiva komponentti. Tässä luvussa esittelemme kahden sovellusadapterin välisen sanomavälitysprosessin integraatiokeskuksen välittämänä. Integraatiokeskuksen rakenne ja palvelimet eivät ole vakiomuotoisia, vaan kyseisessä esimerkkitilanteessa on määritetty palvelinrakenne integraatiokeskukselle

Sanomavälitys kahden sovellusadapteriosapuolen välillä tapahtuu seuraavasti:

1. Lähettävällä palvelimella sijaitseva sovellus (kuva 14, sovellus 1) muodostaa aineiston.

2. Palvelimelle asennettu sovellusadapteri (kuva 14, adapteri 1) tunnistaa muodostetun aineiston ja tälle määritettyjen konfiguraatioiden perusteella poimii muodostetun aineiston lähetyskäsittelyyn. Sovellusadapterin lähetyskäsittely asettaa aineistolle ohjelmiston määrittelemät JMS-parametrit. Parametreihin kuuluvat otsaketiedot sisältäen pakolliset JMS-parametrit, asetustason tason parametrit sekä itse sovelluksen muodostama aineisto JMS-sanoman kuormaksi. Sovellusadapteri muodostaa yhteyden integraatiokeskuksen jonomanageriin (kuva 14, jonomanageri 1) ja lähettää sanoman sovellusadapterin määritettyyn jonoon (kuva 14, jono 1) jonomanageripalvelimella.

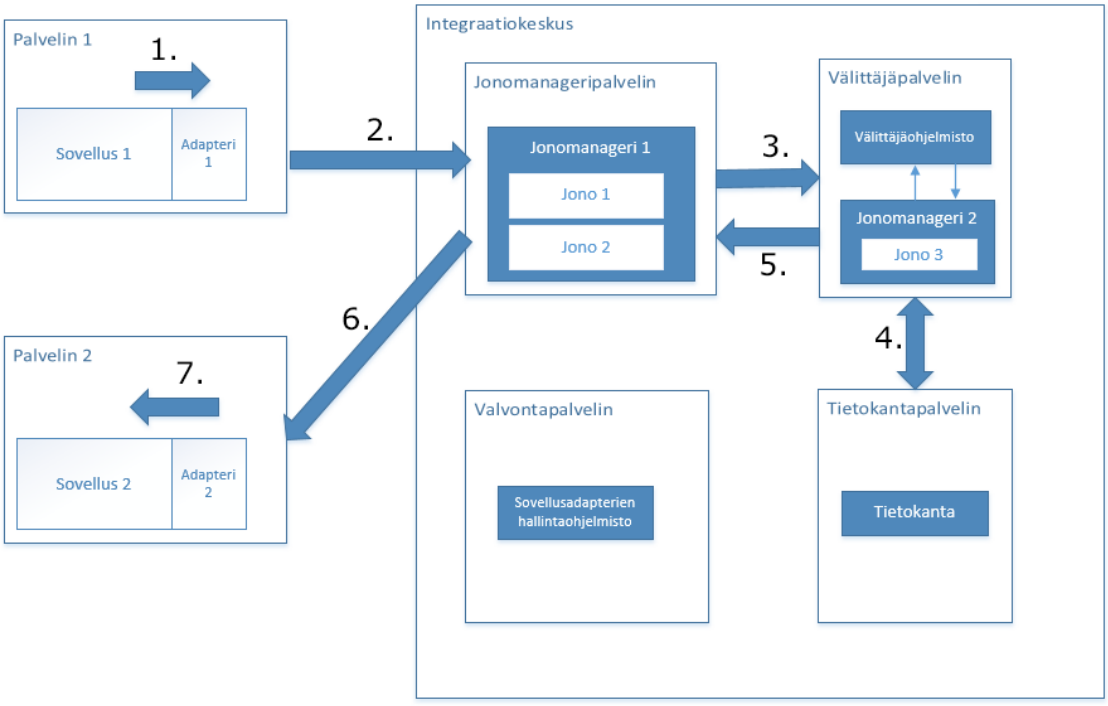
3. Jonomanageripalvelimen (kuva 14, jonomanageri 1) vastaanottojono lähettävälle adapterille, on viitteellinen jono välittäjäpalvelimella sijaitsevalle jonolle (kuva 14, jono 3). Sanoma ohjataan täten suoraan välittäjäpalvelimen alaiselle toiselle jonomanagerille ja kirjoitetaan tämän alaisuudessa olevaan paikalliseen jonoon (kuva 14, jono 3). Välittäjäohjelmisto (kuva 14, välittäjäohjelmisto) on konfiguroitu tarkkailemaan määritettyjä jonoja ja lukee sovellusadapterin lähettämän sanoman paikallisen jonomanagerin (kuva 14, jonomanageri 2) jonosta (kuva 14, jono 3).

4. Välittäjä hakee sanomalle ohjaustiedot perustuen JMS sanoman asetustietoihin tietokannasta (kuva 14, tietokanta) ja asettaa tälle lisäparametreja, kuten tiedon vastaanottajasta.

5. Välittäjä kirjoittaa sanoman jonoon, jota vastaanottavan palvelimen sovellusadapteri lukee (kuva 14, jono 2).

6. Vastaanottavan palvelimen (kuva 14, palvelin 2) sovellusadapteri (kuva 14, adapteri 2) tutkii aktiivisesti sille määritettyä vastaanottojonoa (kuva 14, jono 2). Kun sovellusadapteri havaitsee sanoman jonossa, se poimii sanoman ja tuo sanoman adapterin (kuva 14, adapteri 2) vastaanottokäsittelyyn.

7. Vastaanottavan palvelimen sovellusadapterin vastaanottokäsittelyn määrittelyn perusteella aineistolle toteutetaan standardit sekä kustomoidut vastaanottotoimenpiteet. Standardeihin toimenpiteisiin sisältyy esimerkiksi JMS-parametrien riisuminen sanomasta, mikä jättää vain kuorman lopulliseen sanomaan. Sanomalle voidaan vielä vastaanottovaiheessa suorittaa adapterin avulla toimenpiteitä, kuten salauksen purkaminen tai aineistomuunnos.



Kuva 14 - Sanomavälitys sovellusadaptereiden välillä

4.3 Valvonta

Sovellusadaptereiden valvonta perustuu useaan valvontatasoon ja näiden komponentteihin. Valvontatasoja ovat

- sovelluksen sisäinen valvonta
- paikallinen valvonta
- etävalvonta.

4.3.1 Sovellusadapterin sisäinen valvonta

Sovellusadapteri kirjoittaa toiminnastaan lokitietoa adapterin paikalliselle levylle (kuva 15). Tämä lokitieto pitää sisällään ohjelman toiminnallisuudessa määritetyistä tapahtumista kirjaa. Lokitapahtumat luokitellaan eri tasoisina, joista osa on täysin normaalitoimintaa kuvaavia ja osa virhetilanteita. Nämä tapahtumat voivat olla tasoltaan DEBUG, INFO, WARNING, ERROR, FATAL. Jokainen taso kuvaa tapahtuman luonnetta viitteellisesti. Lokitasot ovat osittain konfiguroitavissa ja osittain koodattuna ohjelmiston sisään.

- DEBUG: Debug-lokirivi on manuaalisesti lisätty lisätieto ohjelmistossa.
- INFO: Info tilaiset sanomat ovat sovellusadapterin normaaliin toimintaan liittyviä lokirivejä onnistuneista tehtävistä.
- WARNING: Warning-tila usein varoittaa määrittelemättömästä muuttujasta, tai muusta ei vaarallisesta, muttei myöskään optimaalisesta havainnosta.
- ERROR: Error-tilainen viesti on ilmoitus sovelluksessa tapahtuneesta virhetilanteesta. Tästä virhetilanteesta on usein mahdollisuus vielä palautua.
- FATAL: Fatal-virhetila on odottamattoman virheen aiheuttama. Nämä virheet ovat yleisimpiä selvitystyötä tuottavia virheitä, joita tässäkin sovellusadapterin kehitystyössä tullaan tarkastelemaan. Fatal-virheestä ei voida usein palautua automaattisesti, vaan tämän kumoaminen vaatii manuaalisia toimenpiteitä.

Lokitiedostoa voidaan käyttää yleisesti tarkastaessa adapterin sanomaliikennettä, tai mahdollisia virheitä ja näiden syitä. Lokitiedostoihin tulostuva teksti voidaan määrittää, joko ohjelmakoodissa asti tai adapterin konfiguroitavissa moduuleissa.

```
2016-10-30 22:04:48,684 INFO Data 28AUM2B000874 wrote to file /opt/Adapter/AdapterADAPTER/bin/ISCW_ALIVEPING
2016-10-30 22:34:47,669 INFO Create data from file:/opt/Adapter/AdapterADAPTER/resources/templates/jmsping modified: 2009-04-27 10:20:34 filesize: 4
2016-10-30 22:34:47,669 INFO Creating data from file jmsping
2016-10-30 22:34:47,675 INFO Created data with Id: 28AUMHB000875
2016-10-30 22:34:47,678 INFO Jms ping
2016-10-30 22:34:47,695 INFO Sending data 18AUMHB000876 to queue ADAPTER.ADAPTER1.TOHUB
2016-10-30 22:34:48,701 INFO Data 18AUMHB000876 sent OK
2016-10-30 22:34:48,707 INFO Created data 28AUMHC000877 from jms message.
2016-10-30 22:34:48,709 INFO Jms message  converted to data 28AUMHC000877
2016-10-30 22:34:48,710 INFO Session commited
2016-10-30 22:34:48,710 INFO Exiting commitrollback task
2016-10-30 22:34:48,710 INFO Siivotaan variablet
2016-10-30 22:34:48,720 INFO Sending data 28AUMHC000877 to queue ADAPTER.ADAPTER1.FROMADAPTER.ADMIN
2016-10-30 22:34:49,725 INFO Data 28AUMHC000877 sent OK
2016-10-30 22:34:49,727 INFO Data 28AUMHC000877 wrote to file /opt/Adapter/AdapterADAPTER/bin/ISCW_ALIVEPING
2016-10-30 22:46:25,378 INFO Create data from file:/opt/Adapter/AdapterADAPTER/resources/templates/jmsping modified: 2009-04-27 10:20:34 filesize: 4
2016-10-30 22:46:25,378 INFO Creating data from file jmsping
2016-10-30 22:46:25,384 INFO Created data with Id: 28AUMN6800878
2016-10-30 22:46:25,384 INFO Check spool
2016-10-30 22:46:25,384 INFO Transaction task begins
2016-10-30 22:46:25,386 INFO Create data from file:/opt/Adapter/AdapterADAPTER/temp/infofiles.txt modified: 2016-10-30 10:46:25 filesize: 70
2016-10-30 22:46:25,386 INFO Creating data from file infofiles.txt
2016-10-30 22:46:25,389 INFO Created data with Id: 28AUMN6800879
2016-10-30 22:46:25,390 INFO Exiting commitrollback task
2016-10-30 23:04:48,724 INFO Create data from file:/opt/Adapter/AdapterADAPTER/resources/templates/jmsping modified: 2009-04-27 10:20:34 filesize: 4
2016-10-30 23:04:48,724 INFO Creating data from file jmsping
2016-10-30 23:04:48,731 INFO Created data with Id: 28AUN2C00087A
2016-10-30 23:04:48,732 INFO Jms ping
2016-10-30 23:04:48,746 INFO Sending data 18AUN2C00087B to queue ADAPTER.ADAPTER1.TOHUB
2016-10-30 23:04:49,751 INFO Data 18AUN2C00087B sent OK
2016-10-30 23:04:49,756 INFO Created data 28AUN2C80087C from jms message.
2016-10-30 23:04:49,757 INFO Jms message  converted to data 28AUN2C80087C
2016-10-30 23:04:49,758 INFO Session commited
```

Kuva 15 - Sovellusadapterin loki

4.3.2 Paikallinen valvonta

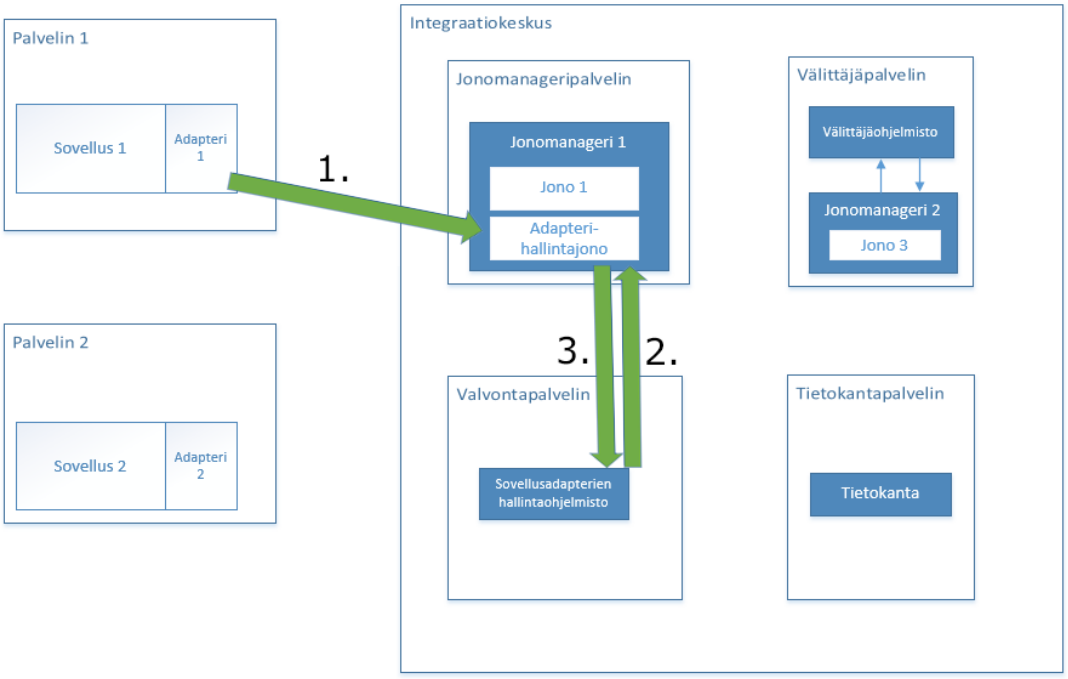
Vaikka sovellusadapterin sisäiseen toimintalogiikkaan on sisällytetty tiettyjen virhetilanteiden varalta olevaa valvontaa, miten sovellusadapteri selviää tilanteista, joita se ei itse pysty havaitsemaan? Tätä tarkoitusta varten sovellusadapterille on kehitetty erillisenä valvontaprosessina toimiva valvontatyökalu.

Valvontaprosessi valvoo sovellusadapterin tilaa ja tarvittaessa suorittaa adapterille komentoja. Valvontaprosessin toiminta perustuu määräajoin adapterin muodostamien tilaa kuvaavien tiedostojen tutkintaan. Osa sovellusadapterin tiloista, joihin valvontaprosessin täytyy reagoida, ovat virheen sijaan ennalta määritettyjä ominaisuuksia. Näihin sisältyvät esimerkiksi ajoitettu uudelleenkäynnistyminen tai sovellusadapterin versio- tai konfiguraatiopäivityksen jälkeinen uudelleenkäynnistäminen. Erikoistilanteen sattuessa sovellusadapteri saattaa kuitenkin jäädä ennalta määrittelemättömän syyn johdosta virhetilaan. Koska adapterin tilan valvonta perustuu tämän kirjoittamiin tilatiedostoihin, voidaan jumittuneelle adapterille suorittaa uudelleenkäynnistys vanhentuneen tilatiedoston perusteella.

4.3.3 Etävalvonta

Etävalvonta on yksi monipuolisimmista sovellusadapterin valvonnan osa-alueista. Sovellusadapteri sijaitsee usein palvelimilla, joille ei välttämättä ole pääsyä, joten sovellusadapterin etähallittavuus ja virhetilanteiden ennakoinnin työkalut ovat hyvin tärkeässä roolissa.

Sovellusadaptereiden toimintaa valvotaan integraatiokeskuksen osana olevan valvontapalvelimen avulla. Valvontapalvelimelle asennettu hallintaohjelmisto valvoo sovellusadaptereiden lähettämiä herätesanomia (kuva 16). Sovellusadaptereiden valvonta perustuu herätesanomiin, joiden tehtävä ei ole pelkästään ilmoittaa adapterin päällä olosta, mutta myös tuoda tietoa adapterin tarkemmasta tilasta. Herätesanoman saapumista tarkkaillaan myös ajallisesti, joten mikäli adapteri ei lähetä herätesanomaa tietyn ajan sisään, tästä nostetaan varoitus ja tämän jälkeen hälytys. Valvontapalvelin tarkkailee aktiivisesti jonomanagerille määritettyä hallintajonoa, jonne kaikki sovellusadapterit lähettävät herätesanomansa. Herätesanomat eritellään JMS-parametrien avulla osapuolikohtaisesti.



Kuva 16 – Herätteiden sanomavälitys hallintapalvelimen ja adapterien välillä

Sanomaliikennevalvonta

Sovellusadapterin lähettämää, kuten kaiken muunkin integraatiokeskuksen läpi kulkevaa liikennettä voidaan tarkkailla tietokannan tai käyttöliittymän avulla. Monitorointia varten integraatiokeskukselle on kehitetty erillinen käyttöliittymä visualisoimaan tietokantojen kautta tapahtuvaa toiminnallisuutta. Sanomat integraatiokeskuksen läpi kulkiessaan jättävät tietokantaan tiedon sanoman kulusta. Näitä tietoja voi käyttäjäystävällisemmin seurata tietokannan lisäksi myös graafisella käyttöliittymällä. Käyttöliittymään pääsevät käsiksi integraatiokeskuksen ylläpitävä henkilöstö. Käyttöliittymän avulla sanomien monitoroinnin lisäksi, voidaan suorittaa myös laaja-alaisesti ylläpito-operointia. Näihin kuuluvat mm. sanomien uudelleenlähetys, sanomien tilan vaihto, osapuolten tietojen päivitys sekä sovellusadaptereiden etäpäivitysten hallinta.

Jonovalvonta

Sovellusadapterin jonot ovat jonomanagerille määritettyjä objekteja, joita voidaan valvoa valvontapalvelimella sijaitsevan valvontaohjelmiston avulla. Valvontaohjelmisto toimii sovellusadaptereiden keskuksena valvoen hälyttäviä merkkejä jonon toiminnassa. Jonon ominaisuuksista voidaan tarkkailla esimerkiksi jonon syvyyttä tai vanhimman sanoman ikää jonossa. Nämä valvonnan hälytysrajat voidaan määrittää valvontapalvelimella tarpeelliseksi nähdylle tarkkuudelle, tai pitää oletuksena.

Levytilavalvonta

Vaikka adapterin lähettämät herätesanomat toimivat indikaattorina adapterin tilasta, myös adapterin tilannepäivitys tehdään näiden sisällön avulla. Tämän kuorman mukana adapteri lähettää myös tiedon käytettävissä olevasta levytilasta. Mikäli levytila uhkaa lähestyä alle sallitun rajan, lähettää adapteri tästä tiedon valvontapalvelimelle. Valvontapalvelin nostaa tästä hälytyksen, jotta valvontahenkilöstö huomaisi tilanteen ja reagoisi siihen tarvittaessa.

5 Nykytila ja ongelman kuvaus

Tarve sovellusadapterin valvonnan kehittämiselle on ilmennyt ajan myötä. Sovellusadapteri sisältää sille jo kehitysvaiheessa luodun valvontaympäristön ja sisäänrakennetun valvonta toiminnallisuuden, joka kuvattiin luvussa 4.3. Vaikka olemassa oleva valvonta kattaa useita virhetilanteita, kaikkiin mahdollisiin tilanteisiin ei ole valvonnan työkalut yltäneet. Jokainen tilanne kyllä havaitaan, mutta millaisessa ajassa? Mikäli virhetilanne osuu kriittisen järjestelmän tiedonsiirtoon, voi tästä olla haittaa suuremmassakin skaalassa ja virheen vaikutus näkyä jopa liiketoiminnassa asti.

Sovellusadapterin virheselvitys tuottaa manuaalista työtä, joka taas vaatii aina henkilöresursseja pois jostain muusta työstä. Ongelma sovellusadapterin valvonnan kehittämiskeinoista nousi esille tilanteessa, jolloin sovellusadapterin virhetilannetta ei havaittu, ennen kuin järjestelmien välinen tiedonsiirron puute huomattiin vastaanottavan tahon toimesta. Tämän johdosta tilanne nostettiin esille ja ongelmasta päätettiin tehdä laajempi selvitystyö.

Selvitystyölle määritettiin seuraavia tavoitteita:

- Ratkaisu parantaa valvonnan nykytilaa.
- Implementointi toimintaympäristöön on mahdollisimman yksinkertaista.
- Se on toteutettavissa resurssien puitteissa.

Kaikki ongelmat eivät kuitenkaan ole olleet näin vakavia kuin edellinen esitelty tapaus. Kuitenkin jokainen pienikin poikkeama normaalista tulisi havaita automatisoidusti. Usein korjaavia toimenpiteitä ei voida suorittaa aina automatisoidusti, mikä johtuu ongelmien laajasta kirjosta sekä valvontatyökalujen hajanaisuudesta. Automatisoitu valvonta auttaisi kuitenkin välttämään tilanteet, joissa virhettä ei ole edes pystytty havaitsemaan järkevässä aikaikkunassa. Ongelmalle lähdettiin etsimään hyvin avoimin mielin ratkaisuvaihtoehtoja.

6 Vaihtoehtojen kartoitus

Selvitystyö vaihtoehtojen kartoittamiselle aloitettiin palaverilla, jossa ongelmaa pohdittiin eri näkökulmista. Ongelman ratkaisulle ehdoteltiin monenlaisia vaihtoehtoja, joiden hyviä ja huonoja puolia sekä toteutuskelpoisuutta arvioitiin.

6.1 Vaihtoehtojen kartoitus

Vaihtoehtojen kartoitus oli monivaiheinen. Tätä kyseistä ongelmaa oli tutkittu jo ennen tätä selvitystä, mutta tuolloin mitään toimenpiteitä ei päätetty vielä tehdä. Ongelman toistuessa päätös toteuttaa valvonnan parantamisen selvitys lopulta tehtiin. Selvitystyö alkoi tutkimalla, mitä ratkaisuja aiemmissa selvitystilanteissa oli harkittu. Ensimmäiset suunnitelmat olivat kehitystyöpohjaisia ratkaisuja, mutta tämän tutkinnan myötä harkittiin myös tuotteistettuja ratkaisuja.

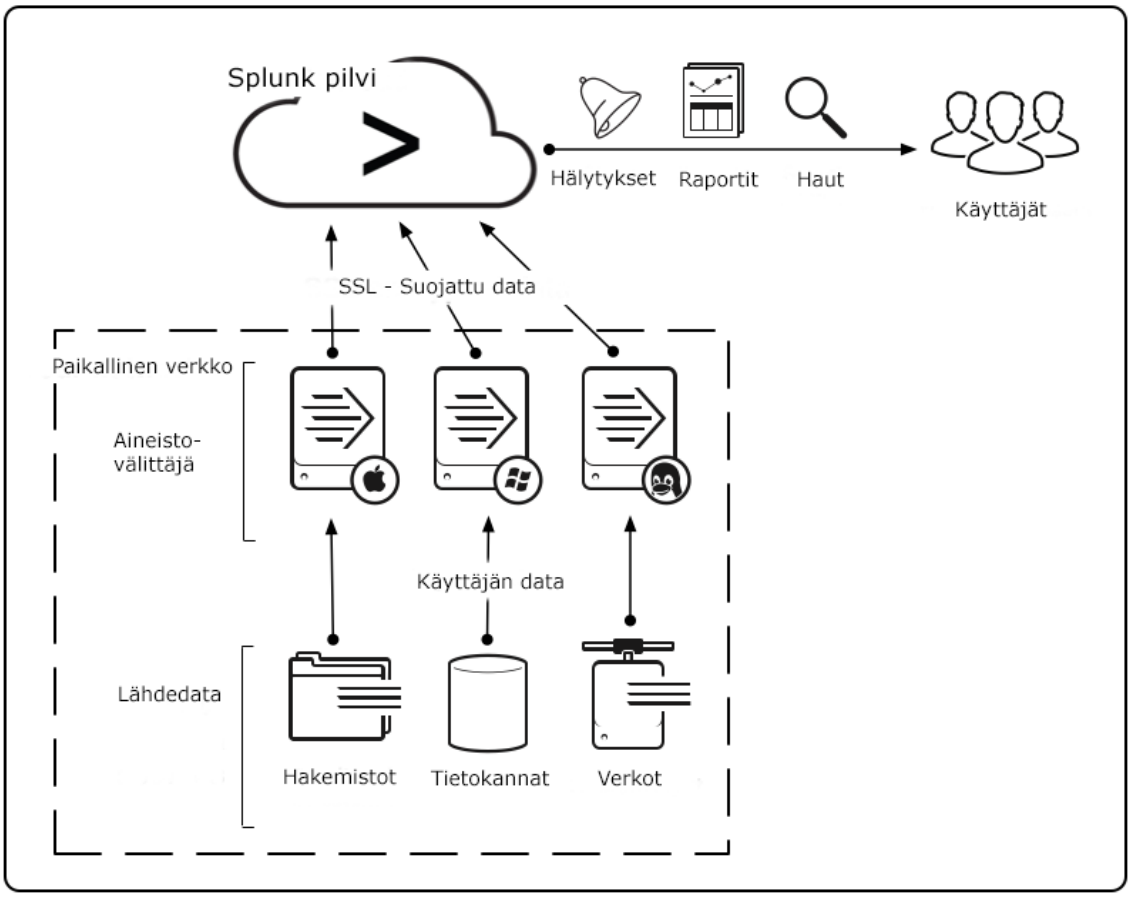
6.2 Tuotteistetut ratkaisut

Vaihtoehtoja kartoittaessa yksi vaihtoehdoista oli ulkoistetun osapuolen ratkaisun hyödyntäminen. Valvontaohjelmistoja on markkinoilla useita, joita monet yritykset onnistuneesti käyttävät. Näistä tuotteistetuista vaihtoehdoista valitsimme selvitykseen ohjelmistot Splunk sekä Logstash. Molemmat ohjelmistot olivat yritystasollamme käytettyjä muissa projekteissa, minkä vuoksi nämä nousivat ensimmäisiksi vaihtoehdoiksi esille.

6.2.1 Splunk

Splunk on samaa nimeä kantavan yrityksen ohjelmistotuote, jonka pääasiallinen käyttötarkoitus on datan analysointi, haku ja monitorointi. Splunk on ohjelmistotyökalu, jolla käyttäjä voi selkeyttää vaikealukuista tietokoneen generoimaan dataa selkeämmin hallinnoitavaksi. Splunk voidaan automatisoida tutkimaan tietyn palvelimen määritettyä resurssia ja raportoimaan tämän tuloksista määritetylle Splunk-palvelimelle, jonka tuloksia voidaan monitoroida web-käyttöliittymästä.

Kuten monet muutkin ohjelmistotarjoajat, myös Splunk tarjoaa ohjelmistoaan SaaS (Software as a service) -mallisena pilvipalveluna. Tässä selvityksessä päädyimme pilvialustan valintaan yksinkertaisen käyttöönoton vuoksi (kuva 17). Tuotteeksi valitsimme Splunk Lightin, koska tämä sisälsi tarpeelliset ominaisuudet testaukseen. Testiympäristönä datan poiminnalle käytettiin Windows-työasemaa.

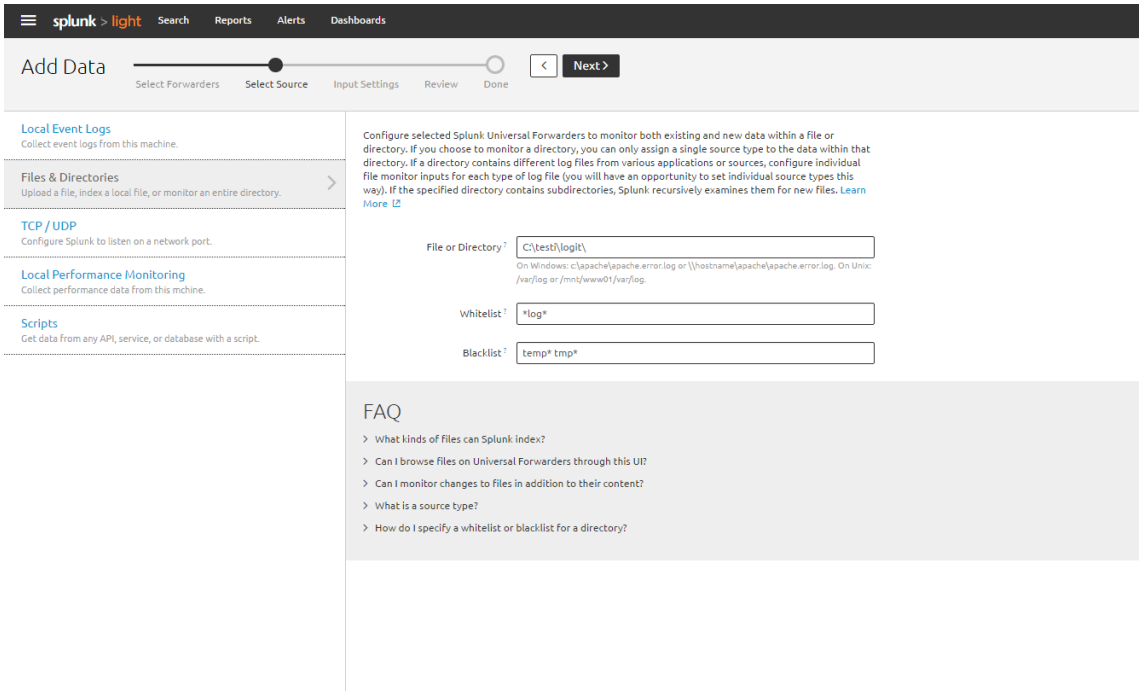


Kuva 17 – Splunk-pilviympäristö [16.]

Splunkin pilvipalvelun käyttöönotto sujui ohjelmiston sivuston tarjoamien asennusohjeiden avulla hyvin sujuvasti. Splunk-pilviympäristössä koostuu lähinnä pilvessä sijaitsevasta palvelusta, joka hoitaa kaiken datan prosessoinnin (indeksointi, haku ja analysointi).

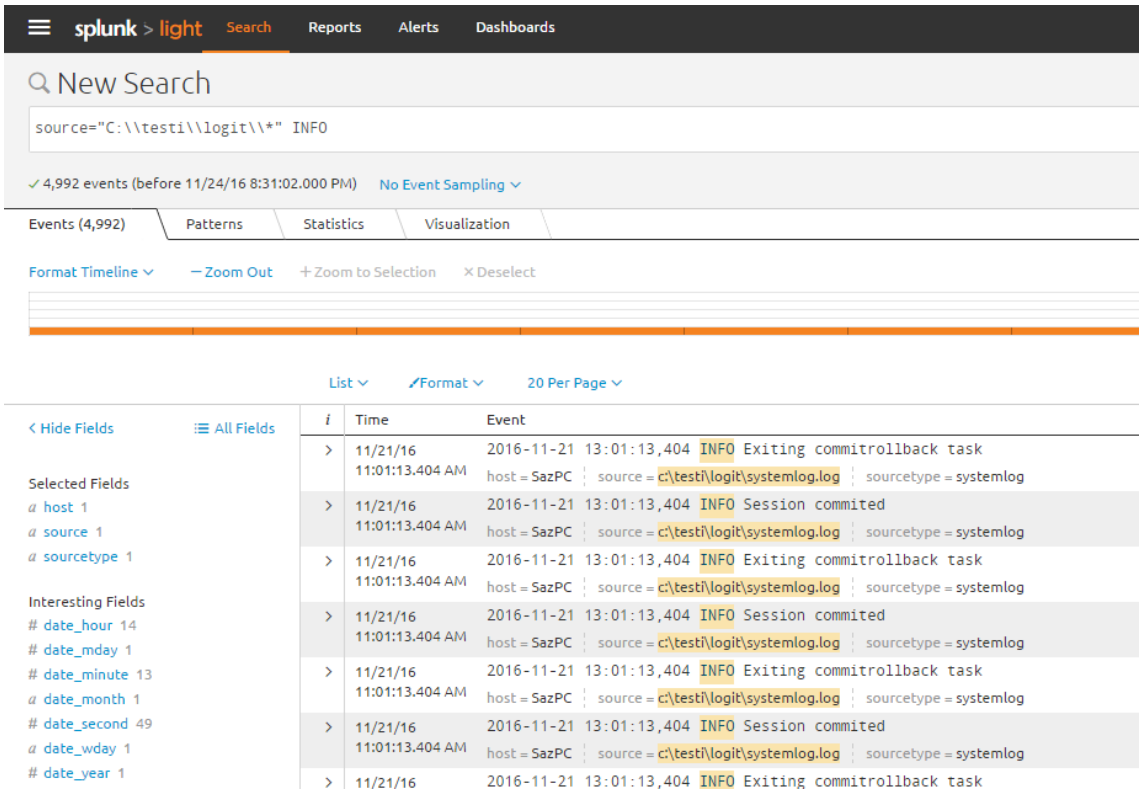
Datan siirtäminen pilveen tapahtuu Splunkin tarjoamalla aineistovälittäjällä (universal forwarder), joka asennetaan Windows-ympäristössä palveluksi. Aineistovälittäjän rooli on

hyvin samankaltainen kuin sovellusadapterin, eli toimia rajapintana ohjelmiston ja palvelimen välillä. Aineistovälittäjä asennuksen jälkeen autentikoidaan Splunkin pilven kanssa, jonka jälkeen yhteys pilven ja kohdekoneen välillä oli luotu. Aineistovälittäjän konfigurointi tapahtui Splunkin web-käyttöliittymän kautta. Tässä tilanteessa tutkittava resurssi haluttiin määrittää hakemistoksi, johon sovellusadapteri tuottaa järjestelmälokiaan (kuva 18).



Kuva 18 - Testihakemiston lisäys datan lukukohteeksi

Hakemistosta noudettua dataa voidaan hyödyntää hakuehdoin, kuten etsimällä aineistoja, joiden sisällöstä löytyy merkkijono ”INFO” (kuva 19). Tämä haku voidaan myös tallentaa hälytykseksi (Alert), jolla voidaan valvoa määriteltyjen ehtojen täyttymistä. Ehtoina voi olla esimerkiksi, että kyseisen kyselyn tulos palauttaa yli viisi osumaa. Valvonta toteuttaa tämän jälkeen määritetyt reaktiot, kuten esimerkiksi lähettää sähköposti-ilmoituksen virheestä. Tarkasteluvälin voi joko ajastaa haluamakseen tai määrittää aktiiviseksi valvonnaksi.



Kuva 19 – Splunk-haku

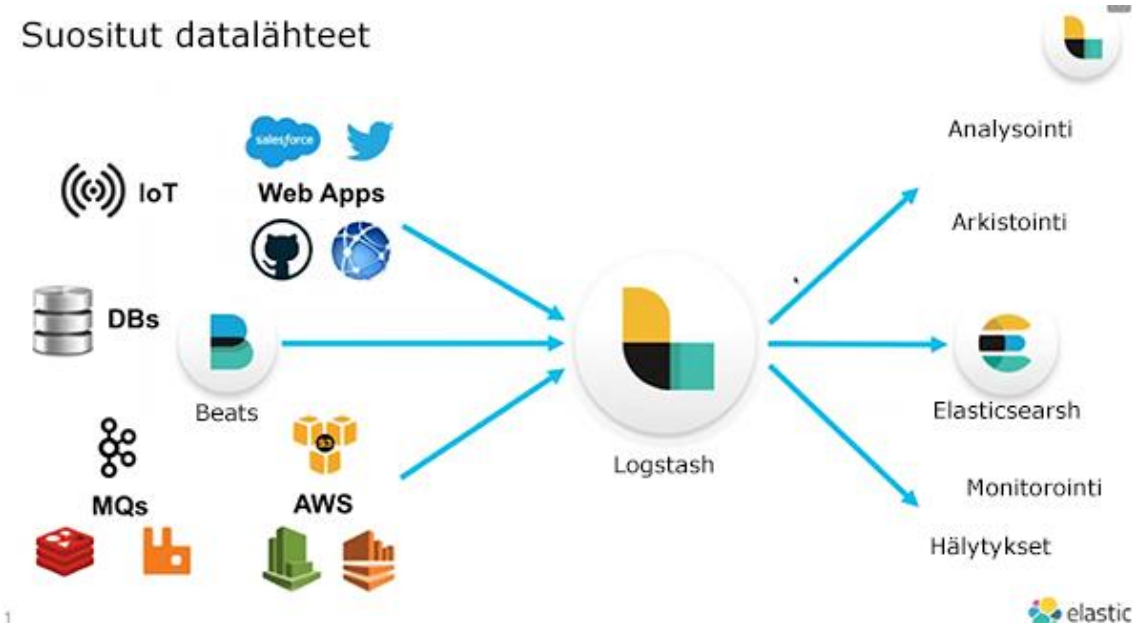
Splunk tuntui testauksen jälkeen hyvin monipuoliselta ja viimeistellyltä työkalulta. Käyttö ja optimaaliset hakuehdot olivat kuitenkin ensimmäistä kertaa käyttävälle henkilölle hieman hankalia muodostaa, mutta tuohonkin nähden ensikertalainen pääsi jo pitkälle.

Tässä tilanteessa ratkaisevaksi tekijäksi määräytyi kuitenkin ohjelmiston maksullisuus, sekä tämän käyttämien aineistovälittäjien asentamisen ja yhteyksien avaamisen tuoma työmäärä. Aineistovälittäjien asentaminen olisi vaatinut erillisiä lupia eri kohdepalvelimien hallinnoijilta edes asentaa kyseinen uusi prosessi palvelimelle. Myös palomuureille olisi jouduttu tekemään lukuisia porttikonfiguraatioita aineistovälittäjän ja Splunkin ohjelmiston välille. Splunk vaikutti kyllä varteenotettavalta kokonaisvaltaiselta valvontaratkaisulta, mikäli kyseessä olisi täysin uuden ympäristön pystytys. Splunk toimii myös optimaalisesti ympäristössä, jossa valvottavien palvelinten määrä ei ole niin suuri.

6.2.2 Logstash

Splunkin ollessa tietoisesti maksullinen vaihtoehto valmisohjelmistojen piirissä, lähestyimme seuraavaksi avoimen lähdekoodin Logstashia. Ohjelmisto oli etäisesti tuttu työväline, jota on käytetty toisessa yhteydessä väliohjelmistojen lokihallintaan.

Logstashin selvityksen kanssa alkoi heti sen roolin määrittäminen. Mihin ohjelmisto oikein kykenisi? Lisäselvityksen jälkeen ohjelmisto ei kuitenkaan tuntunut soveltuvan tarvitsemaamme tilanteeseen, mikä johtuu sen roolista olla enemmän lokien kerääjä ja uudelleenmuotoilija kuin näiden analysoija ja visualisoija. Ohjelmisto toimi enemmän paikallisena työvälineenä kuin keskitetystä pisteestä hallittavalta etäpalvelimilla sijaitsevien lokin läpikävijältä. Tämän lisäksi ohjelmisto toimi oikeastaan vain osakomponenttina siitä, mitä Splunk teki. Täyteen käyttökokemukseen olisi vaadittu useampi lisäohjelmisto, joita Logstashia hallinnoiva yhtiö Elastic myös tarjoaa (kuva 20). Tämän lisäkompleksisuuden vuoksi päätimme jättää Logstashin pois vaihtoehdoista.



Kuva 20 - Logstashin rooli elastic-ekosysteemissä [17.]

6.3 Kehitystyötä vaativat ratkaisut

Vaikka pyörää ei kannattaisi keksiä uudelleen, tuovat valmisratkaisut silti usein rajoitteita. Nämä rajoitteet olivat tämän työn tapauksessa vaikea mahduttaa työn määritysten alaisuuteen. Ideaalein ratkaisumalli olisikin kehitystyö, mutta mihin tämä voitaisiin kohdistaa. Kehitystyötä suunnitellessa tuli huomioida, ettei tämän muutoksen vienti ympäristöön aiheuttaisi riskejä nykyisen ympäristön toiminnalle. Tämän vuoksi kehitysratkaisut tulisivat olla itsenäisiä toiminnallisuudeltaan häiritsemättä nyky-ympäristön tilaa.

6.3.1 Adapterin jatkokehitys

Sovellusadapterin ollessa Javalla toteutettu ohjelmisto tämän jatkokehitys nousi luonnollisesti myös esille. Adapterin lähdekoodien ollessa saatavilla ohjelmistolle olisi mahdollista kehittää uutta toiminnallisuutta ohjelmistokehityksen kautta.

Adapterin sovelluskehityksessä nousi kuitenkin pinnalle useita haasteita. Mikäli ohjelmistoa päädyttäisiin jatkokehittämään, tarkoittaisi tämä käyttöönotossa sovelluksen uudelleenasentamista jokaiselle muutoksen alaiselle adapteriosapuolelle. Kyseisessä sovellusadapterin toimintaympäristössä adapteriasennuksia on useita satoja. Uudelleenasennusprosessi olisi tässä tilanteessa suhteellisen työläs ja aikaa vievä operaatio, mistä johtuu palvelinten hajautetusta omistajuudesta sekä näille pääsystä.

Olennainen ongelma adapterin sisäänrakennetun virhekäsittelyn parantamisessa oli myös se, kuinka adapteri pystyy vikatilanteessa havainnoimaan ongelman, mikäli toiminta on häiriintynyt jostain adapterin toimintaa vahingoittavasta virheestä. Näitä voisivat olla esimerkiksi adapterin muistin loppuminen tai adapterin sammuminen ilman valvontaprosessin päällä oloa. Näiden syiden johdosta selvityksessä päätettiin etääntyä adapterin valvonnassa adapterin ulkopuolelle.

6.3.2 Valvontaprosessin jatkokehitys

Adapterin jatkokehityksestä poiketen valvontaprosessi olisi adapterista ulkoinen ratkaisu, jolloin tähän ei kohdistuisi samoja ongelmatilanteita kuin adapterille. Valvontaprosessin kehityksessä esiintyi kuitenkin omat haasteensa. Ensimmäisenä seikkana valvon-

taprosessia tutkiessa esiintyi tämän asennusympäristökohtaiset eroavaisuudet. Riippuen oliko sovellusadapteri asennettu Windows- tai Unix-pohjaiseen järjestelmään, oli sovellusadapterien tekninen toteutus hyvin poikkeava toisistaan. Windows-ympäristössä valvontaprosessi toimi Windows-palveluna, kuten sovellusadapteri. Unix-ympäristössä kuitenkin valvontaprosessi oli toteutettu ajastetulla skriptillä. Tämä vaikuttaisi jo siihen, että ympäristökohtaisesti ratkaisu jouduttaisiin toteuttaa samalla logiikalla kahtena erillisenä teknisenä toteutuksena.

Valvontaprosessin jatkokehityksellä olisi myös sama haaste kuin adapterin jatkokehityksellä käyttöönottovaiheessa. Käyttöönotossa Windows-ympäristöissä valvontaprosessi tulisi uudelleenasentaa. Tämä toisi jälleen hajanaisessa ympäristössä eri tahojen hallinnoimien palvelimien sovelluspäivityksistä haasteellisen. Lisäksi nämä muutokset valvontaprosessiin tai adapteriin eivät täyttäisi määrittelyn ehtoa, ettei ympäristön nykytilaan koskettaisi.

6.3.3 Status-sanomiin pohjautuva valvontajärjestelmä

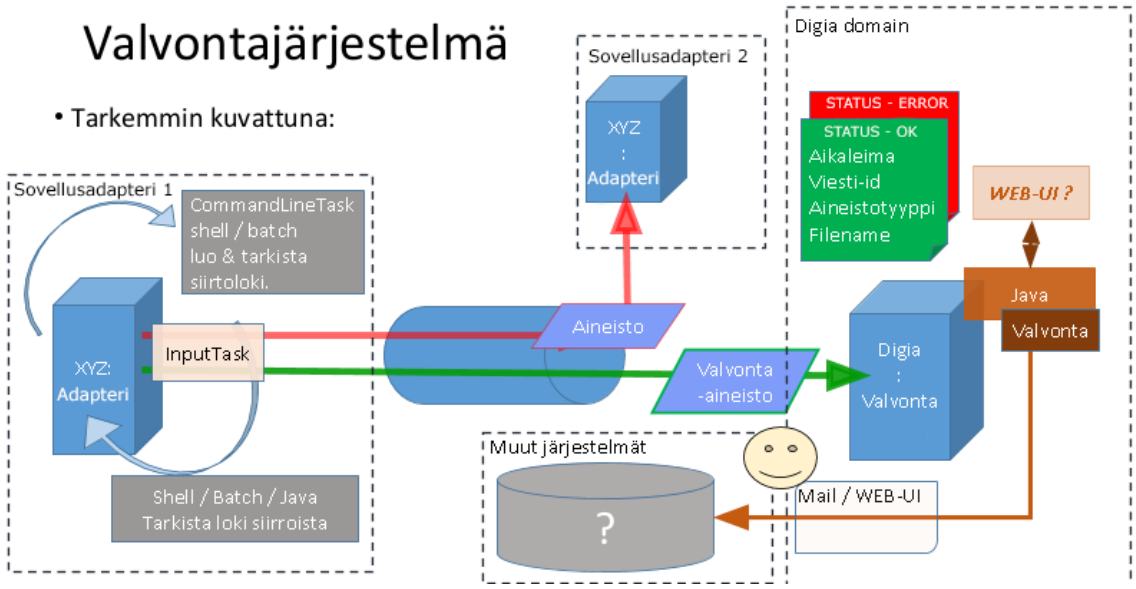
Siirrettäessä painopistettä ulos jo toteutettujen ohjelmistojen muokkauksesta pyrittiin ratkaisu löytämään rakentamalla valvontaratkaisu näiden tuotteiden ulkopuolelle. Valvonnasta alettiin suunnitella mallia, jossa sovellusadapterin palvelinympäristö toimisi yhdessä integraatiokeskuksen kanssa valvonnan kokonaisympäristönä. Tämän tyyppistä ratkaisua oli ehditty pohtia jo ennen tämän insinöörityön aloittamista ja tästä ratkaisusta oli ehditty tehdä esitys, jota ei koskaan lähdetty kuitenkaan toteuttamaan. Valvontaratkaisuehdotukseksi esitettiin seuraavaa toteutusta (kuva 21).

- Valvonta tapahtuisi sovellusadapterin palvelimelle luodulla skriptillä, joka tarkastelisi adapterin lokia ja havaitsisi tästä virheitä.

- Virheistä raportoitaisiin määritetyin aikavälein statusviestien avulla integraatiokeskuksen hallintapalvelimella sijaitsevalle valvontaohjelmistolle.

- Integraatiokeskuksen puolelle kehitettäisiin Javaan pohjautuva työkalu, jolla olisi oma web-käyttöliittymä adaptereiden tilan esittämiseen.

- Lisäksi mahdollisista häiriöistä tehdään käyttöliittymän toimesta virhetilanteissa selvityspyyntö sähköpostilla.



Kuva 21 – Adapteri-hallintapalvelinpohjainen valvontajärjestelmä

Kehityssuunta idealle alkoi tässä vaiheessa tuntumaan oikealta. Ideaa alettiin tarkemmin läpikäymään ja pohtimaan sen eri puolia. Ratkaisumallia tarkastellessa otettiin esille lokikäsittelyn sijainti ja statusviestit sekä web-käyttöliittymä. Mikäli lokien käsittelyskripti sijaitsisi sovellusadapterin palvelimella, asennus jouduttaisiin tekemään useaan eri kohteeseen. Sovellusadapterin palvelinympäristö on myös hyvin vaihteleva, jonka takia skripti pitäisi joko tehdä käyttöjärjestelmäriippumattomaksi tai kehittää jokaiseen ympäristöön oma käsittelynsä. Myös skriptin vienti palvelimelle ja ajastus tulisi hoitaa palvelinkohtaisesti.

Lisäksi skriptin tulisi olla käsittelyltään hyvin kevyt, jotta tämä ei kuormittaisi ulkopuolisia palvelimia, jotka eivät ole integraatiokeskuksen valvontahenkilöstön hallinnassa. Skriptin suunniteltiin lähettävän tilaviestejä tarkastuksen ohella, joilla indikoitaisiin adapterin tarkastuksen hetkistä tilaa. Tämä tarkoittaisi valvontapalvelimelle luotua vastaanottoa ja jatkokäsittelyä näille valvontasanomille. Tähän tarkoitukseen oli suunnitteilla kehittää Javalla sovellus, jolla olisi käyttöliittymä integraatiokeskuksen päässä. Sovellus, joka käsittelisi sanomat, toimisi yhteistyössä web-käyttöliittymän kanssa.

Kuitenkin tarve erilliselle käyttöliittymälle jäi hieman kyseenalaiseksi. Mikäli sähköpostipohjaisia hälytyksiä käytettäisiin, mikä olisi käyttöliittymän rooli, jos virheestä on tehty jo ilmoitus? Tuolloin käyttöliittymän tarkkailukin vaatisi aktiivista valvontaa tai tämän rooli

jäisi olemattomaksi. Kehitysidean kritiikistä huolimatta tämä toteutusmalli alkoi antaa suuntaa uudelle idealle.

6.3.4 Lokitarkasteluun pohjautuva valvonta

Perustuen adapterin ja hallintapalvelimen väliseen yhteistoiminnalliseen valvontajärjestelmään syntyi jatkokehitysidea käsittelylle. Lokia tarkasteleva skripti olisi edelleen olennainen osa käsittelyä, mutta tämä siirrettäisiin ulkoisilta palvelimilta integraatiokeskuksen päähän. Näin käsittely saataisiin keskitettyä yhteen ympäristöön, jolloin ratkaisua ei tarvitsisi kehittää usealle käyttöjärjestelmälle. Myös valvontapalvelimelle pääsy on integraatiokeskuksen ylläpitohenkilöstöllä mahdollista, jolloin skriptin ajoympäristö on täysin hallittavissa. Skriptin kuormituskin saadaan näin siirrettyä pois ulkoisilta palvelimilta. Lokit, jotka sijaitsevat ulkoisilla palvelimilla, noudettaisiin palvelimelta hyödyntäen JMS-sanomavälitystä, jota sovellusadapterit käyttävät normaalissa sanomavälityksessä. Virhetilanteeksi tulkittaisiin joko lokilta löydetyt virheet tai lokin noutamisvirhe. Näin huomioitaisiin laaja-alaisesti virhetilanteita palvelimen yhteysongelmista sovellusadapterin lokin virhetilanteisiin. Valitun ratkaisun teknistä osuutta aloitettiin konkreettisesti toteuttamaan loppukesästä 2016.

7 Toteutus

Valituksi ratkaisuksi päätyi sovellusadapterin lokeihin pohjautuva käsittely valvontapalvelimella. Ratkaisussa hyödynnettäisiin integraatiokeskuksen ympäristöä ja ohjelmistoja sekä luotaisiin lokia tarkastelevia bash-skriptejä. Tässä luvussa käsitellään idean toteutusta prototyypiksi testiympäristöön, testausprosessia ja tämän tuloksia.

7.1 Suunnittelu

Visio lokitarkastelusta oli jo alkuvaiheessa korkealla tasolla selkeä, mutta yksityiskohdat eivät olleet ratkaisulle vielä valittu. Selvää kuitenkin oli, että toteutus olisi pystytty toteuttamaan usealla eri menetelmällä. Suunnittelu lähtötilanteessa oli seuraava.

- Skripti ajastetaan ajettavaksi kerran päivässä per määritelty osapuoli.

- Skripti ottaa parametrinaan osapuolen nimen, joka halutaan tutkia. Kutsuessa skripti aktivoi noudon X -sovellusadapterin lokeille ja sovellusadapteri palauttaa ne integraatiokeskukselle.

- Lokin saavuttua skripti jatkaa ottamalla lokin käsittelyyn ja käymällä tämän läpi.

- Läpikäynnin jälkeen hälytystiedostoon lisätään rivi lokin läpikäynnistä riippumatta oliko virheitä tai ei.

- Kaikki adapterit kirjoittavat tuloksensa samaan raporttiin ja raportti lähetetään kerran päivässä valvovalle henkilöstölle tukipalveluun.

Suunnitteluvaiheessa skriptiä aloitettiin jo testimielessä kehittämään, jotta ideoita saatiin testattua. Rakentaminen skriptille alkoi lokikäsittelystä. Skripti ensin poimi lokin määritetystä sijainnista, jonne testimielessä tämä oli manuaalisesti tuotu. Skripti otti aineiston käsittelyyn, kävi tämän läpi ja teki merkinnät raporttiin. Lokin käsittelyn jälkeen loki poistettiin, mutta raportti säilytettiin. Samaan raporttiin kirjoitettiin kaikkien osapuolten raportit, oli virheitä tai ei. Tarkasteluiden lopuksi suunnitteilla oli ajastaa erillinen sähköpostilähetys raportille, jonka jälkeen raportti poistettaisiin.

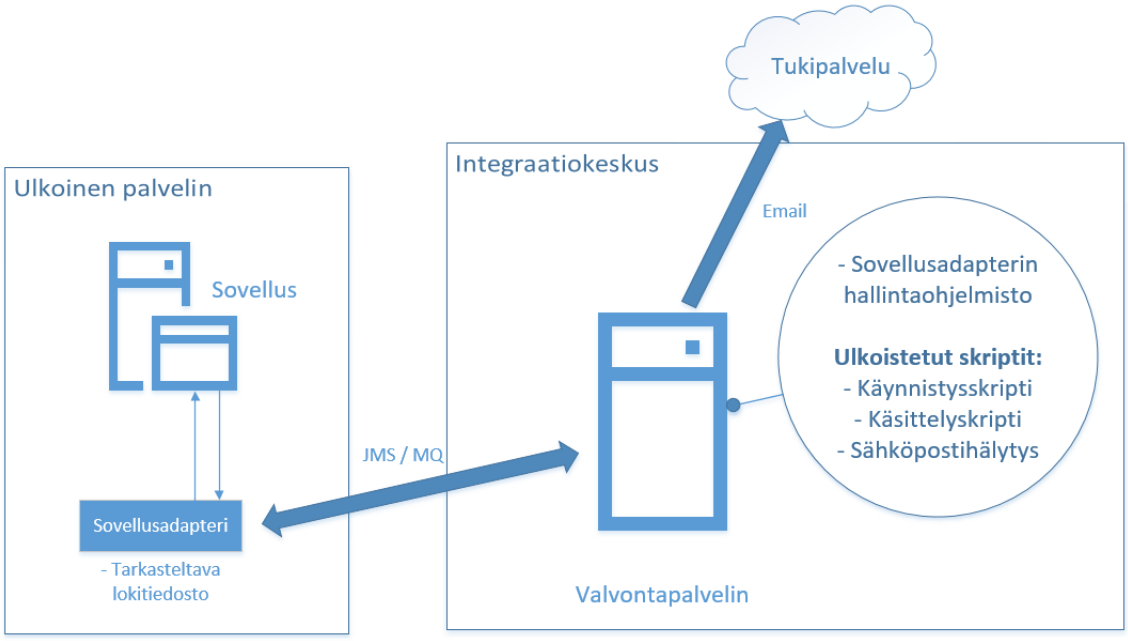
Käsittelyn muokkaus kävi läpi monta iteraatiota, jossa arvioitiin käsittelyn eri aspekteja, kuten tulisiko virheet listata eriteltyinä ja ennalta määriteltyinä vai raportoitaisiinko kaikki virheet. Skriptin toimintoja jaettiin useaan osaan näiden toimien vuoksi, ja lopullinen jako pilkkoi käsittelyn kolmeen erilliseen skriptiin. Käsittelylle päätettiin tehdä käynnistysskripti, jotta käsittelyllä olisi eräänlainen kontrolleri, joka hallitsisi lokikäsittelyn käynnistämistä. Lisäksi hälytysraportista tehtiin osapuoli- ja tarkastuskohtainen, sekä se lähetettiin tarkastuksen läpikäynnin jälkeen vain virhetilanteissa.

Sähköpostilähetys poimittiin olemassa olevasta Pythonilla kirjoitetusta toteutuksesa, jota käytettiin toisessa yhteydessä valvontapalvelimella geneerisesti sähköpostien lähetykseen. Python skriptiä kutsumalla skriptille annettiin parametreina lähettäjä, vastaanottaja, otsikko sekä itse sähköpostin viestisisältö.

Suurimpana haasteena käsittelyn suunnittelussa oli lokien nouto etäpalvelimelta integraatiokeskukselle ja näiden paikallistaminen levyltä. Sovellusadaptereiden lokien nou-

dolle oli jo olemassa toteutus, jolla saatiin valvontatyökalulla kutsuttua adapteria palauttamaan lokitiedostot. Tätä ominaisuutta käytetään integraatiokeskuksen päivittäisessä ylläpitotyössä manuaalisesti. Lokitiedostot palautettiin kuitenkin usealta päivältä ja pakattuna zip-formaattiin. Lokitarkastelun käsittelyn olisi tässä tilanteessa tarvinnut purkaa tuo paketti ja valita sieltä oikean päivän lokitiedosto. Tässä tilanteessa yksinkertaisemmalta tuntui kehittää muokattu lokikäsittely, joka asennettaisiin etäpäivityksenä adapterille ja liitettäisi osaksi adapterin vastaanottomoduulia, jossa nykyinenkin lokikäsittely sijaitsee.

Lopullisena suunnitelmana käsittelyn kokonaiskuva ja toimintojen jaottelu saatiin kasaan (kuva 22).



Kuva 22 – Ylätason suunnitelma lokikäsittelystä

7.2 Ympäristöön kohdistuneet toimenpiteet

Määrittelyiden jälkeen tiedossa oli tarvittavat lisäykset ja muutokset ympäristöön käsittelyn mahdollistamiseksi. Selkein toimenpiteistä oli käsittelyskriptien tuominen valvontapalvelimelle. Valvontapalvelimelle tehtiin sopivaan sijaintiin hakemisto, johon tuotiin loki-

käsittelyn skriptit sekä tehtiin käsittelyn väliaikaishakemistot aineistoille. Skriptejä testivaiheessa ajettiin suljetussa Unix-pohjaisessa testiympäristössä, mutta kun testaus eteni tarpeeksi pitkälle, tuotiin skriptit testattavaksi lopulliseen testiympäristöön.

Lokihaku suunniteltiin toteutettavaksi sovellusadapterin moduuliin tehtävällä lisäyksellä, jonka logiikka palauttaisi halutun lokin oikeassa formaatissa. Sovellusadaptereiden konfiguraatiot kyseisellä moduulilla ovat yksilöllisiä, joten tämä tarkoitti, että tuo moduulin osuus jouduttiin osapuolikohtaisesti viedä etäpäivityksellä palvelimelle. Operaatio todettiin kuitenkin riittävän yksinkertaiseksi, koska päivitys ei vaatinut palvelimelle pääsyä, vaan toteutettiin etäpäivityksenä. Ajallisesti yhden adapterin päivitys kesti testatessa noin 5-15 minuuttia. Päivityksen lisäksi kyseisen adapterin tunnus lisättiin asetustiedostoon, jota käynnistysskripti luki ja käytti käsittelyn käynnistämisen yhteydessä.

Lokipyynnön lähetys toteutettiin hyödyntämällä integraatiokeskuksen valvontapalvelimella sijaitsevan sovellusadaptereiden hallintaohjelmiston ominaisuuksia. Ohjelmistolle pystyttiin sovellusadapterin tavoin määrittämään hakemistorajapinta, josta tämä poimisi aineistoja ja lähettäisi näitä konfiguraatioiden mukaisesti JMS-sanomina kohteille. Tässä tilanteessa määritimme uuden hakemistorajapinnan, johon skripti kirjoittaisi lokipyyntötiedoston. Tiedostonnimi kyseiselle tiedostolle sisältää osapuolen nimen, jonka loki halutaan hakea ja tämä poimitaan regex-käsittelyllä ja parametrisoidaan JMS-vastaanottaja parametriksi. Lisäksi käsittely lisää JMS-parameterina tiedon operaatiosta, jolla vastaanottavan adapterin moduuli tunnistaa käytettävän moduulin operaation. Tässä tapauksessa halusimme adapterin käyttävän moduuliin lisättyä käsittelyä, joka palautti yksittäisen lokitiedoston tekstiformaatissa.

Yhteenvetona toimenpiteet toiminnallisuuden pystytyksessä sekä adaptereiden lisäämisestä valvonnan piiriin:

Toimenpiteet valvonnan pystytyksestä (tehdään vain kerran per ympäristö):

- Luo tarvittavat hakemistot skripteille ja tuo nämä palvelimelle.

- Luo hakemistorajapinta sovellusadapterin hallintaohjelmistolla ja konfiguroi tämä.

- Säädä hakemistojen ja skriptien oikeudet palvelimella sopivalle tasolle.

Toimenpiteet adapterin lisäyksestä (tehdään jokaiselle lisättävälle adapterille):

- Lähetä adapterille etäpäivityksenä moduulilisäys yksittäisen lokin palautukselle.

- Lisää kyseisen adapterin nimi asetustiedostoon.

7.3 Lokikäsittely

Tässä luvussa kuvataan lokikäsittelyn eri vaiheilta. Lokikäsittelyn vaiheet esitetään myös prosessikaaviona liitteessä (1).

1. Käsittely aktivoidaan crontab:iin ajastetulla skriptillä ”startlogwatch”, joka valitsee osapuolen asetustiedostosta ja kutsuu käsittelyskriptiä välittäen parametrina tutkittavan osapuolen tunnuksen. Asetustiedosto käydään läpi rivi riviltä pitäen 15 sekunnin tauko jokaisen rivin välillä.

2. Käsittelyskripti ”logwatcher” alkaa tuottamalla hallintaohjelmistolle tiedoston, jonka tiedostonnimestä ilmenee osapuoli, jonka loki halutaan palauttaa. Aineisto kirjoitetaan hakemistorajapinnan lukukohteeseen. Tämän jälkeen käsittelyskripti asetetaan nukkumaan minuutin ajaksi, ja odottamaan lokin saapumista.

3. Sovellusadaptereiden hallintaohjelmiston hakemistorajapinta poimii tiedoston käsittelyyn ja asettaa sille JMS-parametrit. Parametreiksi asetetaan tieto lähettäjästä, aineistotyypistä, vastaanottajasta ja operaatiosta. Hallintaohjelmisto lähettää aineiston sovellusadapterille JMS-sanomana, ja kirjoittaa tämän adapterille kohdistettuun saapuvien sanomien jonoon.

4. Sovellusadapteri lukee sanoman jonosta ja ottaa sen vastaanottokäsittelyyn. Vastaanottokäsittelyssä sanoma tunnistetaan lisätyssä moduulin osuudessa yksittäisen lokin noutosanomaksi.

5. Sovellusadapteri poimii pyydetyn lokin, asettaa tälle välitykseen tarvittavat JMS-parametrit lähettäjä, aineistotyyppi ja vastaanottaja, sekä lähettää tämän takaisin

integraatiokeskukselle. Integraatiokeskuksella sanoma päätyy arkistoon, joka sijaitsee valvontapalvelimella.

6. Käsittelyskripti herää minuutin jälkeen jälleen aktiiviseksi ja ottaa yhteyden tietokantaan. Tietokannasta suoritetaan kysely, jolla haetaan viimeisin kyseiseltä osapuolelta saapunut lokin ID. ID:tä käytetään palautetun lokin tiedostonnimenä arkistossa.

7. Mikäli käsittely löytää lokin arkistosta tietokannan palauttamalla ID:llä, tekee skriptin lokista kopion työhakemistoon. Mikäli lokia ei löydy, tämä tulkitaan virhetilanteeksi, sähköpostihälytys lähetetään ja skripti lopettaa käsittelyn.

8. Skripti aloittaa läpikäymään lokia rivi kerrallaan, etsien määritettyjä virhettä indikoivia merkkijonoja. Virheen löytyessä skripti kasvattaa kyseisen virheen laskuriarvoa yhdellä.

9. Kun loki on läpikäyty, skripti etenee raportin muodostamiseen. Tässä vaiheessa tarkastetaan, onko raportoituja ongelmia esiintynyt. Mikäli virheitä on löytynyt ja virhelaskureiden jokin arvo on yli 0, tekee käsittely rivin virheestä ja tämän esiintymismäärästä lokissa. Tämän lisäksi skripti lisää lokin rivimäärän lisätiedoksi raporttiin. Mikäli virheitä ei löydy, käsittely päättyy, eikä hälytystä lähetetä.

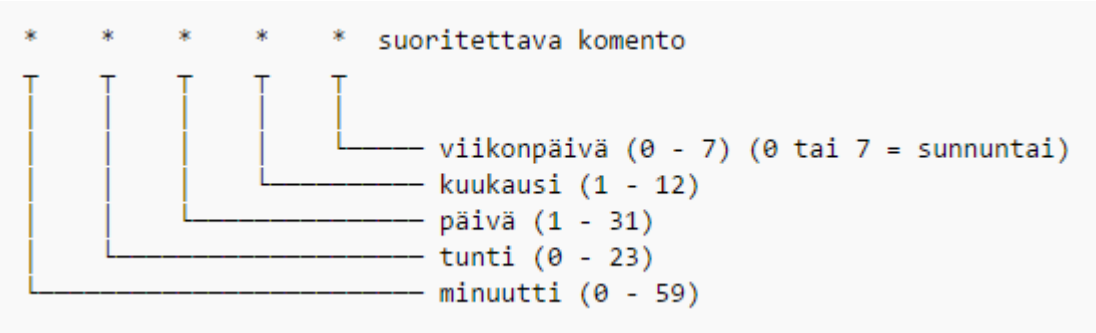
10. Kun raportti on kirjoitettu, kutsuu skripti sähköpostin lähetysskriptiä ”Sendemail” ja välittää raportin parametreineen eteenpäin. Käsittelyskripti poistaa raportin ja lokitiedoston käsittelyhakemistosta. Käsittelyskriptin toiminta päättyy.

11. Sähköpostin lähettävä skripti käynnistyy ja saa parametreina lähettäjän ja vastaanottajan sähköpostiosoitteet, aiheen, sekä raportin sähköpostin viestiksi. Sähköposti lähetetään, ja käsittely päättyy.

Automatisoidun toiminnan jälkeen lopputuloksena on virhetilanteesta aiheutunut sähköpostihälytys tukilaatikossa, jota seuraa integraatiokeskuksen valvontahenkilöstö. Valvontahenkilöstö tarkastaa hälytyksen aiheuttaneen tilanteen ja arvioi, vaatiiko se toimenpiteitä.

7.4 Automatisointi

Käsittelyä määrittäessä tärkeä osa käsittelyä oli automaattisuus. Valvonnasta haluttiin tehdä itsenäinen automaattisesti itseään toistava prosessi. Skriptien automatisointi toteutettiin ajastamalla skriptit cronin avulla (kuva 23). Cron on Unix-pohjaisissa käyttöjärjestelmissä toimiva työkalu, jolla voidaan ajastaa toimintoja. [18.] Automatisoinnin ohessa, cronin komentoon sisällytettiin tulostus skriptin toiminnalle erilliseen lokitiedostoon Unix-komennolla ”tee”.



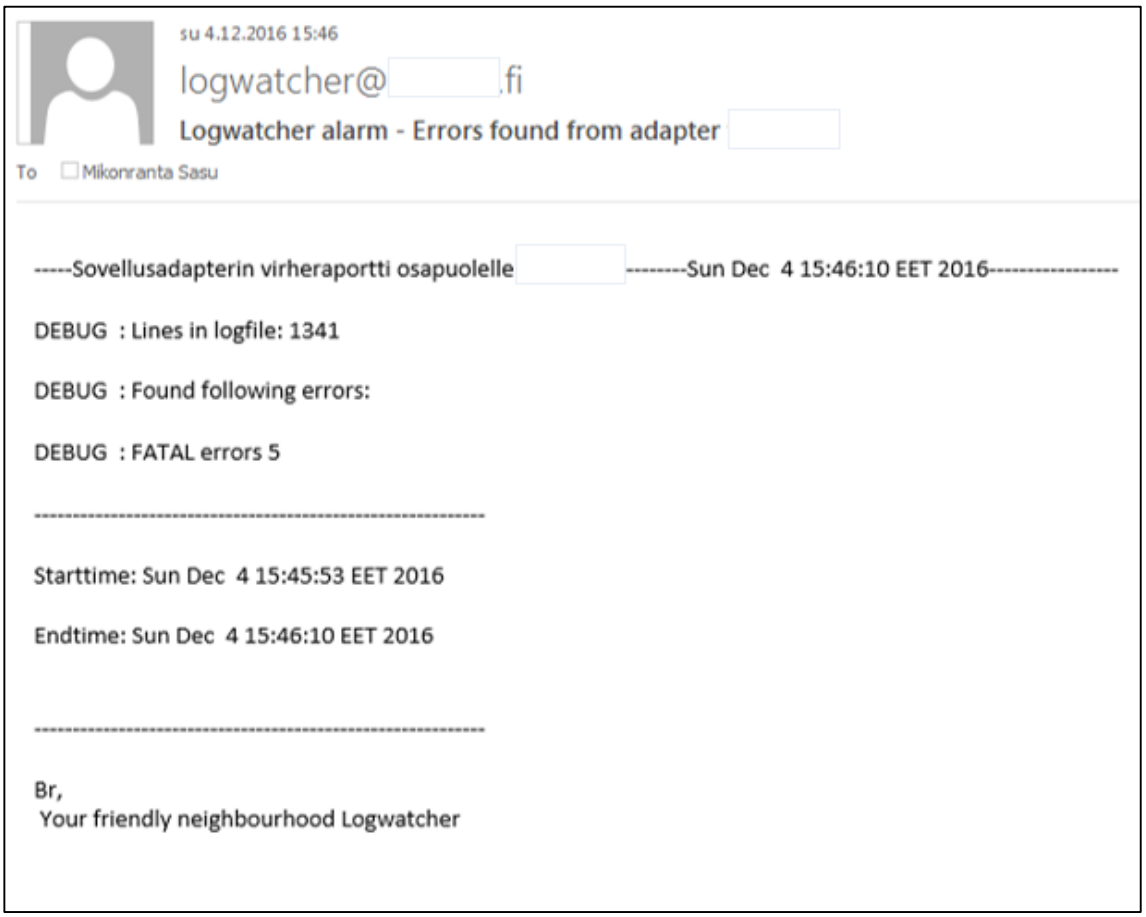
Kuva 23 – Cron-konfiguraatio [18.]

7.5 Testaus

Työtä pilotoitiin testiympäristössä kolmen sovellusadapterin avulla. Sovellusadapterit olivat valittu niiden toimintojen perusteella sekä häiriöherkkyyden takia. Häiriöherkkyys syntyi tavallista monimutkaisemmasta toiminnasta, kuten salauksen muodostamisesta ja purusta, tai aineistomuunnoksien suorittamisesta. Yksi osapuolista oli taas hieman vähemmän häiriötä tuottava, joka otettiin mukaan tarkastamaan, ettei tämä tuottaisi hälytyksiä.

Skriptissä oli eriteltynä yleisimpiä salausvirheitä eriteltyinä, mutta viimeisenä ehtona geneerinen FATAL-virheiden havainnointi, mikäli joku virheistä ei olisi ennalta määritetty. Fatal-virheet täten voisivat laskurissa olla muiden virheiden summa, mikäli nämä ovat jo määriteltynä.

Testivaiheessa skriptiä ajettiin manuaalisesti ja tukilaatikon sijaan sähköpostit ohjattiin omaan sähköpostilaatikkooni. Mikäli ajon aikana lokitiedostossa havaittiin virhetilanteita, näistä raportoitiin sähköpostilla (kuva 24).


su 4.12.2016 15:46

logwatcher@ .fi

Logwatcher alarm - Errors found from adapter

To Mikonranta Sasu

-----Sovellusadapterin virheraportti osapuolelle --------Sun Dec 4 15:46:10 EET 2016----------

DEBUG : Lines in logfile: 1341

DEBUG : Found following errors:

DEBUG : FATAL errors 5

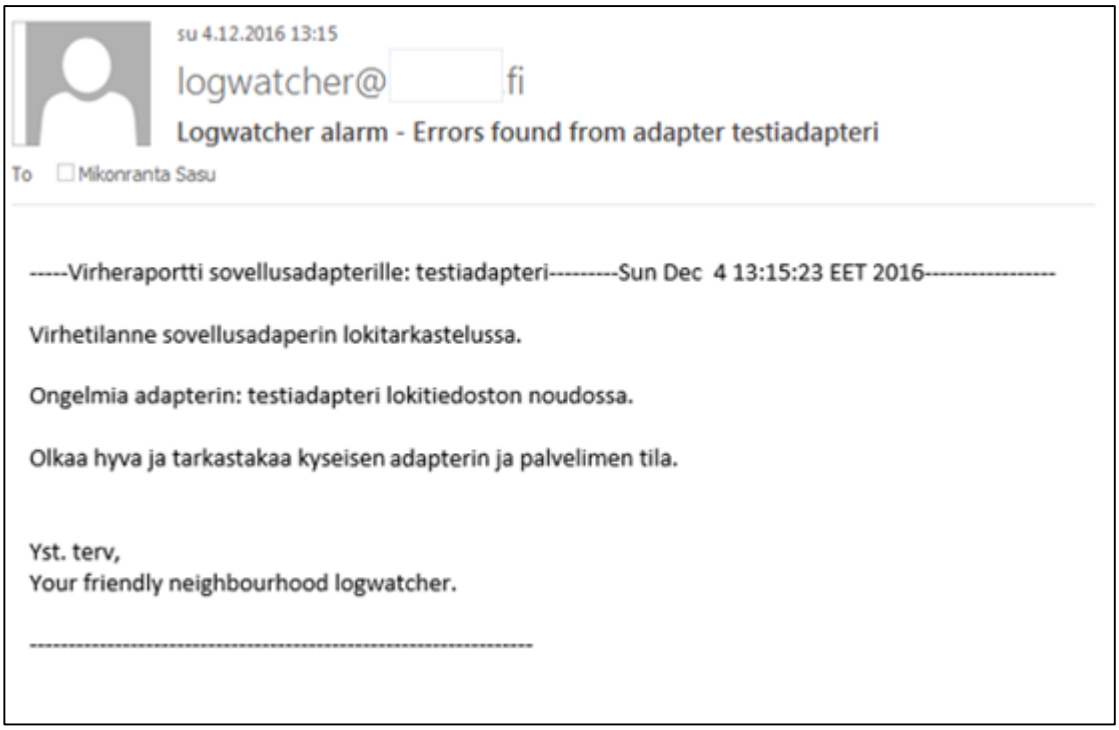
Starttime: Sun Dec 4 15:45:53 EET 2016

Endtime: Sun Dec 4 15:46:10 EET 2016

Br,
Your friendly neighbourhood Logwatcher

Kuva 24 - Lokiraportti löytyneistä virheistä

Tilanteissa joissa adapterin lokitiedostoa ei saatu haettua syystä tai toisesta, syntyi myös sähköposti-ilmoitus (kuva 25).


su 4.12.2016 13:15

logwatcher@ fi

Logwatcher alarm - Errors found from adapter testiadapteri

To Mikonranta Sasu

-----Virheraportti sovellusadapterille: testiadapteri---------Sun Dec 4 13:15:23 EET 2016----------

Virhetilanne sovellusadaperin lokitarkastelussa.

Ongelmia adapterin: testiadapteri lokitiedoston noudossa.

Olkaa hyva ja tarkastakaa kyseisen adapterin ja palvelimen tila.

Yst. terv,
Your friendly neighbourhood logwatcher.

Kuva 25 - Lokiraportin noutovirhe

Tilanteissa, joissa virheitä ei havaittu, ei tuotettu sähköpostiraporttia. Suurin osa testauksessa tapahtuneista tilanteista oli tämänkaltaisia, joten testauksessa jouduimme osittain luoda tahallisia häiriöitä sovellusadaptereille, jotta saimme ominaisuuksia testattua.

Testauksen aikana ilmeni myös odottamattomia tilanteita. Eräs adapteri teki jatkuvaa toistoa info-tasoisesta ”virhetilanteesta” lokitiedostoonsa, joka kerrytti tälle moninkertaisesti lokirivejä päivässä verrattuna muihin tavallisiin osapuoliin. Tämä aiheutti lokiläpikäynnille huomattavaa viivettä kyseisen osapuolen kohdalla. Ongelmaa ei kuitenkaan

arvioitu vakavaksi, koska lokitarkastelu tapahtuu yhden kerran päivässä ja yöaikaan, jolloin tuloksen valmistuminen ei ollut niin aikakriittistä.

Manuaalisen testauksen jälkeen sovellusadaptereiden valvonnan käynnistysskripti ajastettiin ajettavaksi testiympäristöön päivittäin kello 23.45 alkaen.

7.6 Toiminnan arviointi

Skriptin toimintaa tarkkailtiin työn loppuvaiheessa ensin manuaalisten ajojen kautta ja tämän jälkeen ajastetusti. Testivaiheessa testauksen alaisilla sovellusadapterilla ei käynyt juurikaan luonnollisia virhetilanteita, joten näitä pyrittiin luomaan itseaiheutetusti testauksen vuoksi. Ajot suoritettiin testiympäristössä, joten tämä ei tuonut riskejä tuotantoympäristön toimintaan. Itseaiheutetut virheet eivät myöskään aiheuttaneet testiympäristön siirroille haittaa.

Skripti toimi testauksessa yleisesti kuten oli odotettu, mutta muutama huomio nousi testauksessa esille. Yksi testiosapuolista tuotti odottamattoman suuren lokitiedoston, johtuen tämän lokille muodostuvista ylimääräisistä riveistä. Rivit muodostuivat eräiden aineistokäsittelyiden aiheuttamasta jatkuvasta kierrosta. Rivejä esimerkkitapauksessa oli kertynyt 128 tuhatta kappaletta noin 19 tunnin aikana. Tämän aineiston läpikäynti kesti skriptiltä 28 minuuttia ja 20 sekuntia. Normaali lokitiedosto ilman virheitä oli samassa ajassa kerennyt kasaamaan vain noin 2000 riviä. Laskimme muutaman eripituisen lokin pohjalta aineiston keskimääräiseksi läpikäyntinopeudeksi noin 75 riviä sekunnissa. Pidimme tätä läpikäyntinopeutta riittävänä ja jopa poikkeuksellisten isojen lokien läpikäyntiaika ei ollut liian suuri.

7.7 Jatkokehitysideat

Lopullisen sovellusadapterin lokikäsittelyn rakentamisen jälkeen syntyi ideoita eri testattujen lokikäsittelyn osioiden yhdistämisestä. Tuotteena esimerkiksi Splunk oli hyvin viimeistellyn oloinen. Mikäli tämän asettaisi toimimaan vain paikallisesti esimerkiksi valvontapalvelimella ja lokien noudot suoritettaisiin samalla tekniikalla kuin toteutetulla skriptillä, voisi valvontatyökalusta tehdä huomattavasti monipuolisemman. Tässäkin ideassa

kuitenkin tulee ottaa huomioon Splunkin tuomat ylimääräiset kustannukset, vaikka Splunkin omia adaptereita tai pilvipalvelua ei käytettäisikään.

Testausvaiheessa ilmeni myös muutamia optimointikysymyksiä skriptille ja tämän havaitsemille virheille. Työssä toteutettu toteutus on suunniteltu kerran päivässä ajettavaksi, jolloin virheen havaitsemisaika olisi maksimissaan noin 25 tuntia. Toki tämä on parannus nykytilaan, jossa virheen havainnointiin saattoi mennä huomattavasti enemmän aikaa, mutta jatkokehityksessä skripti voitaisiin muokata ajettavaksi esimerkiksi tunnin tai kahden välein ja tuolloin tämä tarkastelisi vain tiettyä aikaväliä lokista. Tämä kehitysidea on todennäköisesti seuraava edistysaskel tämän käsittelyn jatkokehityksessä.

7.8 Tuotantoonsiirron suunnittelu

Käsittelyn tuotantoonsiirto tulee noudattamaan samaa kaavaa kuin testiympäristöön asennus. Käsittelyn pystytys jouduttaisiin toteuttamaan kertaalleen, ja tämän jälkeen vastaavasti jokaiselle sovellusadapterille tulee suorittaa päivitystoimenpide, liittäessä näitä käsittelyn alaisiksi. Tuotantoympäristössä tulee kuitenkin ottaa huomioon mahdollisimman optimaaliset päivitysajat, jotta sovellusadapterien läpi kulkeva liikenne ei häiriintyisi. Tuotantoonsiirron ajankohdaksi arvioidaan vuoden 2017 alkupuoliskoa.

8 Loppusanat

Sovellusadapterin valvonnan kehitys oli pitkään harkinnassa ollut toimenpide, jota lähdettiin lopulta suunnittelemaan alkukesästä 2016. Työssä tarkasteltiin eri toteutusvaihtoehtoja adapterin valvonnan kehitykselle ja näiden ominaisuuksia arvioitiin. Lopullisena ratkaisuna integraatiokeskuksen ja sovellusadapterin välille luotiin lokivalvontaan perustuva valvontaproseduuri, joka hyödyntää integraatiokeskuksen olemassa olevia toimintoja sekä uutta skripteillä toteutettua toiminnallisuutta. Työtä kehitettiin ja pilotoitiin testiympäristössä loppuvuoden 2016 aikana. Insinöörityön lopputuloksena oli selvitystyön kautta valittu ratkaisuehdotus, joka toteutettiin pilottina testiympäristöön. Toteutus toiminnallaan tuo ympäristön nykytilaan tavoitellun parannuksen valvontaan automatisoimalla sovellusadaptereiden lokivalvontaa. Ratkaisu on myös suhteellisen helposti implementoitavissa ympäristöön, eikä vaadi suuria resursseja. Vaikka ratkaisulla ei voida kor-

vata virheiden korjauksia, ratkaisu silti toimii hyödyllisenä työkaluna, jotta virheet havainnoitaisiin entistä nopeammin ja työn lähtötilanteen luoneen ongelman kaltaisia tapauksia ei tulevaisuudessa tulisi vastaan.

Insinöörityöprosessi oli kaikin puolin opettavainen kokemus. Kuvittelin lähtötilanteessa, että sovellusadapteri jonka parissa yksi- jos toinenkin työtunti on vierähtänyt, olisi tuttu kohde ympäristönsä kanssa. Kuitenkin työhön liittyvän teoriaosuuden avaaminen insinöörityössä ohjasi tutustumaan tarkemmin näiden teknisiin yksityiskohtiin ja yleisesti koko ympäristön yhteistoiminnallisuuden kokonaiskuvaan. Työssä selvityksen alle otetut toteutukset toivat myös näkemystä tuotteistettujen ratkaisujen mahdolliseen hyödyntämiseen jossain toisessa yhteydessä. Lopullinen toteutus, joka päätettiin yksinkertaisen implementointimahdollisuutensa takia toteuttaa bash-skripteillä, ei myöskään kuulunut työn alkuvaiheessa vahvimpiin osaamisalueisiini. Työn ohessa skriptauksen syntaksi alkoi pikkuhiljaa tuntumaan tutummalta ja käsittelyyn saatiin tätä myötä lisää ulottuvuuksia. Skriptausta kuitenkin tuki hyvin paljon ohjelmistokehitystausta Javalla ja yleisen ohjelmointilogiikan sisäistäminen. Työkaluna myös Splunk tuntui oikein mainiolta lyhyen testausperiodinsa aikana. Näkisin tuolle olevan paikkansa, jossain uudemmassa ympäristössä, jossa tämän implementointi voitaisiin huomioida jo ympäristön pystytysvaiheessa.

Työn suurimpana haasteena koin insinöörityön kirjoitusprosessin ja tämän aikataulunhallinnan. Kirjoitusprosessia vaikeutti hieman myös teknisen kuvauksen taso. Aiheen ollessa hieman vieraampi, tekninen kuvaus yritettiin pitää sopivalla tasolla, jotta tämä olisi helpommin ymmärrettävissä. Työn kirjoitus vaati myös jatkuvaa harkintaa mitä pystyin työn salliman esitysluvan puitteissa kirjoittamaan. Haasteena olivat myös aluksi tuotteistettujen ratkaisujen käyttöönotto näitä alun perin käyttämättä. Tekninen toteutus pääpiirteittäin valmistui valinnan jälkeen lopulta melko nopeasti, mutta selvitystyö uusien työkalujen kanssa ja näistä raportoiminen oli suhteellisen työläs kokemus. Teknistä toteutusta rakentaessa nälkä vain kasvoi, kun toteutus alkoi näyttämään konkreettisia tuloksia ja tämän vuoksi testailua välillä venytettiin pitkälle yöhön. Lopussa kiitos kuitenkin seisoi ja työ saatettiin valmiiksi. Sovellusadaptereiden parissa työ kuitenkin jatkuu vielä tämän parissa, joten implementointivaihe ja mahdollinen jatkokehitys ovat vielä edessä.

Sovellusadaptereiden uusi valvontaproseduuri on aktiivisena testiympäristössä, ja tämän tuotantoonsiirtoa aloitetaan valmistelemaan alkuvuodesta 2017.

Lähteet

1 Sami Tähtinen, Järjestelmäintegraatio – Tarve, vaihtoehdot, toteutus. Gummerus Kirjapaino Oy 2005 Jyväskylä.

2 Erilaiset integraatiomallit. Verkkodokumentti. <http://www.itewiki.fi/opas/integraatiot/#Erilaiset-integraatiomallit>. Luettu: 02.09.2016.

3 Hub and Spoke [or] Zen and the Art of Message Broker Maintenance. Verkkodokumentti. <http://www.enterpriseintegrationpatterns.com/ramblings/03_hubandspoke.html>. Luettu 30.10.2016.

4 Middleware. Verkkodokumentti. <http://searchsoa.techtarget.com/definition/middleware>. Luettu: 10.10.2016.

5 Agile coding in enterprise IT: Code small and local. Verkkodokumentti. <http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/microservices.html>. Luettu 03.11.2016.

6 Enterprise Integration EAI vs. SOA vs. ESB Author - Anurag Goel. Verkkodokumentti. <http://hosteddocs.ittoolbox.com/Enterprise%20Integration%20-%20SOA%20vs%20EAI%20vs%20ESB.pdf>. Luettu 01.11.2016.

7 Mikropalvelut nousivat hypen huipulle – mitä hyötyä niistä on? Perssi Hämäläinen 21.3.2016. Verkkodokumentti. <http://www.tivi.fi/Kaikki_uutiset/mikropalvelut-nousivat-hypen-huipulle-mita-hyotya-niista-on-6534379>. Luettu 10.11.2016.

8 Hallitsetko API:si. Digiarjessa blogi. Tero Kivisaari 16.11.2016. Verkkodokumentti. <http://blog.digia.com/hallitsetko-apisi>. Luettu 03.12.2016.

9 Message-Oriented Middleware (MOM). Verkkodokumentti. <https://docs.oracle.com/cd/E19575-01/820-6424/aeraq/index.html>. Luettu 30.10.2016.

10 Fuse Message Broker - Exploring JMS. Red Hat, Inc. Verkkodokumentti. <https://access.redhat.com/documentation/en-US/Fuse_Message_Broker/5.5/html-single/Exploring_JMS/index.html>. Luettu 03.11.2016.

11 JMS message structure. Verkkodokumentti. <http://www.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ac24863_.htm>. Luettu 30.10.2016.

12 Spring Framework JMS Integration Tutorial. Kuva poimittu kohdasta 13min 43sec. Verkkodokumentti. <https://www.youtube.com/watch?v=9WGSVnhlOHE>. Katsottu 01.11.2016.

13 Using a Reverse Proxy to Keep Sensitive Data out of the DMZ. Verkkodokumentti. <http://www.jscape.com/blog/bid/86810/Using-a-Reverse-Proxy-to-Keep-Sensitive-Data-out-of-the-DMZ>. Luettu 19.11.2016.

14 IBM WebSphere MQ Explorer. Verkkokuva. <https://kb.b2bits.com/download/attachments/655424/AddQueues_4.png?version=2&modificationDate=1395094636000&api=v2>. Tallennettu 18.11.2016.

15 Sanomamonitorointi SAP ohjelmistolla. Verkkokuva. <https://wiki.scn.sap.com/wiki/download/attachments/311134297/MessageMonitor33.jpg?version=1&modificationDate=1351015837000&api=v2>. Katsottu 03.12.2016.

16 Forwarder overview - Splunk Cloud User Manual. Verkkodokumentti. <http://docs.splunk.com/Documentation/SplunkCloud/6.5.0/User/AddDataUnivFrwrder>. Luettu 20.11.2016.

17 Getting Started with Logstash. Verkkodokumentti. <https://www.elastic.co/webinars/getting-started-logstash?elektra=home&iesrc=ctr>. Luettu 26.11.2016.

18 Komentojen ajastaminen. Verkkodokumentti. <https://www.linux.fi/wiki/Komentojen_ajastaminen>. Luettu 04.12.2016.

Lokikäsittelyn prosessikaavio