VAASAN YLIOPISTO

TEKNILLINEN TIEDEKUNTA

TIETOTEKNIIKKA

Timo Hankkila

YRITYKSEN TIETOJÄRJESTELMIEN INTEGROINTI

Esimerkkinä myyntikonfiguraattorin integrointi toiminnanohjausjärjestelmään

Diplomityö, joka on jätetty tarkastettavaksi diplomi-insinöörin tutkintoa varten Vaasassa 25.3.2010.

Työn valvoja Merja Wanne

Työn ohjaaja Anja Jousranta

ALKULAUSE

Tämä diplomityö on pitkän ja monivaiheisen prosessin tulos. Diplomityön aihekin on vaihtunut prosessin aikana. Matkan varrella on myös ollut esteitä, joita en matkalle lähdettäessä edes kuvitellut tieltä löytäväni. Kaikista esteistä ja vaikeuksista on jotenkin aina selvitty eteenpäin, yksi asia kerrallaan.

Olen oppinut tämän prosessin kautta paljon myös itsestäni ja siitä, miten vaatimatonta ihmisen taivallus joskus on – kuin ensiaskeleitaan haparoiva lapsi. Aikanaan lapsi oppii kävelemään vaikka kehitys onkin välillä hyvin vaatimatonta. Pienet ja vaatimattomat, yksinkertaiset asiat voivat olla suurilta näyttävien asioiden takana ja niiden salaisuus.

Haluan kiittää ja lyhyesti muistaa kaikkia, jotka ovat tähän diplomityöprojektiin osallistuneet tavalla tai toisella. Kiitos vanhemmilleni Erkki ja Hertta Hankkilalle kaikesta tuesta opintojeni ja elämäni aikana. Työnohjaaja Anja Jousranta: Kiitos, että annoit minulle tarvitsemaani vapautta ja luottamusta työn tekemisessä, ohjasit tarvittaessa oikeaan suuntaan. Wapicen integrointitiimille haluan myös esittää kiitokseni: Jens Winberg, Sami Kyllönen ja Jukka Mäkinen, kiitos hyvästä yhteistyötä integrointiprojektissa. Erityiskiitos Jensille rakentavista keskusteluista ja kommenteista diplomityön aikana. Haluan myös kiittää entisiä esimiehiäni, Pauli Perastoa ja Anna-Kaisa Saarta kaikesta tuesta ja avusta.

Kiitos Ovaskaisen perheelle: Kimille ja Lauralle, Askolle ja Iltalle, sekä erityisesti rakkaalleni Helmille. ”*Suurempi kuin elämä on Sinun armosi”* (Psa 63:4).

Vaasassa 18.3.2010 Timo Hankkila

SISÄLLYSLUETTELO

LYHENTEET JA TERMIT

Ad hoc: Tiettyä ongelmaa varten räätälöity ratkaisu, ennalta valmistelematon ja tilapäinen ratkaisu.

API: *Application Programming Interface*. Ohjelmointirajapinta.

CRM: *Customer Relationship Management*. Asiakkuudenhallintajärjestelmä.

DSS: *Decision Support System.* Päätöstukijärjestelmä; auttaa yrityksen päätöksentekijöitä ja johtoa tekemään yritystä koskevia päätöksiä.

EAI: *Enterprise Application Integration.* Metodi tai filosofia yrityksen liiketoimintaprosessien ja datan saumattomaan integrointiin sovellusten ja järjestelmien välillä. Yrityksen tietojärjestelmien integraatioalusta; ohjelmistokerros jonka tarkoituksena on mahdollistaa sovellusten keskinäinen kommunikointi sekä datan jakaminen.

ERP: *Enterprise Resource Planning.* Toiminnanohjausjärjestelmä.

FTP: *File Transfer Protocol*. Tiedostonsiirtoprotokolla tiedostojen siirtämiseen.

HTTP: *Hypertext Transfer Protocol*. Internetissä käytetty hypertekstin siirtoprotokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon, esimerkiksi HTML-sivun lähettämiseen palvelimelta selainohjelmaan.

LDAP: *Lightweight Directory Access Protocol*. Hakemistopalvelujen käyttöön tarkoitettu verkkoprotokolla. Käyttötarkoitus on pääasiassa käyttäjäntunnistus.

Middleware: Väliohjelmisto; ohjelmistokerros, joka yhdistää sovellukset tai sen eri komponentit toisiinsa.

MIS: *Management Information Systems.* Johdon tietojärjestelmä; päätöksentukijärjestelmä, joka auttaa johtoa esimerkiksi organisaation toiminnan suunnittelussa tai strategisen suunnitelman laatimisessa. Tietojärjestelmien kehittämistä ja käyttämistä siten, että ne tukevat yrityksen liiketoimintaa sekä auttavat liiketoiminnalle asetettujen tavoitteiden saavuttamista.

MQ: *Message Queue.* Viestijono.

MRP: *Material Requirement Planning/Manufacturing Resource Planning (MRP II).* Materiaalinhallintajärjestelmä/tuotannonohjausjärjestelmä.

RPC: *Remote Procedure Call.* Proseduurien etäkutsu.

XML: *Extensible Markup Language.* Merkintäkieli tai standardi, jolla tiedon merkitys on kuvattavissa tiedon sekaan (metadata). XML-kieli on rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja.

XSL(T): *Extensible Stylesheet Language* (*Transformations*). Tyylistandardi, jonka avulla määritellään XML-dokumentin esitysmuoto. XSLT on XML-pohjainen merkintäkieli XML-dokumenttien muunnoksiin.

W3C: *World Wide Web Consortium*. Kehittää yhteisiä ja yhteensopivia World Wide Web:in pelisääntöjä ja teknologioita (spesifikaatioita, ohjeita, ohjelmistoja sekä työkaluja).

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Timo Hankkila
Diplomityön nimi: Yrityksen tietojärjestelmien integrointi
Esimerkkinä myyntikonfiguraattorin integrointi toiminnanohjausjärjestelmään
Valvojan nimi: Professori Merja Wanne
Ohjaajan nimi: KTT Anja Jousranta
Tutkinto: Diplomi-insinööri
Laitos: Tietotekniikan laitos
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suunta: Ohjelmistotekniikka
Opintojen aloitusvuosi: 1999
Diplomityön valmistumisvuosi: 2010 **Sivumäärä:** 106

TIIVISTELMÄ:

Yrityksissä on usein käytössä paljon erilaisia sovelluksia ja tietojärjestelmiä, joiden tarkoitus on tukea yrityksen liiketoimintaa. Eristäytyneet tietojärjestelmät ovat ongelma, koska niissä olevaa dataa ei voida hyödyntää organisaatiolaajuisesti automaattisesti, vaan siihen vaaditaan paljon manuaalista työtä, joka on usein virhealtista ja kallista.

Tämän tutkielman tavoite oli tarkastella yleisellä tasolla sitä, miten yrityksen tietojärjestelmien integrointi voitaisiin toteuttaa laadukkaasti, että yrityksen sovellukset voisivat kommunikoida keskenään ja tukea yrityksen liiketoimintaa parhaalla mahdollisella tavalla. Tutkielmassa pyrittiin kirjallisuuteen ja artikkeleihin pohjautuen löytämään käytännössä hyväksi havaittuja ja oikeissa integraatioprojekteissa testattuja malleja yrityksen tietojärjestelmien integraation laadukkaaseen toteuttamiseen.

Yrityksen tietojärjestelmien integrointi on ennen kaikkea metodi tai filosofia yrityksen liiketoimintaprosessien ja datan saumattomaan integrointiin sovellusten ja järjestelmien välillä. Se voidaan ymmärtää myös tietojärjestelmien integraatioalustaksi, ohjelmistokerrokseksi, jonka tarkoituksena on mahdollistaa sovellusten keskinäinen kommunikointi ja datan jakaminen. Se ei ole kuitenkaan yksittäinen sovellus tai tekniikka, vaan tapa ajatella ja hahmottaa yrityksen IT-arkkitehtuuria kokonaisuutena.

Yksinkertaisimmillaan integraatiossa yhdistetään kaksi sovellusta toisiinsa esimerkiksi viestijonotekniikalla kahdenvälistä arkkitehtuuria käyttäen. Usean sovelluksen tapauksessa käytetään väylä- tai tähtiarkkitehtuuria (keskitin), jolloin kaikki sovellukset kommunikoivat keskenään yhden keskuksen, integrointipisteen kautta. Tähän soveltuu parhaiten väliohjelmisto, joka toimii rajapintana eri sovellusten välillä ja tulkkaa keskenään epäyhteensopivat viestit vastaanottavan sovelluksen ymmärtämään muotoon. XML on järjestelmäintegroinnin kannalta merkittävä dataformaatti, ja se mahdollistaa rakenteisen tiedon siirtämisen, tallentamisen ja jakelun Internetissä – sovellus- ja laitteistoriippumattomasti.

AVAINSANAT: Järjestelmäintegraatio, yrityksen tietojärjestelmät, liiketoiminnan kehittäminen, väliohjelmisto, integraatiomallit.

UNIVERSITY OF VAASA
Faculty of technology
Author: Timo Hankkila
Topic of the Thesis: Enterprise Application Integration
Case Sales Configurator Integration with Enterprise Resource Planning System
Supervisor: Professor Merja Wanne
Instructor: DBA Anja Jousranta
Degree: Master of Science in Technology
Department: Department of Computer Science
Degree Programme: Degree Programme in Information Technology
Major of Subject: Software Engineering
Year of Entering the University: 1999
Year of Completing the Thesis: 2010 **Pages:** 106

ABSTRACT:

Enterprises often have many applications and information systems which are supposed to support business. Isolated systems become a problem, because the data stored in them cannot be used efficiently through the organization automatically, but requires a lot of manual work, which is expensive and error-prone.

The goal of this thesis was to examine enterprise application integration on abstract level and study how to implement good quality integration in a way that enterprise applications would support the business as efficient as possible. The research aimed to find best practices and patterns in application integration based on reference books and articles, which are tested in real life integration projects.

Enterprise application integration is first and foremost a method or a philosophy of seamless integration of both data and business processes between applications and systems. It can also be understood as a system integration platform, a software layer that enables applications to communicate and to share data with each other. Nevertheless it is not a certain tool or technology, but a way of thinking and perceiving company's IT-architecture as a whole.

At simplest, integration connects two applications together using for example message queue technology and point-to-point architecture. In case of several applications bus or mesh architecture is used, in which case all the applications communicate via one hub, point of integration. This is best implemented by middleware, which acts as an interface to applications and transforms the incompatible message format to a format that receiving applications use. XML is a significant data format in system integration, and it enables transferring, storing and distributing of structured data – device and application independently.

KEYWORDS: Enterprise Application Integration, information systems, business development, middleware, integration patterns.

1. JOHDANTO

Yrityksillä on nykyään käytössään yhä suurempi määrä erilaisia tietojärjestelmiä, joiden tarkoitus on tukea yrityksen liiketoimintaa sekä päätöksentekoa. Erilaiset toiminnanohjaus-, asiakkuudenhallinta-, tuotannonohjaus- ja myyntijärjestelmät ovat monelle yritykselle oleellinen osa liiketoimintaa, ja niistä on tullut korvaamattomia, kun liiketoiminta on monimutkaistunut ja kasvanut. Samalla järjestelmien keskinäisestä kommunikoinnista on tullut yleinen ongelma. Tyypillisesti yrityksellä on käytössä kirjava joukko eri sovelluksia eri tarkoituksiin ja tietojärjestelmistä on tullut saarekkeita, joilla ei ole tehokasta yhteyttä keskenään eikä ulkomaailmaan.

Tarve tietojärjestelmien integroinnille yrityksissä on ollut olemassa jo pitkään, mutta ei ole olemassa hyvää yleispätevää ratkaisua, miten asiat tulisi suunnitella ja toteuttaa. Monissa yrityksissä IT-arkkitehtuurin suunnitteluun ei ole panostettu nimeksikään, ja tuloksena on lähes kaaosmainen joukko keskenään epäyhteensopivia ja huonosti kommunikoivia järjestelmiä. Tämä tietojärjestelmien hajanaisuus ja epäyhteensopivuus on yhä useampien yritysten ongelma. Tietojärjestelmien tulisi aidosti palvella yrityksen liiketoimintaa, ja niiden tulisi pystyä jakamaan keskenään informaatiota ja hyödyttämään yrityksen liiketoimintaprosesseja mahdollisimman saumattomasti. Monissa yrityksissä tilanne on kuitenkin käytännössä aivan jotain muuta – tietojärjestelmät voivat pahimmillaan olla jopa liiketoiminnan este (Linthicum 2000: xvii).

Tämä diplomityön aihe on syntynyt varsin yleisestä ongelmasta, joka yrityksiä tänä päivänä vaivaa: Informaatiota ja dataa on vuosien saatossa kertynyt valtava määrä mitä erilaisimpiin sovelluksiin ja tietojärjestelmiin, mutta niitä ei useinkaan voida hyödyntää saumattomasti koko organisaatiossa. Tämä ongelma koskee myös yrityksen liiketoimintaprosesseja, koska ne on sidottu yrityksen tietojärjestelmiin – monen yrityksen liiketoiminta on lähes täysin tietojärjestelmien varassa. Ongelman kiertämiseksi joudutaan tekemään paljon manuaalityötä, jotta yhdessä paikassa olevaa dataa tai informaatiota voitaisiin hyödyntää muualla organisaatiossa ja sen järjestelmissä. Tämä manuaalinen työ on usein virhealtista ja puuduttavaa, usein myös täysin turhaa mikäli tietojärjestelmät osaisivat kommunikoida keskenään ilman ihmisen tulkkausta. Monissa korkean teknologian yrityksissä käytetään tänäkin päivänä huomattavasti aikaa ja resursseja esimerkiksi siihen, että Excel-taulukon tietoja muokataan käsin, jotta niitä voidaan hyödyntää jossain toisessa järjestelmässä. Tästä allekirjoittaneella on myös henkilökohtaista kokemusta useammastakin eri yrityksestä, jotka eivät ole mitään ”nyrkkipajoja” vaan isoja, kansainvälisiä yrityksiä. Perusongelma on siis se, että yrityksen sovellusten ja tietojärjestelmien informaatiota sekä liiketoimintaprosesseja ei voida hyödyntää kunnolla koko organisaatiossa.

Tietojärjestelmien integrointi on aiheena niin laaja ja myös haastava, että tässä tutkielmassa ei edes yritetä vastata kaikkiin tietojärjestelmä- ja sovellusintegrointia koskeviin kysymyksiin ja ongelmiin. Tutkielmassa ei myöskään syvennytä tietoteknisiin yksityiskohtiin eli sovellusintegraatioon, vaan tarkoitus on tarkastella tietojärjestelmien integrointia kokonaisuuden kannalta – yrityksen tietojärjestelmien integroinnin näkökulmasta, jossa yrityksen työntekijät käyttä-

vät yrityksen liiketoimintasovelluksia jonkun tietyn liiketoiminnon suorittamiseen.

Tämän tutkielman tavoite on tarkastella yleisellä tasolla sitä, miten yrityksen tietojärjestelmien integrointi voitaisiin toteuttaa, mitä asioita olisi hyvä ottaa huomioon integrointiarkkitehtuuria suunniteltaessa, että yrityksen sovellukset voisivat kommunikoida keskenään ja tukea yrityksen liiketoimintaa parhaalla mahdollisella tavalla. Tutkin tietojärjestelmien integrointia kirjallisuuteen pohjautuen, tavoitteena löytää hyviä malleja ja kriteerejä integroinnin laadukkaaseen toteuttamiseen. Tutkielmassa keskitytään kirjallisuudessa usein esiintyvään termiin EAI (*Enterprise Application Integration*) tietojärjestelmien integroinnin yhteydessä. Selvitän, mitä tähän käsitteeseen liittyy ja minkälaisia malleja yrityksen tietojärjestelmien integrointiin on olemassa.

2. YRITYKSEN TIETOJÄRJESTELMÄT

Tässä luvussa käsittelen yrityksen tietojärjestelmiä yleisellä tasolla: Mitä tietojärjestelmällä tarkoitetaan alan kirjallisuudessa, miten se eroaa sovelluksesta, minkälaisia tietojärjestelmiä yrityksillä tyypillisesti on käytössä ja mitä tarkoitusta ne palvelevat. Esittelen yleisellä tasolla tietojärjestelmien historiaa sekä niiden käyttötarkoitusta yrityksissä.

2.1 Tietojärjestelmien peruskäsitteitä

Tietojärjestelmien yhteydessä esiintyvät usein sanat *data* ja *informaatio*. Data on rakenteetonta faktaa – tiedon esitys, joka on luettavissa, viestitettävissä tai käsiteltävässä muodossa, eräänlaista käsittelemätöntä ”raakadataa”. Yrityksissä dataa kerätään kaikista järjestelmistä sekä ympäristöstä, ja syötetään tietokantaan. Informaatiolla tarkoitetaan käsiteltyä ja jalostettua dataa – datan ihmiselle tuottama mielle tai merkitys, tietämyksen lisäys. Eri tilanteista riippuen sama tieto voi olla sekä dataa että informaatiota. Dataa prosessoidaan ohjelmistojen avulla siten, että siitä syntyy informaatiota yrityksen johdon tarpeisiin sekä yksilöiden ja organisaatioiden käyttöön yrityksen toimiympäristössä (Pohjonen 2002: 4; McLeod & Schell 2007: 9).

Tietojenkäsittelyksi (*data processing, information processing*) kutsutaan Pohjosen (2002: 4) mukaan erilaisia tietoihin kohdistuvia toimituksia, kuten yhdistelyä, valintaa, uudelleenjärjestämistä tai laskutoimituksia, tai edellä mainittujen toimitusten sarjan suorittamista haluttujen lopputulosten aikaansaamiseksi. Tietojenkäsittely ei ole välttämättä automaattista tietotekniikkaa ja ohjelmistoja hyödyntävää, vaan se voi tapahtua myös manuaalisesti ihmisen suorittamana. Laajemmat tietojenkäsittelykokonaisuudet sisältävät yleensä molempia, sekä manuaalisesti että automaattisesti suoritettavia tietojenkäsittelytehtäviä.

Tietojenkäsittely voi tapahtua epäformaalisti tai ns. *ad hoc*, mutta jos se rajataan koskemaan vain järjestelmällisesti ja formaalisti tapahtuvaa tietojenkäsittelyä, niin oleellista on se, miten tietojenkäsittelytehtävät on määritelty. Tietotekniikassa ohjelmaksi (*program*) kutsutaan tehtävän (tai tehtävien) esitystä joukkona toteutettavaksi tarkoitettuja toimenpiteitä jollain ohjelmointikielellä. Ohjelmistolla (*software*) taas tarkoitetaan tiettyyn tietokoneeseen tai tehtäväalueeseen liittyvää ohjelmakokonaisuutta. Perinteisiä ohjelmistoja ovat varus- ja työkaluohjelmistot (käyttöjärjestelmät ja laiteajurit), ja tunnetuimpia henkilökohtaiset ohjelmistot kuten tekstinkäsittely- ja taulukkolaskentaohjelmistot. Ajanvieteohjelmistot kuten pelit ovat myös tänä päivänä erittäin suosittuja (Pohjonen 2002: 5).

McLeod ja Schell (2007: 10) määrittelevät johdon tietojärjestelmän (*Management Information System, MIS*) siten, että se on tietokonepohjainen järjestelmä, joka mahdollistaa informaation saatavuuden käyttäjille, joilla on samankaltaisia tarpeita. Yritysjohdon tapauksessa yhteinen tarve voi olla esimerkiksi liiketoimin-

nallisten tavoitteiden laatiminen myyntidatan perusteella. Tässä tutkielmassa tietojärjestelmien käyttäjät muodostavat muodollisen organisaation – yrityksen tai sen tytäryhtiön. ATK-sanakirja (2004: 241–242) määrittelee tietojärjestelmän seuraavasti:

1) Ihmistä, tietojenkäsittelylaitteista, tiedonsiirtolaitteista ja ohjelmista koostuva järjestelmä, jonka tarkoitus on tietoja käsittelemällä tehostaa tai helpottaa jotakin toimintaa tai tehdä toiminta mahdolliseksi.

2) Abstrakti systeemi, jonka muodostavat tiedot ja niiden käsittelysäännöt.

Tähtinen (2005: 16) kirjoittaa, että sovellus on tietojenkäsittelytehtävän suorittava tietojärjestelmä, joka koostuu sekä ohjelmistoista että ohjelmistoja käyttävistä ihmisistä. Tietojärjestelmän määritelmä ei ota kantaa siihen, tapahtuvatko tietojenkäsittelytoiminnot automaattisesti vai manuaalisesti. Manuaalinen tietojärjestelmä tarkoittaa järjestelmää, joka edellyttää ihmisen osallistumista tietojenkäsittelyyn. Automaattinen järjestelmä kykenee hoitamaan tietojenkäsittelytoiminnot itsenäisesti (Pohjonen 2002: 5).

Useimmat nykyiset tietojärjestelmät sisältävät sekä manuaalisia että automaattisia osia. Molemmilla osilla täytyy olla oma rajanpintansa (*interface*) niin toisiinsa kuin ”ulkomaailmaankin” eli siihen ympäristöön, jossa tietojärjestelmä toimii. Rajapinnan tarkoitus on määritellä minkälaisia syötteitä (*input*) tietojärjestelmä kykenee vastaanottamaan ja millaisia tulosteita (*output*) se kykenee tuottamaan. Tietojärjestelmä on laajempi käsite kuin ohjelmisto, ja se pitää sisäl-

lään myös tietojenkäsittelyn ympäristön erilaiset organisationaaliset, sosiaaliset ja inhimilliset ulottuvuudet. Tietojärjestelmän ympäristö koostuu kaikista niistä sovelluksista, järjestelmistä ja ihmisistä, joiden kanssa tietojärjestelmä on vuorovaikutuksessa. Tietojärjestelmä saa syötteitä ympäristöstään. Tämä voi tapahtua joko ihmisen välityksellä manuaalisesti esimerkiksi syöttämällä tietojärjestelmään asiakastietoja tai tiedot voidaan noutaa automaattisesti suoraan jostain toisesta järjestelmästä, kuten esimerkiksi asiakashallintajärjestelmästä. Tämän jälkeen tulosteet voidaan välittää tietojärjestelmästä takaisin ympäristöön automaattisesti esimerkiksi toiminnanohjausjärjestelmään tai ihminen voi muokata niitä manuaalisesti ja siirtää ne käsin esimerkiksi Excel-taulukkoon. Vaikka tietojenkäsittely tapahtuisi täysin automaattisesti, tarvitaan yrityksissä aina ihmistä tulkitsemaan informaatiota ja tekemään päätöksiä sen pohjalta (Pohjonen 2002: 5–6).

2.2 Yrityksen tietojärjestelmien historiaa

Tietojärjestelmien kehitysvaiheet voidaan jakaa kolmeen aikakauteen: laskentajärjestelmiin, toimintokohtaisiin järjestelmiin ja integroituihin, toimintoja yhdistäviin tietojärjestelmiin.

Nimi	Ajanjakso	Kohde	Perspektiivi	Esimerkki	Teknologia
Laskenta-järjes-telmät	1950–1980	Yksittäinen käyttötarkoi-tus	Rutiininomaisten laskutoimitusten eliminointi	Palkanlaskenta Kirjanpito Inventaario	Keskustieto-kone Reikäkortti
Toimin-tokohtai-set järjes-telmät	1975–20??	Liiketoimin-to	Yksittäisten liike-toimintaosastojen hallinnon ja toi-minnan paranta-minen tietotek-niikkaa hyväksi-käyttäen	Henkilöstö-resurssit Talousrapor-tointi Tuotanto (MRP ja MRP II)	Keskustieto-kone Yksittäinen PC Tietoverkot ja lähiverkot
Inte-groidut järjes-telmät	2000–...	Liiketoimin-taprosessi	Tietojärjestelmien kehittäminen eril-listen liiketoimin-tayksiköiden inte-groimiseksi orga-nisaation liiketoi-mintapro-sesseiksi	Asiakkuuden-hal-lintajärjestelmä (CRM) Toiminnanoh-jaus järjestelmä (ERP)	PC:t verkossa Asiakas-Palvelin Internet Intranet

Taulukko 1. Yrityksen tietojärjestelmien historia (Kroenke 2007: 196).

Taulukossa 1 on esitetty tietojärjestelmien kehityksen historia eri aikakausina. Laskentajärjestelmien tarkoitus oli vapauttaa työntekijät ikäviltä ja pitkäveteisiltä, toistuvilta laskentatoimenpiteiltä, joita tehtiin rutiininomaisesti. Ensimmäisiä yrityksen tietojärjestelmiä olivat palkanlaskenta ja kirjanpitosovellukset. Varastonhallintasovellukset pitivät kirjaa yrityksen varastosaldoista; varastosaldomäärät tarkistettiin käsin laskemalla noin neljännesvuosittain. Alkuvaiheen järjestelmien ongelmina olivat tekniset vaikeudet. Laskentajärjestelmät säästivät työvoimaa, mutta käytännössä ne tuottivat hyvin vähän informaatiota (Kroenke 2007: 196).

Kroenken (2007: 196) mukaan laskentajärjestelmiä seuraavan aikakauden toimintokohtaiset tietojärjestelmät helpottivat yksittäisen liiketoimintaosaston tehtäviä ja yksittäisiä toimintoja. Tietojärjestelmien suorituskyky ja kapasiteetti li-

sääntyi ja esimerkiksi palkanlaskenta laajeni henkilöstöhallinnoksi, kirjanpito talousraportoinniksi ja varastonhallinta yhdistyi tuotantoon. Kehitys vaikutti yritysten toimintaan tukien liiketoimintaa ja tuottaen lisäarvoa.

Toimintokohtaisien järjestelmien suurin ongelma oli niiden itsellisyys ja hajanaisuus. Tästä syystä toimintokohtaisia sovelluksia kutsutaankin joskus automaatiosaarekkeiksi, koska ne toimivat itsenäisesti ja muista sovelluksista riippumattomasti. Toimintojen väliset yhteydet olivat heikosti rakennettuja, joten järjestelmät eivät kyenneet tuottamaan yrityksen liiketoiminnassa kaivattua hyötyä. Näistä järjestelmistä ei saatu sitä tietoa, mitä yrityksen johto ja päätöksentekijät tarvitsevat liiketoiminnan kokonaisuutta suunniteltaessa; esimerkiksi ostot vaikuttavat tuotantoon, joka vaikuttaa myynnin kautta asiakastyytyväisyyteen, joka puolestaan vaikuttaa myyntiin tulevaisuudessa. Hyvä tuote usein myös myy hyvin, ja jos tuotteella on hyvät oheispalvelut, niin lopputuloksena on tyytyväinen ja sitoutunut asiakas. Applen iPod ja iPhone -tuotteet ovat tästä hyviä esimerkkejä. Päätökset, jotka ovat asianmukaisia yksittäisten toimintojen (kuten osto) kannalta voivat kuitenkin aiheuttaa ongelmia koko liiketoimintaprosessin kannalta. Esimerkiksi puhtaasti oston näkökulmasta voi olla järkevää ostaa komponentit aina sieltä, mistä halvimmalla saadaan, mutta jos tämä vaikuttaa lopputuotteen laatuun tai toimitusaikaan negatiivisesti, niin liiketoiminnan kannalta se voi olla suuri virhearvio ja yritys voi näennäisen säästön takia menettää huomattavasti potentiaalisia asiakkaita.

Kolmannen aikakauden tietojärjestelmien kehityksessä tietojärjestelmiä ei suunniteltu enää helpottamaan yksittäistä liiketoimintaosastoa tai toimintoa,

vaan pikemminkin integroimaan kokonaisen liiketoimintaprosessin aktiviteetteja. Koska nämä aktiviteetit rikkovat osastorajoja, tällaisia järjestelmiä kutsutaan joskus osastojen tai toimintojen välisiksi järjestelmiksi. Niitä kutsutaan myös prosessipohjaisiksi järjestelmiksi, koska ne tukevat kokonaisia liiketoimintaprosesseja (Kroenke 2007: 197).

Siirtyminen yksittäisen käyttötarkoituksen sovelluksista toimintokohtaisiin sovelluksiin oli Kroenken (2007: 197) mukaan suhteellisen helppoa. Toimintokohtaiset sovellukset tarjosivat lisää toiminnallisuutta yksittäisen liiketoimintaosaston tai toiminnon tarpeisiin. Vastuunjako oli yksinkertaista ja selkeää eikä liiketoimintaosastojen välistä koordinointia juurikaan tarvittu. Valitettavasti siirtyminen toimintokohtaisista sovelluksista integroituihin, toimintoja yhdistäviin tietojärjestelmiin on vaikeaa. Integroitu tiedon prosessointi vaatii usean eri osaston liiketoimintojen välistä koordinointia, päällekkäisten prosessien poistamista. Selkeää vastuunjakoa ei ole, yrityksen sisäinen kilpailu voi olla ankaraa, ja osastojen välinen, keskinäinen kilpailuhenki voi jarruttaa – pahimmassa tapauksessa estää – uuden järjestelmän kehittämistä ja käyttöönottoa.

Useimmissa yrityksissä tänä päivänä on käytössä sekä toimintokohtaisia että integroituja järjestelmiä. Pysyäkseen kansainvälisessä kilpailussa mukana, yritysten on ennen pitkää pakko hyödyntää integroitujen, liiketoimintaosastojen välisten, (liiketoiminta)prosessipohjaisten tietojärjestelmien etuja. Integroitujen järjestelmien osuus tullee todennäköisesti kasvamaan yrityksissä lähivuosina. Myös voittoa tuottamattomat organisaatiot voivat hyötyä integroiduista tieto-

järjestelmistä esimerkiksi datan jakamisen ja erilaisten toimintojen yhtenäistämisen kautta (Kroenke 2007: 197).

2.3 Tyypillisiä yrityksen tietojärjestelmiä

Tietojärjestelmät voidaan jaotella niiden käyttötarkoituksen perusteella eli millaisia toimintoja niillä suoritetaan (Pohjonen 2002: 7–8; Haikala & Märijärvi 2006: 17):

Toimistoautomaatiojärjestelmä (*office automation system*) on järjestelmä, joka soveltuu omien dokumenttien hallintaan tai työmäärältään pienten töiden automatisointiin. Se koostuu päivittäisessä työssä tarvittavista apuvälineohjelmistoista, kuten esimerkiksi tekstinkäsittely-, sähköposti- ja kalenteriohjelmistot.

Tapahtumankäsittelyjärjestelmällä (*transaction processing system, data processing system, on-line system*) voidaan käsitellä erilaisia organisaation tapahtumia ja transaktioita. Ne ovat tyypillisesti perinteisiä usean käyttäjän järjestelmiä, joissa yhteistä tietokantaa voi käyttää yhtä aikaa useat eri käyttäjät.

Tietojenkäsittely voi tapahtua suorakäsittelynä (*online processing*) tietokantayhteyden välityksellä, jolloin käyttäjät tekemät transaktiot käsitellään heti. Tällaisia järjestelmiä ovat esimerkiksi lipun- ja paikanvarausjärjestelmät. Tietojenkä-

sittely voi olla myös eräkäsittelyä (*batch processing*), jolloin tiedot ensin kerätään ja käsittely suoritetaan myöhemmin tiettynä ajankohtana, esimerkiksi öiseen aikaan, jolloin toimenpide kuormittaa muuta tietojenkäsittelytoimintaa mahdollisimman vähän. Tällaisia järjestelmiä ovat esimerkiksi palkanlaskenta- ja laskutusjärjestelmät.

Reaaliaikajärjestelmä (*real time system*) kontrolloi jotakin ympäristöä siten, että se kerää tietoa ympäristön toiminnasta, käsittelee sitä ja palauttaa tulokset välittömästi ympäristöön mahdollisesti muuttaen ympäristön toimintaa. Reaaliaikajärjestelmissä ohjelmiston on kyettävä suorittamaan vaaditut toiminnot ennalta määrättyjen aikavaatimusten mukaisesti. Usein reaaliaikajärjestelmiltä vaaditaan myös automaattisuutta eli kykyä toimia luotettavasti ilman ihmisen ohjausta, mahdollisesti pitkiäkin aikoja. Reaaliaikajärjestelmiä ovat muun muassa sulautetut järjestelmät, esimerkiksi automaattiset hälytysjärjestelmät.

Päätöstukijärjestelmä (*decision support system, DSS*) tuottaa tyypillisesti informaatiota päätöksenteon tueksi yrityksessä. Informaatiota saadaan analysoimalla organisaatiota koskevaa tietoa esimerkiksi organisaation tapahtumankäsittelyjärjestelmistä. Päätöstukijärjestelmä esimerkiksi auttaa yrityksen myyntipäällikköä laatimaan optimaalisen komission myyntihenkilöstölle. Järjestelmä voi näyttää eri vaihtoehtojen vaikutukset yritykselle jäävään myyntikatteeseen, kun myyjän saamaa provisio-osaa muutetaan. Myyntipäällikkö voi käyttää tätä tietoa hyväkseen määritellessään myyntikomission siten, että se on riittävän suuri kannustin myyjille, mutta kuitenkin mahdollisimman pieni, jotta yritykselle jäävä hyöty olisi suurin mahdollinen.

Asiantuntija- ja tietämyspohjaisella järjestelmällä (*expert system, knowledge-based system*) tarkoitetaan tyypillisesti sovellusta, johon on ohjelmoitu johonkin tiettyyn, rajattuun erikoisalueeseen liittyvää tietämystä sekä tapoja avustaa ko. erikoisalueeseen liittyvässä päätöksenteossa. Parhaimmillaan tietämyspohjainen järjestelmä voi auttaa päätöksentekotilanteessa simuloimalla ihmisasiantuntijan toimintaa, esimerkiksi sovellus, joka päättelee maatilan lainoitusehdot tilan ominaisuuksista ja alan lainsäädännöstä.

Prosessinohjaus- ja prosessiautomaatiojärjestelmät (*process control system, process automation system*) ohjaavat ja valvovat yrityksen tuotantoprosessien toimintaa. Prosessinohjausjärjestelmällä voidaan ohjata esimerkiksi öljynjalostusprosessia, paperikonetta tai sähköjakeluverkkoa. Matkapuhelinverkon hallintajärjestelmä, jolla hallitaan verkon eri elementtejä, on hyvä esimerkki prosessiautomaatiojärjestelmän hyödyntämisestä: Sen avulla miljoonat ihmiset voivat samanaikaisesti soittaa ääni- ja datapuheluja sekä lähettää tekstiviestejä ilman, että järjestelmä kuormittuu käyttökelvottomaksi.

Johdon tietojärjestelmä (*management information system*) on eräänlainen päätöstukijärjestelmä, jonka tarkoitus on auttaa yrityksen johtoa esimerkiksi organisaation toiminnanohjauksessa. Ne sisältävät ja prosessoivat sellaista yrityksen dataa, joka kuvastaa yrityksen liiketoimintaa. Niiden tuottama informaatio kuvastaa yrityksen tai sen toimintokohtaisten järjestelmien tilaa, mitä on tapahtunut menneisyydessä, mitä on tapahtumassa juuri nyt ja mitä todennäköisesti tapahtuu tulevaisuudessa. Johdon tietojärjestelmä tuottaa edellä mainittua informaatiota raportointiohjelmistojen ja matemaattisten mallien kautta. Rapor-

tointiohjelmistot tuottavat sekä määräajoin tuotettavia jaksollisia raportteja että erikoisraportteja. Erikoisraportteja (kutsutaan myös *ad hoc*-raporteiksi) tuotetaan yleensä erikseen pyydettäessä ja ennakoimattomiin tarpeisiin. Matemaattiset mallit tuottavat informaatiota yrityksen toimintojen simuloinnin tuloksena. Matemaattisia malleja voidaan käyttää esimerkiksi liiketoiminnan kannattavuuden arvioinnissa tai investointien takaisinmaksuajan selvittämisessä (McLeod & Schell 2007: 11).

2.4 Toiminnanohjausjärjestelmät

Toiminnanohjausjärjestelmä mahdollistaa yrityksen kaikkien resurssien hallinnan organisaatiolaajuisesti sekä niin sanotun yhden syötön periaatteen eli tieto saadaan kaikkien saataville yhdellä syötöllä (McLeod & Schell 2007: 13).

Toiminnanohjausjärjestelmän idea on siinä, että sen avulla voidaan integroida kaikki yrityksen pääprosessit yhteen järjestelmään. Toiminnanohjausjärjestelmä on materiaalihallintajärjestelmän luonnollinen seuraus, ja sen pääasialliset käyttäjät ovat valmistavan teollisuuden yrityksiä. Toistaiseksi toiminnanohjausjärjestelmä edustaa niiden järjestelmien huippua, joilla voidaan hallita eri liiketoimintaprosesseja yli osastorajojen. Toiminnanohjausjärjestelmä integroi myynti-, tilaus-, varasto-, tuotanto- ja asiakaspalvelutoiminnot. Toiminnanohjausjärjestelmät tarjoavat ohjelmiston, tarkoitukseen suunnitellut tietokannat, toimintatavat sekä tehtäväkuvaukset organisaatiorajat ylittävien prosessien integroimi-

seen. Toiminnanohjausjärjestelmän käyttöönotto yrityksissä ei ole aina mikään itsestäänselvyys ja kivuton asia – suuret muutokset aiheuttavat usein aluksi muutosvastarintaa.

Kuva 1. Toiminnanohjausjärjestelmän moduulit Microsoftin näkökulmasta (Microsoft Oy 2009).

Yrityksen liiketoimintaprosessit eli ne työnkulut, joiden kautta yritys tuottaa asiakkailleen lisäarvoa ja itselleen tuottoa, jakautuvat tyypillisesti useiden eri sovellusten alueelle (Tähtinen 2005; 15) Toiminnanohjausjärjestelmän syvin olemus on siinä, että sen lähtökohtana on tarkastella koko yritystä toimintoja yhdistävien prosessien kautta. Kuvassa 1 on esitetty Microsoftin näkemys toiminnanohjausjärjestelmän moduuleista (joukko toisiinsa yhdistettyjä toimintoja), joista asiakas voi valita heidän yrityksen tarpeisiin soveltuvat moduulit. Toiminnanohjausjärjestelmän muotosidonnainen lähestymistapa perustuu dokumentoituihin ja testattuihin liiketoimintamalleihin. Toiminnanohjausjärjestelmän sovellukset sisältävät kattavan joukon olennaisia prosesseja kaikkeen yrityksen liiketoimintaan. Esimerkiksi SAP määrittelee tämän sovellusjoukon

prosessisuunnitelmaksi (*process blueprint*), ja dokumentoi jokaisen liiketoimintaprosessin kaavioksi, jonka kuvaamisessa on käytetty standardeja symboleja.

Koska toiminnanohjausjärjestelmä perustuu menettelytapoihin, joiden muoto on tarkoin määritelty, yritysten täytyy muokata heidän toimintatapansa ja prosessinsa toiminnanohjausjärjestelmän prosessisuunnitelman mukaisiksi. Mikäli tätä ei tehdä lainkaan tai se tehdään puutteellisesti, järjestelmä ei voi toimia tehokkaasti tai pahimmassa tapauksessa se toimii väärin. Jossain tapauksissa on mahdollista mukauttaa toiminnanohjausjärjestelmä joihinkin liiketoimintaprosesseihin, jotka eivät täysin vastaa prosessisuunnitelmaa, mutta pääsääntöisesti tällainen menettely on usein ongelmallista ja tulee myös ajan myötä kalliiksi (Kroenke 2007: 215–216).

Toiminnanohjausjärjestelmissä kaikki yrityksen data on tallennettuna yhteen keskitettyyn tietokantaan. Keskitetyn tiedon (tietokannan) ansiosta käyttäjien on helppo saada nopeasti tarvittavaa tietoa yhdestä paikasta. Kun yritys on saanut otettua toiminnanohjausjärjestelmän onnistuneesti käyttöön, se voi saavuttaa huomattavia hyötyjä järjestelmän käytön myötä. Kuitenkin siirtyminen yksittäisistä, toimintokohtaisista sovelluksista toiminnanohjausjärjestelmään ei yleensä ole helppoa ja yksinkertaista vaan täynnä erilaisia haasteellinen ja usein hidas operaatio. Erityisesti yrityksen toimintatapojen muuttaminen vastaamaan toiminnanohjausjärjestelmän prosessisuunnitelmaa on käytännössä osoittautunut suureksi haasteeksi useille yrityksille. Ääritapauksissa tästä on muodostunut sudenkuoppa, joka on estänyt toiminnanohjausjärjestelmän onnistuneen

käyttöönoton yrityksessä ja järjestelmän implementoinnista on jouduttu luopumaan kokonaan (Kroenke 2007: 216).

Toiminnanohjausjärjestelmään siirtymisestä on tullut myös kallista – ei ainoastaan uusien ohjelmisto- ja laitehankintojen takia vaan uusien toimintatapojen kehittämisen, henkilöstön kouluttamisen, datakonversioiden, konsulttipalkkioiden sekä muiden kehityskulujen myötä.

ERP-järjestelmän hyödyt:
✓ Tehokkaat ja pätevät liiketoimintaprosessit
✓ Varaston pieneneminen
✓ Läpimenoajan lyheneminen
✓ Laadukkaampi asiakaspalvelu
✓ Laajempi, reaaliaikainen kokonaiskuva ja käsitys yrityksen tilasta
✓ Parempi kannattavuus

Taulukko 2. Toiminnanohjausjärjestelmän merkittävimmät hyödyt (Kroenke 2007: 216).

Toiminnanohjausjärjestelmän (liiketoiminta)prosessisuunnitelmat on testattu ja koeajettu useissa sadoissa yrityksissä. Toiminnanohjausjärjestelmään kuuluu pätevät liiketoimintaprosessit, jotka ovat usein myös hyvin tehokkaita. Toiminnanohjausjärjestelmän käyttöönotto yrityksessä ei tarkoita sitä, että yrityksen täytyy ”keksiä pyörä uudelleen” liiketoimintaprosessien suhteen – pikemminkin päinvastoin, ne hyötyvät samoista liiketoimintaprosesseista, jotka ovat jo auttaneet monia yrityksiä menestymään (Kroenke 2007: 216).

Toiminnanohjausjärjestelmä mahdollistaa yrityksen kaikkien resurssien hallinnan organisaatiolaajuisesti, ja tämän ansiosta yritykset ovat voineet vähentää jossain tapauksissa hyvin merkittävästi varastomääriään. Paremman suunnittelun ansiosta ei ole välttämätöntä ylläpitää suuria varastopuskureita. Lisäksi tuotteet jäävät varastoon lyhyemmäksi aikaa, joskus ainoastaan muutamaksi tunniksi tai päiväksi. Toiminnanohjausjärjestelmä hyödyttää valmistavia yrityksiä myös läpimenoaikojen lyhentymisenä. Toiminnanohjausjärjestelmän tehokkaampien liiketoimintaprosessien ja paremman informaation ansiosta tilaustenkäsittelyprosessi paranee - yritykset voivat reagoida paremmin uusiin tilauksiin tai muutoksiin jo tehdyissä tilauksissa. Käytännössä tämä näkyy siten, että yritys voi toimittaa tuotteitaan asiakkaalle nopeammin. Jossain tapauksissa yritys voi saada maksun toimitetusta tuotteesta ennen kuin ovat itse maksaneet tuotteen raaka-ainekuluista aiheutuneet kustannukset (Kroenke 2007: 216–217).

Kroenken mukaan (2007: 217) toimintokohtaisten tietojärjestelmien ongelmaa eli datan ristiriitaisuus- ja hajaantumisongelmia ei toiminnanohjausjärjestelmissä ole, koska kaikki järjestelmän data on tallennettu integroituun tietokantaan. Tärkeä data on myös nopeasti saatavilla, koska kaikki asiakasta, tilausta tai vastaavaa kokonaisuutta koskeva data sijaitsee yhdessä paikassa. Yritys voi näin ollen tuottaa parempaa informaatiota asiakkailleen tilauksista, tuotteista ja muista asiakasriippuvaisista asioista. Asiakaspalvelun paraneminen on usein myös edullisempaa yritykselle. Toiminnanohjausjärjestelmän implementoineet yritykset voivat usein sekä valmistaa että myydä tuotteitaan toimintokohtaisia järjestelmiä käyttäviä kilpailijoitaan kustannustehokkaammin, sillä varastot py-

syvät pieninä, läpimenoajat lyhyinä ja asiakaspalvelu edullisena, joten yrityksen kannattavuus paranee.

Toiminnanohjausjärjestelmän käyttöönottoon liittyy riskejä tai kysymyksiä, jotka yrityksen on hyvä tiedostaa päätöksenteossa. Kroenke (2007: 219a) kirjoittaa, että vaikka toiminnanohjausjärjestelmän valmiit ja dokumentoidut liiketoimintaprosessit (*process blueprint*) sekä toimintatavat ovat hyvin hiottuja ja testattuja, ja niiden hyödyt ovat ilmeiset, niiden käyttöönotto voi aiheuttaa yllättäviä ongelmia. Toiminnanohjausjärjestelmän liiketoimintaprosessit edellyttävät sitä, että yritys muokkaa liiketoimintaprosessinsa vastaamaan toiminnanohjausjärjestelmän dokumentoituja prosesseja. Tällöin esimerkiksi yrityksen myyntiprosessin täytyy vastata toiminnanohjausjärjestelmän myyntiprosessia. Vaihtoehto on räätälöidä toiminnanohjausjärjestelmän prosesseja vastaamaan yrityksen omia liiketoimintaprosesseja, mutta tähän harva yritys ryhtyy, koska se on vaikeaa ja työlästä toteuttaa, riskialtista, kallista ja vaatii usein paljon konsultointia. Räätälöidyissä liiketoimintaprosesseissa on myös se ikävä puoli, että yritys joutuu itse sitomaan resursseja näiden prosessien ylläpitoon, koska ne eivät sisälly toiminnanohjausjärjestelmän toimittajan tuotekehitykseen ja ylläpitoon eivätkä ole siis ”standardeja” prosesseja.

Yrityksen johto joutuu ottamaan kantaa siihen, miten liiketoimintaprosesseja muutetaan, jotta ne vastaavat toiminnanohjausjärjestelmän prosesseja. Yrityksen omat liiketoimintaprosessit eivät ole välttämättä kovin hyvin (jos ollenkaan) dokumentoituja eivätkä niistä ole muut perillä kuin niiden toteuttajat. Tähän yrityksen liiketoimintaprosessien selvitystyöhön ja sopeuttamiseen kuluu pal-

jon aikaa eikä se ole kivuton ja yksinkertainen prosessi. Olemassa olevien toimintatapojen ja prosessien muuttamisessa kohtaa usein muutosvastarintaa työntekijöiden taholta. Käytännössä toiminnanohjausjärjestelmän implementointi onnistuu harvoilta yrityksiltä omin päin, vaan he joutuvat turvautumaan järjestelmän toimittajan konsultaatioon tai kolmannen osapuolen tukeen käyttöönoton yhteydessä. Tätä ei aina välttämättä ymmärretä kunnolla tai oteta huomioon toiminnanohjausjärjestelmää hankittaessa (Kroenke 2007: 219a).

Kun toiminnanohjausjärjestelmän hankkinut yritys joutuu sopeuttamaan omia liiketoimintaprosessejaan vastaamaan toiminnanohjausjärjestelmää (mikäli eivät halua räätälöidä järjestelmään omia prosessejaan), niin herää useita kysymyksiä. Esimerkiksi autoteollisuudessa, jos ajan myötä kaikki auton osien toimittajat käyttävät samoja parhaiksi toteamiaan liiketoimintaprosesseja ja toiminnanohjausjärjestelmän toimintoja (mahdollisesti vieläpä saman toimittajan järjestelmää), niin eivätkö he ala ennen pitkää muistuttaa toisiaan? Eikö niistä tule ajan myötä toinen toisensa klooneja? Mikä on heidän kilpailuetunsa, jos kaikki käytännössä toimivat samoin? Mitä tapahtuu innovaatiolle? Vaikka yksi valmistaja onnistuisikin kehittämään liiketoimintaprosessejaan ja hyötymään innovaatiosta, niin siirtävätkö toiminnanohjausjärjestelmän toimittajat tämän kilpailuedun yrityksen kilpailijoille jo seuraavan järjestelmäpäivityksen yhteydessä? Tarkoittaako toiminnanohjausjärjestelmän testatut ja dokumentoidut liiketoimintaprosessit käytännössä sitä, että yksikään yritys ei voi säilyttää innovaatioetuaan?

Yritys voi yrittää suojautua tältä riskiltä siten, että sen toiminnanohjausjärjestelmässä on vain ja ainoastaan heidän tarpeisiinsa räätälöityjä prosesseja, joita yritys myös ylläpitää itse. Täydellinen suoja ei tämäkään ole, koska suurin riski liittyy usein yrityksen henkilöstöön ja siihen, että miten lojaalia se on yritystä kohtaan. Yrityksen prosessi-innovaatiot voivat siirtyä kilpailijalle muutaman avainhenkilön mukana, jos he päättävät vaihtaa työnantajaa. Yrityksen täytyy omassa riskienhallinnassaan päättää, mikä on yrityksen kannalta paras kompromissi toiminnanohjausjärjestelmän räätälöinnin suhteen. Täysin räätälöity ratkaisu on käytännössä kallein mahdollinen vaihtoehto ja sitoo eniten yrityksen omia resursseja. Miten paljon yritys on valmis sitomaan aikaa ja resursseja toiminnanohjausjärjestelmän räätälöintiin riskien pienentämiseksi? Tällaisiin kysymyksiin yritysjohdon tulisi kyetä ottamaan kantaa päätöksenteon yhteydessä (Kroenke 2007: 219; 219a–219b).

Suurimpia toiminnanohjausjärjestelmien toimittajia ovat saksalaislähtöinen, monikansallinen suuryritys SAP sekä sen merkittävimmät kilpailijat Oracle ja Microsoft. Esimerkiksi SAP:in käyttäjiä on maailmanlaajuisesti 12 miljoonaa yli 120 maassa (McLeod & Schell 2007: 13–14; Kroenke 2007: 215).

2.5 Yrityksen tietojärjestelmät osana organisaatiota

Tietojärjestelmiä on käytännössä kaikissa yrityksissä vaikka niiden olemassaoloa ei välttämättä aina tiedosteta. Tietojärjestelmän luonteeseen kuuluu se, että

ne eivät välttämättä ole aina automaattisia tai yrityksen muodollisesti määrittelemiä, minkä takia niistä ei olla kaikissa yrityksissä ja organisaatioissa tietoisia. Jotta voidaan käsitellä tietojärjestelmien merkitystä ja asemaa yrityksen kannalta, tässä rajataan yrityksen käsite toimintayksikköön (*functional organisation*). Pohjonen (2002: 8) kirjoittaa, että toimintayksikkö on ”kokonaisuus, joka on rajattavissa ympäristöstään ja jolla on omat tavoitteensa ja jonka toiminnan seurauksena toimintayksikölle annetusta syötteestä (esimerkiksi raaka-aineet, tieto) syntyy jokin tulos (esimerkiksi tuote)”.

Käsitteenä toimintayksikkö on hierarkkinen eli toimintayksikkönä tarkasteltavan kokonaisuuden osia voidaan myös tarkastella omina toimintayksikköinään. Esimerkiksi yhteiskunta voi olla toimintayksikkö, jolla on omat tavoitteensa ja jolla on käytettävissä omat resurssit ja tuotantotekijät niiden saavuttamiseksi. Yhteiskunnan osa voisi olla esimerkiksi oppilaitos, jonka tehtävänä on kouluttaa ihmisiä sekä tuottaa tutkintoja yhteiskunnan resursseja, opettajia, hyödyntämällä. Oppilaitoksen osa voisi puolestaan olla esimerkiksi teknillinen tiedekunta, joka olisi oma toimintayksikkönsä, ja sen resursseja puolestaan opettajat ja muu henkilökunta (Pohjonen 2002: 8).

Toimintayksikön sisäiset toiminnot on jaoteltu kolmeen osa-alueeseen: Ohjaustoimintoihin, perustoimintoihin sekä tukitoimintoihin. Perustoiminnot ovat niitä toimintoja, joita ilman toimintayksikkö (yritys) ei voisi olla olemassa. Esimerkiksi kaikki tuotteen valmistamiseen liittyvät toiminnot ovat perustoimintoja. Ohjaustoiminnot ovat puolestaan lähinnä yritysjohdon toimintoja, ja näitä voi olla esimerkiksi toiminnan suunnitteluun, toimeenpanoon ja valvontaan liit-

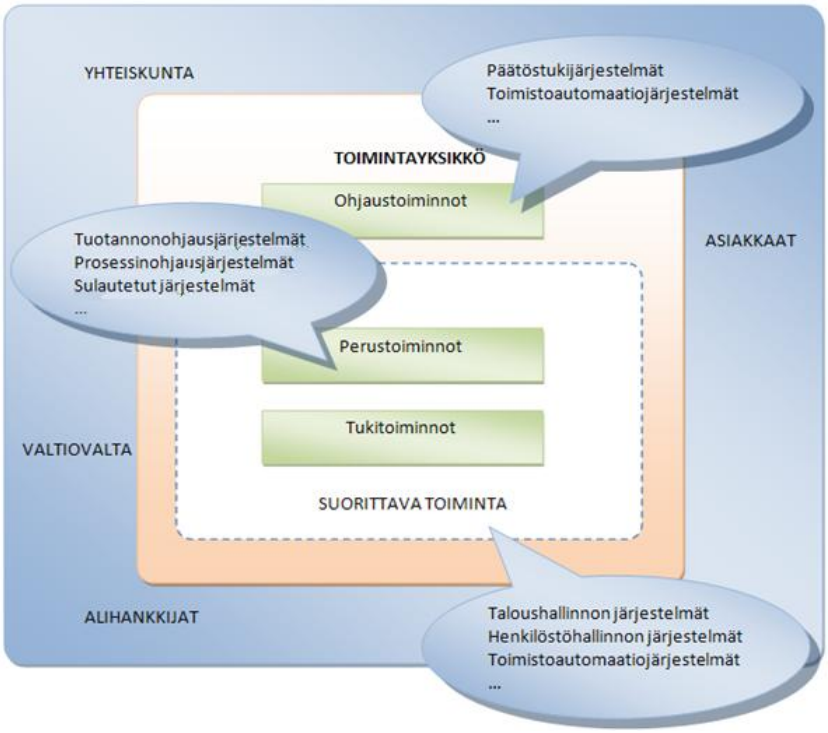
tyvät toiminnot. Tukitoiminnoilla tarkoitetaan henkilöstöhallintoon, taloushallintoon, tietohallintoon, tuotekehitykseen sekä huoltoon liittyviä toimintoja. Toimintayksikön suorittavaan toimintaan sisältyy perus- ja tukitoiminnot. Toimintojen keskinäisellä vuorovaikutuksella on myös oma merkityksensä toimintayksikön eli tässä tapauksessa yrityksen toiminnassa. Tietojärjestelmien kannalta tässä on tarkasteltu toimintayksikköä järjestelmänä, jossa Pohjosen (2002: 9) mukaan ”joukko vuorovaikutteisia komponentteja toimii yhteisen tavoitteen saavuttamiseksi vastaanottamalla syötteitä ja tuottamalla niiden pohjalta tulosteita organisoidussa muunnosprosessissa” (Pohjonen 2002: 8–9).

Tämän päivän yrityksille toimivat sovellukset ja tietojärjestelmät ovat niin tärkeitä, että ne ovat usein jopa edellytys liiketoiminnalle. Kun tietojärjestelmät ovat kriittinen osa organisaatiota, niiden toimimattomuus voi pahimmassa tapauksessa lamauttaa yrityksen toiminnan kokonaan. Tietojärjestelmien käytön syitä on karkeasti kolmea eri tyyppiä:

- Perus-, liike- ja operatiivisten toimintojen tukeminen
- Johdon päätöksenteon tukeminen
- Strategisen kilpailuedun saavuttaminen

Tietojärjestelmien tarkoitus on tukea ja osaltaan myös mahdollistaa varsinaista liiketoimintaa yrityksissä – niitä ei hankita eikä ylläpidetä niiden itsensä vuoksi. Toimistoautomaatiojärjestelmillä (erilaiset toimisto-ohjelmistopaketit, kuten MS Office, Lotus Notes jne.) pyritään helpottamaan päivittäisiä rutiineja, kuten

tekstinkäsittelyä, sähköpostin lähettämistä ja kalenterin ylläpitoa. Tapahtumankäsittelyjärjestelmillä hallitaan yrityksen tuotantoa kokonaisuutena, ja niitä käytetään myös muiden yrityksen perus- ja tukitoimintojen apuna. Päätöstukijärjestelmillä taas seurataan, raportoidaan ja ennustetaan tuotannon ja muun toiminnan muutoksia, ja analysoitava data saadaan usein esimerkiksi yrityksen tapahtumankäsittelyjärjestelmistä. Monissa yrityksissä käytetään usein myös tietämysjärjestelmiä esimerkiksi avustamaan tuotantohäiriöiden syiden selvittelyssä sekä strategisten mahdollisuuksien etsinnässä sekä kilpailuedun saavuttamisessa.



Kuva 2. Tietojärjestelmien roolit yrityksessä (Pohjonen 2002: 11).

Kuvassa 2 on esitetty tietojärjestelmien eri roolit ja niiden asema yrityksessä. Tietojärjestelmät ovat tavallisesti mukana kaikessa yrityksen toiminnassa: tietojärjestelmä voi olla osa yrityksen perustoimintoja, sillä voi olla avustava rooli yrityksen tuki- tai ohjaustoiminnoissa tai se voi olla osa yrityksen valmistamaa tuotetta. Oli tietojärjestelmä hankittu palvelemaan mitä yrityksen toimintoa tai osa-aluetta tahansa, tietojärjestelmä tai sovellus ei koskaan eikä missään yrityksessä saa olla mikäli yrityksen liiketoiminta ei nimenomaan ole tietojärjestelmien kehittäminen ja toimittaminen asiakkaille (esimerkiksi SAP). Kaikille yrityksen tietojärjestelmille ja niiden toiminnalle yhteistä onkin se, että niiden tarkoitus on palvella yritystä ja sen liiketoimintaa sille asetettujen tavoitteiden saavuttamiseksi – tämä pätee kaikkiin organisaatioihin ja niiden tietojärjestelmiin. Tietojärjestelmät eivät ratkaise automaattisesti kaikkia yrityksen ongelmia vaan niitä täytyy osata hyödyntää tehokkaasti (Pohjonen 2002: 11).

3. YRITYKSEN TIETOJÄRJESTELMIEN INTEGROINTI

Tässä luvussa käsittelen yrityksen tietojärjestelmien ja sovellusten integrointia ja sen problematiikkaa yleisellä tasolla, esittelen alan termistöä sekä tarkastelen alan kirjallisuudessa olevia erilaisia malleja yrityksen tietojärjestelmien integrointiin. Tutkielmassa keskitytään järjestelmäintegraatioratkaisuun *(Entrerprise Application Integration*), ja esittelen sen hyviä ja huonoja puolia sekä soveltuvuutta yrityksen tietojärjestelmien integrointiin.

David, Liu, Roxburgh, Mason, Nadhan ja Slater (2003: 9) määrittelevät yrityksen sovellusten integroinnin seuraavasti: ”turvallista ja organisoitua liiketoimintaprosessien ja/tai datan jakamista yrityksen sovellusten välillä”.

Yksinkertainen datakonversio-taulukko tuotekoodeille	
Tuotekoodi asiakkuudenhallintajärjestelmässä:	
Formaatti:	cccc-nnnnnnnnn
Esimerkki:	COMP-334455667
Tuotekoodi toiminnanohjausjärjestelmässä:	
Formaatti:	nnnnnnnnncccc
Esimerkki:	34455667COMP

Taulukko 3. Esimerkki järjestelmäintegraatio-ongelmasta (Kroenke 2007: 205).

Taulukossa 3 on esitetty tietojärjestelmäintegraatioon liittyvä yksinkertainen ongelma: Yrityksen asiakkuudenhallintajärjestelmä käyttää erilaista tapaa mer-

kitä tuotekoodi kuin toiminnanohjausjärjestelmä. Joka kerta, kun tuotekoodeja sisältävää dataa siirretään toiminnanohjausjärjestelmästä asiakkuudenhallintajärjestelmään (tai toisinpäin), tuotekoodit täytyy konvertoida skeemasta toiseen vastaamaan toisen järjestelmän käyttämää muotoa. Kun tämä konvertointiprosessi tehdään useille sadoille tuotekoodeille ehkä puolentusinassa eri järjestelmässä, niin voidaan todeta, että datan prosessointi on hajanaista ja katkonaista toiminnallisten sovellusten välillä. Yrityksen datan integraation puute on seurausta tästä sovellusten keskinäisen tiedonkulun ja -käsittelyn katkonaisuudesta sekä tietojärjestelmien hajanaisuudesta (Kroenke 2007: 215).

3.1 Järjestelmäintegraatio

Tietojärjestelmiä ja sovelluksia on kehitetty usean sukupolven ajan ilman pitkän tähtäimen visiota ja strategiaa. Vasta 2000-luvulla tietojärjestelmien arkkitehtuurisuunnitteluun on alettu kiinnittää huomiota koko yrityksen näkökulmasta. Tietotekniikkaa koskevat päätökset on totuttu tekemään yrityksissä osastotasolla – kukin osasto on valinnut tietotekniset ratkaisunsa sen omien tarpeiden ja tietotaidon perusteella. Esimerkiksi kirjanpito-osasto on saattanut rakentaa ratkaisunsa keskustietokoneen ympärille, kun taas henkilöstöhallinnon osasto on hyödyntänyt hajautettuja järjestelmiä tai asiakas–palvelin -arkkitehtuuria. Lopputuloksena on saattanut olla eri tekniikoiden ja ratkaisumallien sekoitus ja kaaosmainen kokonaisuus. Tämän seurauksena yrityksellä on ollut kokonaisuuden kannalta ”järjestelmä”, jota on ollut lähes mahdotonta integroida ilman

perusteellista arkkitehtuurin uudelleensuunnittelua ja merkittäviä taloudellisia investointeja (Linthicum 2000: 3–4, 7).

Monissa yrityksissä on useita tietojärjestelmiä, jotka ovat vanhoja niin sanottuja savupiipputeollisuuden järjestelmiä. Tällaisia järjestelmiä voivat olla esimerkiksi yrityksen varastohallinnan, myynnin, kirjanpidon ja henkilöstöhallinnan tietojärjestelmät. Nämä ovat tyypillisesti räätälöityjä järjestelmiä kunkin yrityksen erityistarpeisiin käyttäen teknologiaa, joka on silloin ollut suosittua. Monet on toteutettu käyttäen epästandardia sovelluskehitysteknologiaa ja datavarastoa. Vaikkakin teknologia on vanhentunutta, niin nämä järjestelmät ovat silti erittäin tärkeitä yritykselle ja niitä myös käytetään päivittäin. Monet näistä järjestelmistä ovat lisäksi kriittisiä yrityksen liiketoiminnalle, ja niitä on vaikea muuttaa siten, että ne voisivat kommunikoida ja jakaa dataa muiden, kehittyneempien järjestelmien kanssa. Toiminnanohjausjärjestelmien toimittajat, kuten SAP ja PeopleSoft, ja heidän paketoidut sovelluksensa ovat ainoastaan pahentaneet ongelmaa – informaation jakaminen näiden järjestelmien välillä on erityisen vaikeaa, koska useimpia niistä ei suunniteltu kommunikoimaan minkään muun ulkopuolisen tekniikan kanssa (*omisteinen tekniikka, suljetun lähdekoodin ohjelmisto)* (Linthicum 2000: 4).

Järjestelmäintegraatio-termillä (EAI) yleensä tarkoitetaan integrointiin liittyvää filosofiaa tai metodologiaa pikemminkin kuin valmista kaupasta ostettavaa sovellusta tai järjestelmää, vaikka ohjelmistotoimittajat mielellään liittävätkin hyvin myyvän termin heidän tuotteisiinsa. Järjestelmäintegraation edustama teknologia-liiketaloudellinen filosofia keskittyy liiketaloudellisiin kysymyksiin ja

ongelmiin, ja suosittelee että kaikki järjestelmät joko yrityksen sisällä tai sen ulkopuolella voisivat vapaasti jakaa informaatiota ja liiketoimintalogiikkaa tavalla, joka ei aikaisemmin ollut mahdollista toteuttaa käytettävissä olevan tekniikan rajoitteiden takia. Järjestelmäintegraatio yhdistää tietojärjestelmien integroinnin teknologian, metodit sekä filosofian, ja sen kautta halutaan ratkaista yritysten IT-arkkitehtuuriin liittyviä ongelmia (Linthicum 2000: xvii).

Monissa yrityksissä on nykyisin käytössä jokin toiminnanohjausjärjestelmä. Toiminnanohjausjärjestelmän käyttäminen ei kuitenkaan sovellu ratkaisuksi jokaiselle yritykselle. Esimerkiksi palvelualan yritykset saattavat kokea toiminnanohjausjärjestelmän valmistus-orientaation ja tuotantolähestymistavan epäsoveliaana heidän ympäristössään. Jopa jotkut valmistavat yritykset kokevat siirtymäprosessin nykyisestä tuotannonohjausjärjestelmästä (MRP, *Manufacturing Resource Planning*) toiminnanohjausjärjestelmään liian arveluttavana ja riskialttiina toimenpiteenä. Jotkut yritykset taas ovat melko tyytyväisiä olemassa oleviin järjestelmiin eivätkä halua niihin muutoksia tai että ne korvattaisiin uusilla järjestelmillä (Kroenke 2007: 219).

Kroenken (2007: 219) mukaan niillä yrityksillä, joilla toiminnanohjausjärjestelmän käyttö ei syystä tai toisesta sovellu, on kuitenkin ongelmia eristäytyneiden tietojärjestelmien kanssa. Jotkut näistä yrityksistä saattavat valita järjestelmäintegraatioratkaisun ratkaistakseen nämä ongelmat. Järjestelmäintegraatio mahdollistaa olemassa olevien järjestelmien integroimisen tarjoamalla ohjelmistokerroksia, jotka yhdistävät sovellukset toisiinsa.

Järjestelmäintegraation käyttötarkoituksia:

- Yhdistää järjestelmäsaarekkeita uuden ohjelmistokerroksen, väliohjelmiston (*middleware*) kautta
- Mahdollistaa nykyisten sovellusten keskinäisen kommunikoinnin sekä liiketoimintaprosessien ja datan jakamisen
- Tarjoaa integroitua informaatiota eli tiedon parempaa hyödyntämistä: Informaatio on yhtenäistä joka järjestelmässä eikä samaa informaatiota ole turhaan useassa eri järjestelmässä
- Hyödyntää olemassa olevia sovelluksia ja järjestelmiä: Säilyttää perinnejärjestelmät ja toimintokohtaiset sovellukset ennallaan, mutta tarjoaa niille integraatiokerroksen, eräänlaisen julkisivun, jonka kautta muut järjestelmät ja sovellusryppäät voivat olla suoraan yhteydessä yrityksen nykyisiin sovelluksiin
- Mahdollistaa vaiheittaisen siirtymisen toiminnanohjausjärjestelmän käyttämiseen yrityksessä
- Mahdollistaa toimittajariippumattomuuden: Liiketoimintalogiikka kopioidaan olemassa olevista sovelluksista ja implementoidaan ne järjestelmäintegraatioratkaisuun, jolloin tietyn ohjelmistotoimittajan sovellus voidaan vaihtaa toisen toimittajan sovellukseen ilman, että liiketoimintalogiikkaa tarvitsee implementoida uudelleen (Kroenke 2007: 219; Tähtinen 2005: 23–27; Gable 2002)

Kroenke kirjoittaa, että (2007: 219) järjestelmäintegraatioratkaisun ohjelmistokerroks(i)en tarkoitus on mahdollistaa yrityksen nykyisten sovellusten keski-

näinen kommunikointi, ja toimia ohjelmistoalustana datan sekä liiketoiminta-prosessien jakamisessa eri sovellusten välillä. Järjestelmäintegraatio-ohjelmisto voidaan konfiguroida esimerkiksi siten, että se tekee automaattisesti vaadittuja datakonversioita tuotekoodeihin liittyen aiemmin esitellyn taulukon 3 mukaisesti. Kun asiakkuudenhallintasovellus lähettää dataa toiminnanohjausjärjestelmään tässä esimerkkitapauksessa, asiakkuudenhallintajärjestelmä lähettää datan järjestelmäintegraatio-ohjelmistolle. Järjestelmäintegraatio-ohjelmisto tekee tarvittavan datakonversion ja lähettää sen jälkeen konvertoidun datan toiminnanohjausjärjestelmälle. Sama toimenpide tehdään käänteisessä järjestyksessä silloin, kun toiminnanohjausjärjestelmä lähettää dataa asiakkuudenhallintajärjestelmälle.

Vaikka järjestelmäintegraatioratkaisu ei sisälläkään keskitettyä tietokantaa, järjestelmäintegraatio-ohjelmisto säilyttää tiedostoja metadatasta, johon on kuvattu tieto siitä, että mihin data on tallennettu. Käyttäjät voivat käyttää järjestelmäintegraatio-ohjelmistoa ja löytää haluamansa datan sen kautta. Jossain tapauksissa järjestelmäintegraatioratkaisu tarjoaa käyttäjille palveluja, joiden kautta heidän on mahdollista hyödyntää ”virtuaalista integroitua tietokantaa”.

Järjestelmäintegraatioratkaisun **etuja**: Suurin hyöty yritysten kannalta on siinä, että järjestelmäintegraation avulla yrityksen tietojärjestelmäkokonaisuutta on helpompaa hahmottaa ja hallita: yritys voi käyttää olemassa olevia sovelluksia ja kuitenkin välttää useat vakavat ongelmat, jotka ovat tyypillisiä eristetyille järjestelmille, kuten datan hajanaisuus ja ei-yhtenäisyys eri järjestelmien välillä. Järjestelmäintegraatio-ohjelmistoon siirtyminen ei kuitenkaan ole läheskään yh-

tä häiriöaltis ja ongelmallinen prosessi yritykselle kuin toiminnanohjausjärjestelmään siirtyminen, mutta se tarjoaa useita toiminnanohjausjärjestelmän hyötyjä. Jotkut yritykset kehittävät järjestelmäintegraatio-ohjelmistoja niin sanotuksi ponnahduslaudaksi vaiheittaiseen toiminnanohjausjärjestelmään siirtymiseen. Järjestelmäintegraatioratkaisun toinen merkittävä hyöty yrityksen liiketoiminnan kannalta on se, että sen avulla informaation reaaliaikainen saatavuus paranee yrityksen eri tietojärjestelmien välillä, ja informaation eheyttä voidaan ylläpitää useiden eri järjestelmien välillä. Järjestelmäintegraatioratkaisun kautta voidaan myös virtaviivaistaa yrityksen liiketoimintaprosesseja, mikä parantaa organisaation toimintojen tehokkuutta (Kroenke 2007: 219; Tähtinen 2005: 22; Gable 2002).

Järjestelmäintegraatioratkaisun **haittoja**: Merkittävin haitta järjestelmäintegraatioratkaisuun liittyen on tuotekehityskustannukset. Ne muodostuvat usein esteeksi etenkin pienten ja keskisuurten yritysten kohdalla, joilla on käytettävissään hyvin rajallinen budjetti tietoteknisiin hankkeisiin. Toinen merkittävä haitta yrityksen liiketoiminnan kannalta on se, että järjestelmäintegraatioratkaisu vie huomattavasti aikaa sekä sitoo yrityksen resursseja. Yrityksen johto ei välttämättä ymmärrä tai osaa ennakoida etukäteen sitä, miten paljon suunnittelua järjestelmäintegraatioratkaisu vaatii ennen kuin se voidaan toteuttaa. Toisaalta järjestelmäintegraatioon ei välttämättä olla valmiita panostamaan yrityksen johdossa vaikka tietohallinnossa sen edut ymmärrettäisiinkin (Gable 2002).

3.2 Integrointitekniikat

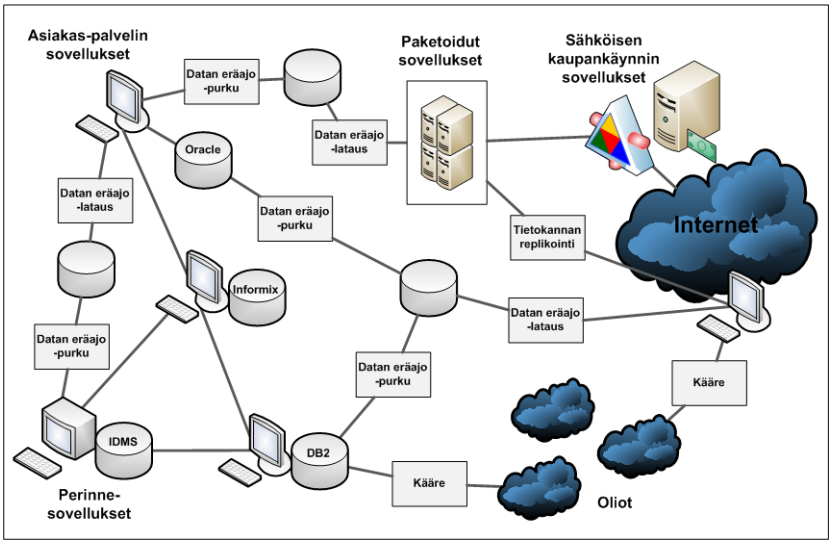
Tässä kappaleessa tarkastelen yleisellä tasolla muutamia integrointiin liittyviä erilaisia tekniikoita. Käsittelen järjestelmäintegraation kannalta merkittävää XML-formaattia sekä XML-dokumenttien muokkaamista. Esittelen lyhyesti XML-dokumenttiformaatin rakennetta ja käyttötarkoitusta, sekä miten sitä voidaan hyödyntää sovellusten integroinnissa. Väliohjelmistoja tarkastelen laajemmin, koska niiden varaan järjestelmäintegrointiteknologia rakentuu. Väliohjelmistoja on monia erityyppisiä, mutta niitä ei käsitellä kaikkia vaan ainoastaan järjestelmäintegraation kannalta merkittävintä, sanomavälittäjää (*Message Broker*).

3.2.1 Väliohjelmistot integroinnissa

Perinteisesti tietojärjestelmien integraatioon liittyviä ongelmia on yritetty ratkaista välikerrosohjelmiston (väliohjelmisto) avulla. Niiden tarjoama ratkaisu on kuitenkin hyvin rajallinen: Perinteinen välikerrosohjelmisto, joka käyttää joko viestijonoa (*message queue, MQ*) tai proseduurien etäkutsua (*remote procedure call, RPC*), mahdollistaa ainoastaan kahdenvälisen (*point-to-point*) arkkitehtuuriratkaisun. Näillä teknisillä ratkaisuilla voidaan muodostaa yhteys järjestelmän A ja järjestelmän B välille. Mikäli tällä arkkitehtuurilla halutaan yhdistää toisiinsa useampi kuin kaksi järjestelmää, niin vastaan tulee ongelmia hyvin nopeasti, kun väliohjelmistolinkkien määrä kasvaa. Lopputuloksena on hyvin monimutkainen, sekava sekä vaikeasti ylläpidettävä ja hallittava kokonaisuus. Mikä pahinta, perinteiset väliohjelmistot vaativat huomattavia muutoksia sekä

lähde- että kohdejärjestelmiin, jotta väliohjelmistokerros voidaan sulauttaa sovellukseen tai tietovarastoon. Väliohjelmistolinkkien määrän lisääntyessä yritysten tietojärjestelmien ja sovellusten välillä lopputulos näyttää lähinnä huonosti suunnitellulta kokonaisuudelta, jossa on tehty useita pieniä integraatioprojekteja, mutta joka ei tue yrityksen liiketoimintastrategiaa. Yrityksen johto kuitenkin tietojärjestelmiltä sitä edellyttää, joten tällaisilla järjestelmillä on hyvin vähän liiketaloudellista arvoa yritykselle (Linthicum 2000: 4–5).

Perinteisellä väliohjelmistotekniikalla, kuten viestijono-ohjelmistoilla, voidaan luoda yhteyksiä sovelluksien välille, mutta nämä kahdenväliset ratkaisut (*point-to-point solution*) luovat yksittäisiä linkkejä eli yhteyksiä usean sovelluksen välille. Tällainen integrointiratkaisu saattaa tulla kalliimmaksi ylläpitää kuin ne sovellukset, jotka se yhdistää toisiinsa. Kahdenvälistä arkkitehtuuria käytettäessä integrointi on käytännössä sovelluksien muokkaamista siten, että ne voivat sekä lähettää että vastaanottaa viestejä toisilta sovelluksilta. Tämä voidaan toteuttaa lukuisilla viestijonoon perustuvilla väliohjelmistoilla (*message-oriented middleware, MOM*), kuten IBM:n MQSeries tai MSMQ (*Microsoft Message Queuing*) -tuotteilla. Toteuttaminen on suhteellisen helppoa kahden integroitavan sovelluksen yhteydessä, mutta integroitavien sovellusten lisääminen tarkoittaa automaattisesti lisää linkkejä sovellusten välille. Jos yritys on onnistuneesti integroinut sovellukset A ja B toisiinsa, ja haluaisi sisällyttää integraatioon myös sovellukset C ja D, täytyy luoda yhteydet jokaisen sovelluksen välillä erikseen. Tämä prosessi kasvaa hyvin nopeasti niin monimutkaiseksi, että kokonaisuudesta tulee kankea ja lähes mahdoton käsitellä ja hallita (Linthicum 2000: 7–8).



Kuva 3. Yrityksen tietojärjestelmäviidakko (Linthicum 2000: 9).

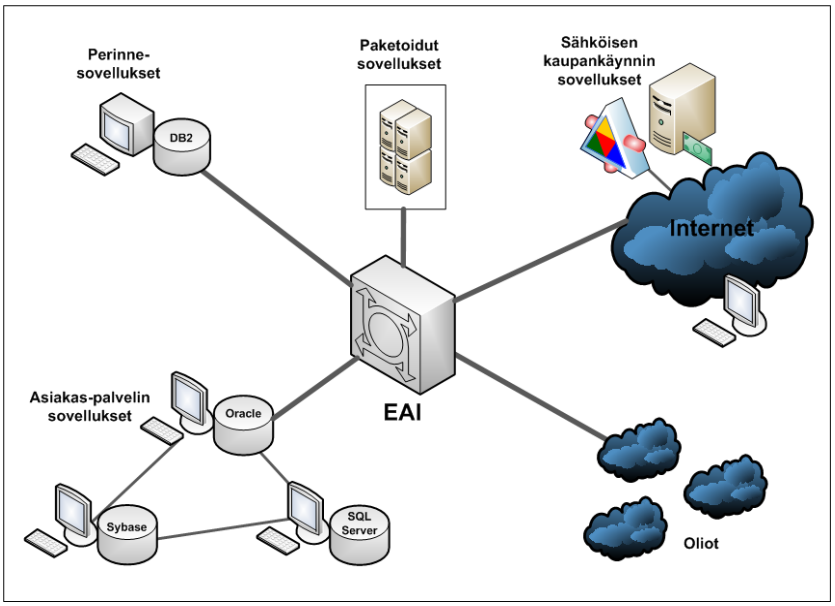
Vaikka tällainen kuvassa 3 havainnollistettu ratkaisu on käytännössä absurdi, perinteisellä väliohjelmistolla ei ole mahdollista toteuttaa muunlaisia integraatioarkkitehtuureja. Monet yritykset, joilla on ollut tarvetta integroida sovelluksiaan, ovat päätyneet vastaavanlaiseen integraatioarkkitehtuuriin ja ylläpitävät sitä tänäkin päivänä – käytännössä tämä näkyy yleensä korkeina ylläpitokustannuksina (Linthicum 2000: 8–9).

Ratkaisu edellä kuvattuun ongelmaan ei ole ihan yksinkertainen, mutta se voidaan jakaa kahteen osaan. Ensinnäkin yrityksen täytyy ymmärtää sekä tietojärjestelmä- että liiketoimintaprosessiarkkitehtuurin **kokonaiskuva**. Yrityksen täytyy ymmärtää ja tuntea kaikki prosessit ja data, mitä sen liiketoimintaan liittyy ja minkälaisia suhteita niillä on toisiinsa, mikä vaikuttaa mihinkin ja mikä merkitys niillä on kokonaisuuden kannalta. Tämän jälkeen yritys voi määritellä kaikki ne sovellukset ja datavarastot, joiden täytyy jakaa informaatiota keske-

nään. Kun tämä perusvaatimusmäärittely on saatu tehtyä, ongelman ratkaisu yksinkertaistuu ja kulminoituu oikean arkkitehtuurin määrittelyyn ja sitä tukevan (tai sen mahdollistavan) tekniikan valitsemiseen ja implementointiin. Käytännössä yritykset ovat hyvin monimutkaisia ja niiden eri prosesseja ja tietojärjestelmien yhteistoimintaa on vaikea hahmottaa. Vastaan tulee esteitä, kuten esimerkiksi eri liiketoimintaosastojen väliset reviiritaistelut ja alati vaihtuvat liiketoimintavaatimukset, joilla on hyvin vähän tekemistä tekniikan kanssa, mutta jotka vaikeuttavat ratkaisun toteuttamista merkittävästi. Laadukas integrointiratkaisu vaatiikin usein muutoksia yrityksen organisaatiossa, ja tähän liittyvät psykologiset tekijät tulisi ottaa huomioon ja käsitellä asianmukaisesti, kun työvoimaan ja työnkulkukaavioihin tehdään rakenteellisia muutoksia (Linthicum 2000: 9–10).

Toiseksi yrityksen täytyy **implementoida uutta tekniikkaa**, jotta yrityksen järjestelmäintegraatio-ongelma saadaan aidosti ratkaistua ja tietojärjestelmien integrointi toteutettua kunnolla. Esimerkiksi perinteiset kahdenväliset väliohjelmistot eivät ole lopullinen ratkaisu. Sen sijaan sanomanvälittäjät (*message brokers*) tarjoavat aidon mahdollisuuden integraation toteuttamiseen sillä aikaa kun muita tekniikoita kehitetään. Sanomanvälittäjien teho perustuu siihen, että niiden kautta voidaan välittää viestejä minkä tahansa tyyppisestä järjestelmästä toiseen viestin formaattia muokkaamalla siten, että vastaanottava järjestelmä sen ymmärtää. Sanomanvälittäjät varmistavat myös sen, että viestit välitetään oikeassa järjestyksessä sekä oikeassa yhteydessä sovelluksen sisällä. Nykyään yrityksen tietojärjestelmien integrointiin liittyy datan ja sanomavälityksen lisäksi oleellisesti myös liiketoimintaprosessien integrointi. Markkinoilla on jo jonkin aikaa ollut olemassa teknologiaa, kuten integrointipalvelimia (esimerkiksi Mic-

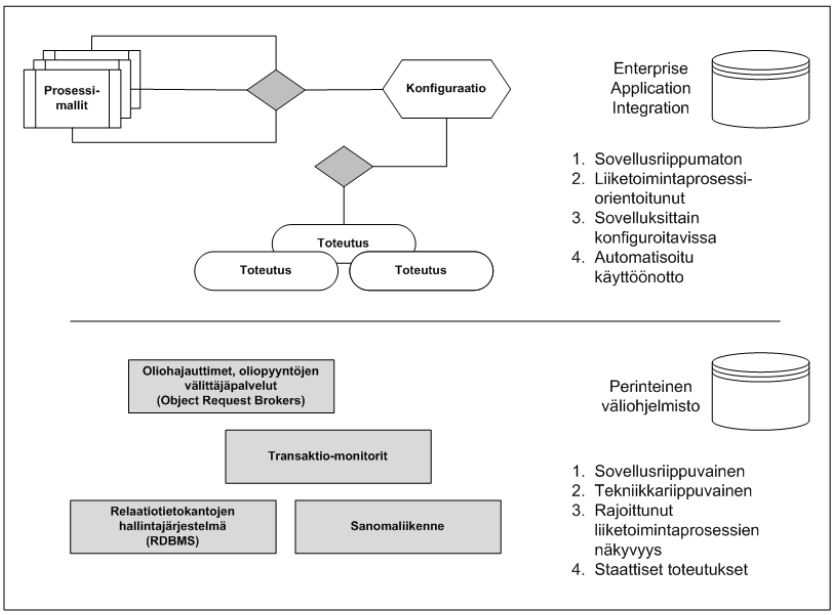
rosoftin BizTalk Server) ja hajautettujen järjestelmien tekniikkaa, joiden avulla kehittäjät voivat luoda uusia liiketoimintaprosesseja ja hyödyntää olemassa olevia prosesseja sekä yrityksen sisällä että niiden välillä (*B2BAI, Business-to-Business Application Integration*) (Linthicum 2000: 10–11, Linthicum 2001: 9–11).



Kuva 4. Järjestelmäintegraatioratkaisu yrityksen tietojärjestelmäviidakon selvittämiseen (Linthicum 2000: 10).

Kuvassa 4 on käytetty järjestelmäintegraatioratkaisua (EAI) yrityksen integraatioarkkitehtuurin suunnittelussa. Yrityksen tietojärjestelmäarkkitehtuuri on selkeytynyt huomattavasti järjestelmäintegraation myötä. Siinä missä perinteisiä väliohjelmistoja hyödynnettäessä yrityksen sovellusarkkitehtuuri oli yhtä kaaosta, järjestelmäintegraatio on yksinkertaistanut ja selkeyttänyt asioita abstraktilla tasolla huomattavasti. Linthicum (2000: 5–6) toteaakin, että järjestelmäintegraatioratkaisu on hyvin erilainen lähestymistapa järjestelmien integrointiin pe-

rinteisiin väliohjelmistoihin ja sovellusintegraatioon verrattuna. Yksinkertaistettuna järjestelmäintegraatioratkaisu luo yhteisen tavan sekä yrityksen liiketoimintaprosesseille että datalle keskustella keskenään sovellusten kautta.



Kuva 5. Järjestelmäintegraatioratkaisun ja perinteisen väliohjelmiston erot (Linthicum 2000: 6).

Kuvassa 5 on esitetty järjestelmäintegraatioratkaisun visio perinteiseen väliohjelmistoon verrattuna:

- Järjestelmäintegraatio keskittyy integroimaan sekä yrityksen liiketoimintaprosesseja että dataa, kun taas perinteinen väliohjelmisto on data-orientoitunutta ja sovelluskohtaista
- Järjestelmäintegraatio käsittää sekä liiketoimintaprosessien että datan uudelleenkäytön ja hajauttamisen

- Järjestelmäintegraatio mahdollistaa sovelluksien integroimisen myös sellaisille käyttäjille, jotka ymmärtävät sovelluksien yksityiskohdista hyvin vähän

Väliohjelmistot ovat ainakin toistaiseksi paras tekniikka siirtää informaatiota sovellusten ja tietokantojen välillä. Kuten aiemmin on jo todettu, perinteiset väliohjelmistot eivät kuitenkaan käytännössä ole vastaus aiemmin esiteltyyn järjestelmäintegraatio-ongelmaan. Uudentyyppiset väliohjelmistotekniikat, kuten sovelluspalvelimet ja sanomavälittäjät, tarjoavat huomattavasti enemmän mahdollisuuksia yrityksen tietojärjestelmien integroinnin laadukkaaseen toteuttamiseen perinteisiin väliohjelmistoihin verrattuna. Nämä uudet väliohjelmistotekniikat ovatkin järjestelmäintegraation ”moottori” ja ydin (Linthicum 2000: 5–6).

3.2.2 Sanomanvälittäjien rooli järjestelmäintegroinnissa

Väliohjelmistoja on olemassa useita erilaisia, ja erityyppiset väliohjelmistot ratkaisevat erityyppisiä ongelmia. Väliohjelmistotekniikat ovat kehittyneet ja eräistä niistä on tullut erittäin lupaavia yritysten järjestelmäintegraatio-ongelman ratkaisun kannalta. Yrityksen johto ja IT-asioista vastaavat henkilöt (esimerkiksi tietohallintopäällikkö) joutuvat puntaroimaan eri vaihtoehtojen välillä ja päättämään, mikä tekniikka soveltuu parhaiten juuri heidän tilanteeseensa. Jotta tämä voidaan tehdä, on ensinnäkin tunnettava hyvin yrityksen IT-arkkitehtuuri, sen prosessit ja ylipäätään liiketoimintaa, jotta voidaan valita sellainen tekniikka, joka parhaiten tukee yrityksen tarpeita. Toiseksi järjestelmäin-

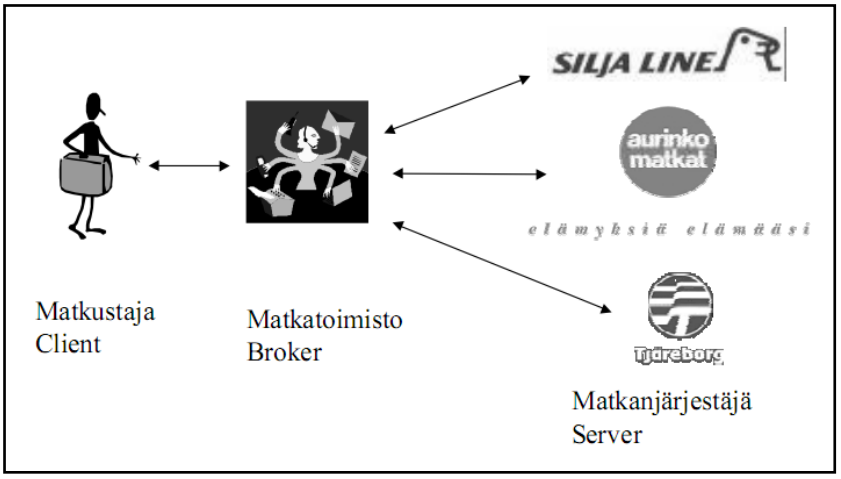
tegraatioon liittyvässä päätöksenteossa tulee myös ymmärtää eri tekniikoiden erot – edut ja haitat – jotta voitaisiin valita oikea tekniikka tukemaan yrityksen liiketoimintaa (Linthicum 2000: 18, 120).

Linthicumin (2000: 131–132) mukaan yritysten järjestelmäintegraatio-ongelman näkökulmasta lupaavin väliohjelmistotekniikka on sanomanvälittäjä (*Message Broker*). Sanomanvälittäjät helpottavat viestien välittämistä kahden tai useamman kohteen (lähettävä ja vastaanottava sovellus) välillä sekä auttavat tulkitsemaan sovellusten semantiikkaa ja ohjelmistoalustojen välisiä eroja. Kohteet voivat olla myös verkkoja, väliohjelmistoja, ja/tai järjestelmiä sovellusten lisäksi. Pelkästään tästä syystä ne soveltuvat erinomaisesti järjestelmäintegraation yhteyteen. Sanomanvälittäjät voivat myös yhdistää useita sovelluksia yhteisiä sääntö- ja reititysmoottoreita käyttäen. Sanomanvälittäjät ovat palvelimia, jotka välittävät viestejä kahden tai useamman lähettävän tai vastaanottavan sovelluksen välillä. Viestien välittämisen lisäksi niiden avulla voidaan muuttaa viestien sisältöä ja skeemaa viestien liikkuessa eri sovellusten ja tietokantojen välillä.

Sanomanvälittäjät eivät ole täydellisiä, eivätkä nekään voi ratkaista aivan kaikkia ongelmia, mutta järjestelmäintegraation kannalta merkittävää on se, että niiden avulla voidaan integroida useita toisistaan eristäytyneitä liiketoimintaprosesseja. Tämä on mahdollista vaikka liiketoimintaprosessit olisivat vanhoja, uusia, perinteisiä, keskitettyjä tai hajautettuja. Sanomanvälittäjät toimivat siltana eri ohjelmistoalustoille ja ohjelmointiympäristöille. Ne voivat myös olla yhteydessä eri sovelluksiin sekä reitittää informaatiota sovellusten välillä lukemat-

tomien väliohjelmistojen tai ohjelmointirajapintojen (*API*) kautta. Sanomavälittäjä sisältää usein kolme päätasoa tai -komponenttia:

1. **Sanomankäsittelymoottori**, jonka avulla voidaan muuttaa informaation esitystapaa sovelluskohtaisesti.
2. **Sääntömoottori**, jolla voidaan luoda sääntöjä viestien käsittelyyn ja muokkaamisen sekä informaation reitittämiseen.
3. **Älykäs viestien reititys**, jonka avulla sanomat voidaan tunnistaa ja reitittää haluttuun paikkaan (Linthicum 2000: 291).



Kuva 6. Sanomanvälittäjä (Laine 2001: 2).

Kuvassa 6 on havainnollistettu sanomanvälittäjän toimintaperiaate. Matkustaja (a*siakas, client*) kommunikoi matkatoimiston kanssa (s*anomanvälittäjä, broker*) ja ilmaisee haluavansa lähteä lomamatkalle. Matkatoimisto toimii palvelun välittäjänä, joka ottaa asiakkaan palvelupyynnön vastaan, ja toimittaa sen edelleen eri matkanjärjestäjille (p*alvelin, server*). Välittäjä myös toimittaa matkanjärjestäjiltä saadut vastaukset (eri tarjoukset) asiakkaalle. Eri matkanjärjestäjät tarjoavat

erilaisia palveluja, ja matkatoimiston täytyy kyetä löytämään eri matkanjärjestäjien joukosta sellainen, joka kykenee toteuttamaan kyseisen asiakkaan palvelupyynnön (Laine 2001: 2–3).

Asiakas pyytää palvelua yhdeltä välittäjältä, joka voi kommunikoida useiden eri matkanjärjestäjien kanssa, eikä asiakkaan tarvitse tietää näistä matkanjärjestäjistä mitään. Asiakas ei myöskään tiedä miten välittäjä kommunikoi matkanjärjestäjien kanssa – asiakkaan näkökulmasta välittäjä on vain yksi palvelurajapinta, mutta välittäjällä voi olla useita eri palveluntarjoajien liittymiä tai vain yksi liittymä, jonka kautta välittäjä saa yhteyden useisiin palveluntarjoajiin. Matkanjärjestäjä toimii rajapintana asiakkaan ja eri matkanjärjestäjien välissä, ja tietää esimerkiksi eri matkanjärjestäjien sijainnin ja yhteystiedot. Matkanjärjestä toteuttaa palvelut, ja kullakin matkanjärjestäjällä voi olla oma palveluvalikoimansa sekä oma varauspalvelunsa (liittymä), jonka muoto poikkeaa muiden matkanjärjestäjien toteutuksesta. Välittäjällä on käytössään kaikki nämä erilaiset varauspalvelut, ja se tekee matkavaraukset asiakkaan puolesta, joten asiakkaan ei tarvitse näistä mitään tietää. Välittäjä ilmoittaa asiakkaalle, kun asiakkaan vaatimusten mukainen matka on varattu ja ilmoittaa asiakkaalle tarvittavat tiedot matkasta, jotta asiakas osaa olla oikeassa paikassa oikeaan aikaan matkalle lähdettäessä (Laine 2001: 3).

3.2.3 Integroinnin toteutuskielet

XML (*Extensible Markup Language*) on merkintäkieli tai standardi, jolla tiedon merkitys voidaan kuvata tiedon sekaan (metadata). XML-standardi on kehitetty

W3C:n (*World Wide Web Consortium*) toimesta, joka vastaa myös World Wide Web:in HTML-standardista. XML muistuttaa HTML-kieltä, jolla Web-sivut kirjoitetaan, ja ne ovat molemmat SGML-kielen (*Standard Generalized Markup Language,* ISO-8879 standardi) yksinkertaistettuja osajoukkoja. XML-kieltä käytetään sekä formaattina dokumenttien tallentamiseen että tiedonvälitykseen järjestelmien ja sovellusten välillä. HTML-kieli suunniteltiin tiedon esittämiseen eikä informaatiosisällön kuvaamiseen, joten se ei soveltunut tiedonsiirtoon sovellusten ja järjestelmien välillä eikä siihen, että samasta lähteestä olisi voitu muodostaa useita eri tulostusformaatteja. XML-kieli on dataorientoitunut ja korvaa lukuisia HTML-standardin vajavaisuuksia, kuten esimerkiksi puutteellista tiedon rakenteisuutta, syntaksin tarkistusmekanismia sekä heikkoa integroitavuutta eri sovelluksiin ja järjestelmiin (W3C Suomen toimisto; Eckstein 1999: 1–2; World Wide Web Consortium 2009).

XML:stä on tullut lyhyessä ajassa maailmanlaajuinen formaatti datan välittämiseen järjestelmästä toiseen. HTML-formaattiin verrattuna XML soveltuu erinomaisesti tietojärjestelmien ja sovellusten integrointiin sen omaisuuksiensa ansiosta. Se on rakenteellinen tiedon kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja selkeämmin. XML:n avulla data voidaan esittää käytännössä rajattomassa määrässä eri rakenteita, joten se soveltuu erinomaisesti dataformaatiksi järjestelmäintegraatioratkaisuihin. XML mahdollistaa myös rakenteisen tiedon siirtämisen, tallentamisen ja jakelun Internetissä sovellus- ja laitteistoriippumattomasti. Esimerkiksi perinnejärjestelmien data voidaan koota XML-dokumentteihin, ja data voidaan siirtää XML-formaatissa eri sovelluksiin ja järjestelmiin. Vastaanottava järjestelmä tulkkaa XML-dokumentin datarakenteen ymmärtämäänsä muotoon. XSL-tyylistandardin (*Extensible Stylesheet Language*)

avulla määritellään XML-dokumentin elementtien esitysmuoto ja varmistetaan, että XML-dokumentit ovat halutun näköisiä sovellusympäristöstä riippumatta (W3C Suomen toimisto; World Wide Web Consortium 2009).

XML-dokumentin datarakenteen muuntamiseen käytetään esimerkiksi XSLT (*Extensible Stylesheet Language Transformations*) ja XPath (*XML Path Language*) -tekniikoita. XPath:in avulla XML-dokumentista voidaan helposti paikantaa yksi tai useampi osa rakenteisesta datasta, ja viitata paikannettuun dataan esimerkiksi ohjelmakoodissa. Järjestelmäintegroinnissa tästä ominaisuudesta on huomattavasti hyötyä. XSLT:n avulla voidaan määritellä, mitkä XPath:illa paikannetut osat lähde-XML -dokumentista halutaan muuntaa kohdedokumentiksi. XSLT-muunnoksessa syntyy uusi XML-dokumentti, jossa haluttu data on esitetty halutussa muodossa. XML-data voidaan tarvittaessa esittää myös muissa kuin XML-formaatissa, mutta XML on nykyään hyvin yleisesti käytössä ja soveltuu hyvin sovellusintegrointiin, joten tässä tutkielmassa tarkastellaan ainoastaan XML:ää (Holman 2002: xv; World Wide Web Consortium 2009).

XML-dokumentilla on looginen ja fyysinen rakenne, joiden tulee olla synkroniset. Fyysinen rakenne koostuu joukosta tiedon säilytysobjekteja, entiteettejä. Looginen rakenne koostuu yhdestä tai useammasta elementistä, joista muodostuu hierarkinen rakenne (puu). Puurakenteessa on yksi juurielementti (*root*), jonka sisällä voi olla yksi tai useita sisar- ja/tai lapsielementtejä. XML-dokumentin elementti merkitään seuraavasti:

```
<Body>Tämä teksti on muotoiltu Body-elementin mukaisesti.</Body>
```

Elementti koostuu kahdesta tagista: aloitus- ja lopetustagista, jotka merkitään hakasulkeilla <> ja </>. Kuten HTML:ssä, aloitus- ja lopetustagien sisältö (teksti) muotoillaan elementin sääntöjen mukaisesti. Elementeillä voi olla myös attribuutteja:

```
<Hinta valuutta=”euro”>67.80</Hinta>
```

Tässä elementissä attribuutti on määritelty aloitustagin sisään, on nimeltään *valuutta* ja sille on annettu arvo lainausmerkkien sisään: *euro*. Myös tyhjiä elementtejä voidaan käyttää esimerkiksi XML-dokumentin käsittelyä (*parse*) varten:

```
<Tyhjä_tagi></Tyhjä_tagi> TAI
<Tyhjä_tagi />
```

Elementtejä voi myös olla rajaton määrä sisäkkäin (lapsielementti):

```
<elementti_1 attribuutti_1="arvo_1">
  <elementti_2 attribuutti_2="arvo_2">
  ...
    <elementti_n attribuutti_n="arvo_n">
    </elementti_n>
  </elementti_2>
</elementti_1>
```

XML-dokumentin oikeellisuus käsittää kaksi eri asiaa: hyvin muodostettu ja validi. Jotta XML-dokumentti olisi hyvin muodostettu (*well-formed*), sen täytyy täyttää seuraavat vaatimukset:

- ✓ Dokumentissa on täsmälleen yksi juurielementti
- ✓ Ei-tyhjillä elementeillä on sekä aloitus- että lopetustagi. Tyhjät elementit voidaan merkitä erikseen yllä kuvatulla tavalla
- ✓ Kaikkien attribuuttien arvot on määritelty lainausmerkkien sisään
- ✓ Elementtejä voi olla sisäkkäin, mutta ne eivät mene ristiin toisten elementtien kanssa

XML-dokumentti on validi, mikäli se on jonkun dokumenttityyppimäärittelyn mukainen, esimerkiksi DTD (*Document Type Definition*) tai XML-skeema (*XML-Schema, W3C*). DTD ja XML-skeema ovat XML-kuvauskieliä, joiden avulla määritellään XML-dokumentin rakenne, esimerkiksi elementit sekä niiden väliset suhteet ja arvojen arvoalueet. Validi XML-dokumentti sisältää viittauksen johonkin dokumenttityyppimäärittelyyn (tiedosto), kaikki sen elementit ja attribuutit ovat kyseisen dokumenttityypin mukaisia sekä se noudattaa kyseisen dokumenttityypin kielioppia (Eckstein 1999: 2–4; World Wide Web Consortium 2009).

XML-dokumentin prosessoinnissa (esimerkiksi tekstinkäsittelysovelluksessa) käytetään yleensä kolmea eri tiedostoa: XML-tiedosto, tyylitiedosto (*stylesheet*) sekä dokumenttityyppimäärittely-tiedosto. *XML-tiedosto* sisältää dokumentin varsinaisen datan, joka on kuvattu XML-elementteihin käyttämällä tageja, ja jotka mahdollisesti sisältävät myös attribuutteja. *Tyylitiedosto* määrittelee, miten

XML-tiedoston elementit tulisi esittää, oli sitten kyseessä tekstinkäsittelyohjelma tai Web-selain. Yhteen XML-dokumenttiin voidaan soveltaa useita eri tyylitiedostoja käyttötarkoituksesta riippuen, jonka ansiosta data voidaan esittää usealla eri tavalla ilman, että itse dataa muokataan. Tämä on erittäin käytännöllinen ominaisuus XML-formaatissa, ja siitä on paljon hyötyä sovellusintegroinnissa, koska datan esitystapoja voi olla käytännössä lukemattomia. Esimerkiksi jokaiselle asiakkaalle voidaan räätälöidä heidän näköisensä lasku, logoa ja fonttia myöten. XML-formaatin suosio sovellusintegraatiossa perustuukin XML-formaatin filosofiaan erotella toisistaan data ja sen esitysmuoto. *Dokumenttityyppimäärittely-tiedosto* sisältää säännöt, joiden mukaan XML-dokumentin elementit, attribuutit ja kaikki muu data on määritelty, sekä elementtien, attribuuttien ja datan keskinäiset suhteet XML-yhteensopivassa dokumentissa (Eckstein 1999: 5).

Esimerkki DTD-määrittelystä (diplomityo.dtd -tiedosto):

```
<!ELEMENT diplomityo (otsikko, asia+, tekija+, huomautus?)>
<!ATTLIST diplomityo pvm CDATA #REQUIRED
                     sivulkm CDATA #IMPLIED>
<!ELEMENT otsikko (#PCDATA)>
<!ELEMENT asia (#PCDATA)>
<!ELEMENT tekija (#PCDATA)>
<!ELEMENT huomautus (#PCDATA)>
```

Yllä olevassa määrittelytiedostossa on käytetty värejä pelkästään asian havainnollistamiseksi, niillä ei ole mitään merkitystä varsinaisen tyyppimäärittelyn kannalta. Elementit ja attribuutit on määritelty erikseen ELEMENT- ja ATTLIST-termeillä. Tyyppimäärittely kuvaa XML-dokumentin rakennetta; ensimmäinen ”ELEMENT”-termi määrittelee, että dokumentin juurielementti on

”diplomityo”, joka koostuu lapsielementeistä ”otsikko”, ”asia”, ”tekijä” ja ”huomautus”. Kysymysmerkki (?) elementin perässä tarkoittaa valinnaista elementtiä eli sitä ei ole pakko olla dokumentissa, joten sen puuttuminen ei tee dokumentista epävalidia. Plusmerkki (+) tarkoittaa, että kyseinen elementti täytyy esiintyä dokumentissa vähintään kerran kyseisessä kohdassa eli diplomityo-elementin lapsielementtinä. Pakollisia diplomityo-elementin lapsielementtejä ovat tässä siis asia- ja tekijä-elementti, huomautus-elementti on valinnainen. ”ATTLIST”-termi määrittelee, että diplomityo-elementtiin kuuluu attribuutit ”pvm” ja ”sivulkm”. REQUIRED-termillä pvm on määritelty pakolliseksi attribuutiksi, IMPLIED-termillä on määritelty sivulkm- attribuutti valinnaiseksi. CDATA-termi (*Character Data*) tarkoittaa jäsentämätöntä merkkitietoa, PCDATA-termi (*Parsed Character Data*) tarkoittaa, että kyseinen elementti ei koostu jostain toisesta elementistä, vaan on jäsennettävää merkkitietoa (Eckstein 1999: 5–6; Lipitsäinen 2001).

Diplomityo.dtd -määrittelytiedostoon viitataan XML-dokumentissa seuraavasti:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE diplomityo SYSTEM "diplomityo.dtd">
<diplomityo pvm="26-01-2010">
  <otsikko>Yrityksen tietojärjestelmien integrointi</otsikko>
  <asia>EAI ja XML</asia>
  <tekija>Timo Hankkila</tekija>
  <huomautus>Värit havainnollistamisen vuoksi!</huomautus>
</diplomityo>
```

Yllä olevan XML-dokumentin kaksi ensimmäistä riviä kutsutaan esittelyosaksi, ja varsinainen data (esiintymä, *instance*) alkaa riviltä kolme. Esittelyosa kuvaa XML-dokumentin ja esittelee mahdollisen tyyppimäärittelyn. Ensimmäinen rivi

on pakollinen ja voi sisältää kolme eri attribuuttia: käytetyn XML-standardin versio (”1.0”; pakollinen attribuutti), merkkien koodaamisen esitystapa (”ISO-8859-1”; valinnainen attribuutti) sekä riippumattomuusjulistus eli selviääkö dokumentin validius dokumentin sisältämien tietojen perusteella (”no”; valinnainen attribuutti). Esittelyosan toinen rivi määrittelee dokumentin juurielementin sekä mahdollisen dokumentin rakennetta kuvaavan tyyppimäärittelyn. Tyyppimäärittely voi olla määritelty suoraan XML-dokumentin yhteyteen tai se voi olla erillään dokumentista omana tiedostonaan.

Esimerkki-dokumentissa juurielementti on ”diplomityo”, tyyppimäärittely on erillinen dokumentti, johon viitataan termillä ”SYSTEM” ja antamalla tyyppimäärittelytiedoston nimi ”diplomityo.dtd”. XML-dokumentti on loogiselta hierarkialtaan puurakenne, ja tyyppimäärittely validoi jokaisen juurielementin sisäisen elämentin (lapsielementit). Kyseinen XML-dokumentti on validi, sillä se on tyyppimäärittelyn mukainen vaikka esimerkiksi ”sivulkm”-attribuuttia ei esiinny diplomityo-elementissä, koska se on valinnainen attribuutti. (Eckstein 1999: 9–10; Microsoft Office Infopath; Lipitsäinen 2001).

XSL (*Extensible Stylesheet Language*) on W3C-suosituksiin kuuluva XML-kieliperhe, joka koostuu XML-kielistä, joiden avulla voidaan määritellä XML-dokumenttien rakennemuunnokset sekä esitystapa. XSL-kieliperheeseen kuuluu XSLT-muunnoskieli (*XSL Transformations*) XML-dokumenttien muuntamiseen, XPath-viittauskieli (*XML Path Language*) XML-dokumentin osien (polkujen) osoittamiseen sekä XSL-FO -merkintäkieli (*Extensible Stylesheet Language*

Formatting Objects), jonka avulla voidaan kuvata XML-dokumentin sivumuotoinen esitystapa yhdessä sisällön kanssa (World Wide Web Consortium 2009).

Esimerkki XML-dokumentin muokkaamisesta XSLT-kielen avulla:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons>
  <person username="tiha">
    <name>Timo</name>
    <family-name>Hankkila</family-name>
  </person>
  <person username="mame">
    <name>Matti</name>
    <family-name>Meikäläinen</family-name>
  </person>
</persons>
```

Yllä olevaa XML-dokumenttia halutaan muokata ja yksinkertaistaa siten, että sen pohjalta luodaan uusi dokumentti, johon poimitaan henkilötiedoista pelkät etunimet omiin elementteihin, ja attribuuteiksi määritellään kyseisen henkilön käyttäjätunnus. XML-dokumenttia ei tarvitse muokata manuaalisesti, vaan sen voi tehdä XSLT-kielen avulla, jolloin muokkaaminen on (oikein sovellettuna) huomattavasti nopeampaa sekä virheetöntä etenkin jos dataa olisi enemmän kuin tässä esimerkissä. Tämä XSLT-tyylidokumentti tarjoaa mallin, jonka mukaan XML-dokumenttia voidaan muokata toivotulla tavalla:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="xml" indent="yes" />
  <xsl:template match="/persons">
    <root>
```

```
      <xsl:apply-templates select="person" />
    </root>
  </xsl:template>
  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>
</xsl:stylesheet>
```

Kun alkuperäistä XML-dokumenttia prosessoidaan XSLT-tyylidokumentin avulla, lopputuloksena on uusi XML-dokumentti, jonka rakenne on halutun mukainen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
  <name username="tiha">Timo</name>
  <name username="mame">Matti</name>
</root>
```

3.3 Yrityksen tietojärjestelmien integrointiratkaisun toteutus

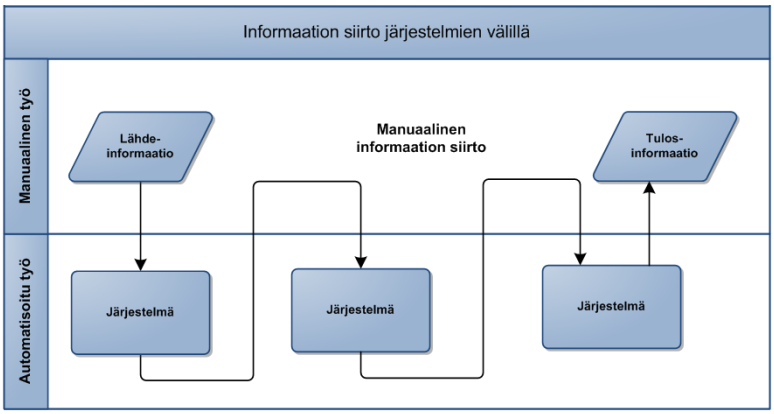
Tässä luvussa käsittelen erilaisia malleja yrityksen tietojärjestelmien integroinnin toteuttamiseen. Tarkastelussa ei syvennytä varsinaisesti mihinkään teknologiaan tai tuotteeseen vaan malleihin ja toimintatapoihin yleisellä (abstraktilla) tasolla. Lisäksi esittelen erilaisia integraatioarkkitehtuuriratkaisuja, integrointitopologioita.

3.3.1 Järjestelmäintegraation yleinen malli

Järjestelmäintegraatiossa on kysymys erityyppisten tietojärjestelmien vuoropuhelun automatisoinnista. Tyypillisin syy järjestelmäintegraation käyttöönottoon on hänen mukaansa se, että yritykset pyrkivät virtaviivaistamaan ja tehostamaan liiketoimintaprosessejaan tietoteknisiä ratkaisujaan tehostamalla. Yrityksien liiketoimintaprosesseihin liittyy paljon erilaista informaatiota, joita käsitellään useissa eri tietojärjestelmissä. Kun informaation jakaminen näiden järjestelmien välillä automatisoidaan integraatioratkaisun kautta, liiketoimintaprosessit nopeutuvat ja virheet vähenevät. Järjestelmäintegraation toteuttaminen vaatii kehittynyttä tapaa ajatella ja kokonaisuuden hahmottamista yksityiskohtiin keskittymisen sijaan (Tähtinen 2005: 22–23).

Tähtinen (2007: 24) on esitellyt abstraktin mallin järjestelmäintegraatioon, jonka tarkoituksena on vähentää manuaalityötä esimerkkitapauksessa, jossa yrityksellä on eri tietojärjestelmiä. Malli on esitelty yleisellä ja yksinkertaistetulla tasolla, jotta sitä voidaan soveltaa mahdollisimman moneen tapaukseen. Tässä esi-

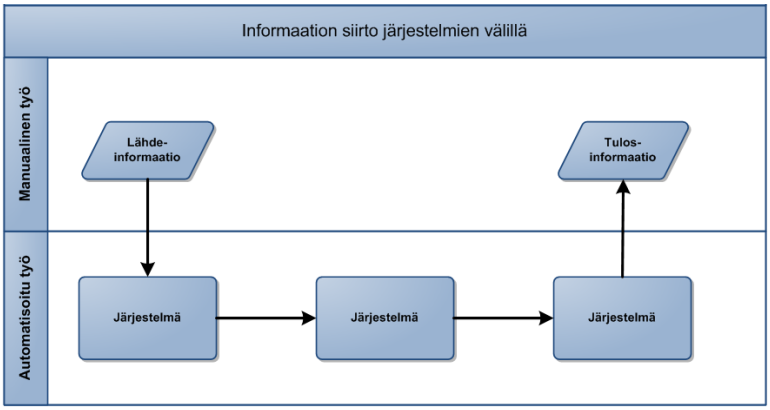
merkkinä on käytetty yritystä, jossa on kolme tietojärjestelmää: Taulukkolaskentajärjestelmä, asiakkuudenhallintajärjestelmä sekä päätöstukijärjestelmä.



Kuva 7. Informaation siirto järjestelmien välillä manuaalisesti (Tähtinen 2005: 23).

Kuvassa 7 on esitetty informaation (datan) kulkua eri järjestelmien välillä manuaalisena tiedonsiirtona. Automatisoitu tiedon prosessointi tapahtuu vain ja ainoastaan järjestelmien sisällä, mutta ei niiden välillä. Yrityksellä voi olla esimerkiksi huomattava määrä myyntidataa (lähdeinformaatio) taulukkolaskentajärjestelmässä (esimerkiksi Excel), joka halutaan siirtää asiakkuudenhallintajärjestelmään ja sieltä edelleen päätöstukijärjestelmään, jotta myyntidatasta voitaisiin tehdä raportteja yrityksen johdolle. Myyjien Exceliin syöttämä data on pelkkää raakadataa ilman tulkintaa eikä siis sovellu sellaisenaan päätöksenteon tueksi. Excel-data voidaan siirtää käsin kopioimalla asiakkuudenhallintajärjestelmään, jolloin jokainen myynti voidaan kohdistaa tietylle myyjälle ja määritellä myös asiakkaan tietoja. Jotta tätä dataa voidaan käyttää hyväksi päätöksenteossa, se joudutaan siirtämään manuaalisesti päätöksentekojärjestelmään, josta

voidaan tuottaa haluttuja raportteja esimerkiksi tietyille asiakkaille myydyistä tuotteita. Tätä dataa (tulosinformaatio) voidaan tulkita, ja yrityksen johto voi tehdä sen pohjalta päätöksiä. Ilman järjestelmäintegrointiratkaisua sama lähdeinformaatio joudutaan kuitenkin syöttämään manuaalisesti useaan eri järjestelmään. Tämä on virhealtista ja sitoo yrityksen resursseja. Pahimmassa tapauksessa tämä voi myös tulla yritykselle kalliiksi esimerkiksi laskutukseen liittyvien virheiden takia. Ongelma voidaan ratkaista soveltamalla yrityksen tietojärjestelmiin järjestelmäintegraatioratkaisua (Tähtinen 2005: 23–24).



Kuva 8. Informaation siirto järjestelmien välillä automaattisesti (Tähtinen 2005: 25).

Kuvassa 8 on esitetty järjestelmäintegraatioratkaisu, jossa informaation siirtäminen järjestelmien välillä tapahtuu automaattisesti. Ihmisen osuus on syöttää järjestelmään lähdeinformaatio (data) ja tulkita järjestelmien kautta syntynyt tulosinformaatio, jolloin sillä on jotain merkitystä yrityksen kannalta ja päätöksenteon tukena. Järjestelmäintegrointiratkaisun ansiosta lähdetieto syötetään järjestelmään vain kerran, minkä jälkeen integroidut järjestelmät jakavat lähde-

tiedon automaattisesti keskenään. Esimerkkitapauksessa Excel-data siirtyisi integrointiratkaisussa automaattisesti asiakkuudenhallintajärjestelmään, ja sieltä edelleen päätöstukijärjestelmään, josta voitaisiin ajaa raportteja automaattisesti tietyin parametrein. Ihminen osallistuu tähän prosessiin vain sen alussa syöttämällä lähdetiedot ja lopussa tulkitsemalla tulosinformaatiota – kaikki muu hoituu automaattisesti järjestelmäintegraation kautta. Ihmistä tarvitaan tässäkin ratkaisumallissa, mutta järjestelmät hoitavat manuaalisen työn ihmisen puolesta mahdollisimman nopeasti ja tehokkaasti. Pelkästään tiedon syöttämiseen kuluvan ajan väheneminen voi johtaa suoraan henkilötyön tehostumiseen. Automaattinen datan välittäminen integraatioratkaisun avulla järjestelmien välillä virheiden todennäköisyys pienenee ja data pysyy yhtenäisenä eri järjestelmien välillä (Tähtinen 2005: 24–25).

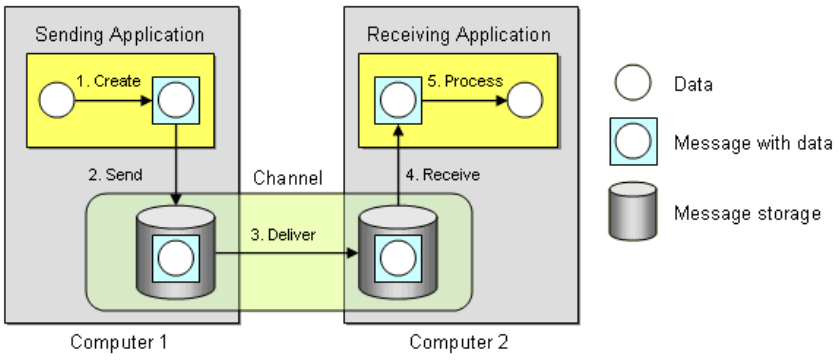
Tähtisen esittämässä integraatioratkaisumallissa liiketoimintaprosessien automatisointi ja informaation jakaminen eri ohjelmistojen tai järjestelmien välillä perusluonteeltaan on loogisten operaatioiden automatisointia, joka toteutetaan käytännössä ohjelmoimalla. Kyseisessä integrointiratkaisumallissa informaation käsittelytyötä siirretään mahdollisimman paljon käyttäjä-ohjelmisto-rajapinnan alapuolelle. Järjestelmäintegraatioratkaisun ansiosta yrityksen sovellusarkkitehtuurin hahmottaminen ja ylläpito helpottuu. Yrityksen sovelluksia ja niiden suorittamia tehtäviä voidaan ajatella kokonaisuutena, eräänlaisena ”supersovelluksena”, jonka ansiosta jokin toivottu toiminnallisuus ei välttämättä tarkoita ohjelmistojen päivittämistä, sillä haluttu ominaisuus voi löytyä nykyisistä käytössä olevista sovelluksista (Tähtinen 2005: 23–26).

3.3.2 Järjestelmäintegraation suunnittelumalleja

Kirjassa ”Enterprise Integration Patterns” (Hohpe & Woolf 2008) on esitetty erilaisia suunnittelumalleja yrityksen järjestelmäintegraatioon. Kirjassa esitetyt integrointimallit pohjautuvat viestien välittämiseen (*messaging*), ja ovat käytännössä hyväksi havaittuja ja testattuja aidoissa ympäristöissä. Nämä hyväksi havaitut integrointitavat ja -menetelmät ovat syntyneet oikeissa järjestelmäintegraatioprojekteissa vuosien aikana, ja näitä on koottu integrointimalleiksi, joita voi soveltaa omaan tapaukseensa ja ympäristöönsä. Nämä mallit ovat suunnittelumalleja jonkin integrointiongelman ratkaisuun – ne eivät mene syvälle teknisiin yksityiskohtiin, vaan pyrkivät esittämään sellaisen ratkaisun, joka ei ole sidottu tiettyyn tekniikkaan tai ohjelmistoympäristöön.

Järjestelmäintegrointia, joka perustuu viestien välittämiseen eri sovellusten välillä, voi verrata puhelimella soittamiseen. Synkroninen viestien välittäminen vastaa puhelimella soittamista – soittaja voi soittaa henkilölle puhelinverkon välityksellä, ja jos vastaanottaja vastaa puheluun, he voivat keskustella toistensa kanssa reaaliajassa ilman viiveitä. Asynkroninen viestien välittäminen vastaa viestin jättämistä vastaanottajan puhelinvastaajaan, mikäli henkilö ei syystä tai toisesta vastaa puheluun. Hän saa tiedon viestistä ja kuuntelee sen sitten, kun hänelle sopii ja soittaa kenties takaisin. Soittaja voi tällöin tehdä muita asioita luottaen siihen, että vastaanottaja saa viestin aikanaan. Sovellukset lähettävät dataa pakettimuodossa toisilleen verkon yli joko synkronisesti tai asynkronisesti. Näitä datapaketteja kutsutaan viesteiksi (*message*). Viestikanava (*message channel*) – kutsutaan joskus myös viestijonoksi – on looginen yhteys, eräänlainen polku, jonka kautta viestit löytävät perille, ja joka muodostetaan lähettäjän

ja vastaanottajan välille tietoverkossa. Lähettäjä (*sender, producer*) on sovellus, joka muodostaa viestin ja lähettää sen viestikanavaan. Vastaanottaja (*receiver, consumer*) on sovellus, joka vastaanottaa viestin lukemalla (ja poistamalla) sen viestikanavasta. Viesti voi olla esimerkiksi dataa kuten XML-dokumentti, kuvaus jostain tietystä komennosta, joka halutaan suorittaa vastaanottavassa sovelluksessa tai kuvaus tapahtumasta (*event*), joka esiintyi lähettävässä sovelluksessa. Viesti sisältää kaksi osaa: otsikon (*header*) ja sisällön (*body*). Viestin otsikko on metadataa varsinaisesta sisällöstä, kuten tietoa viestin lähettäjästä, vastaanottajasta ja niin edelleen. Viestin otsikko-osaa käyttää viestin välittävä järjestelmä, viestin vastaanottava sovellus ei yleensä sitä mihinkään käytä vaan ohittaa sen. Viestin sisältöosa on varsinainen data, mikä välitetään toiseen sovellukseen, viestin välittävä järjestelmä tyypillisesti ohittaa sen. Järjestelmäintegraation yhteydessä viestistä puhuttaessa viitataan yleensä tähän välitettävän viestin sisältöön, dataosaan (Hohpe & Woolf 2008: xxxi).



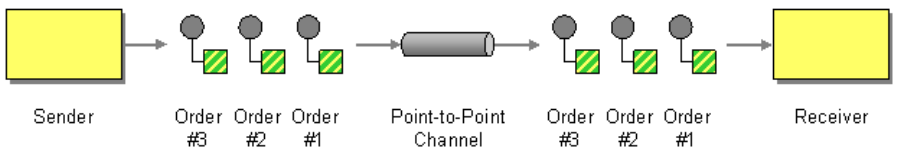
Kuva 9. Viestin välittämisen malli (Hohpe & Woolf 2008: xxxii).

Kuvassa 9 on esitetty malli viestin välittämiseen askel askeleelta. Ensimmäisessä vaiheessa (1) viesti luodaan: Lähettävä sovellus (tietokone 1) luo viestin sisäl-

täen otsikko- ja sisältöosan (data). Seuraavassa vaiheessa (2) viesti lähetetään: Lähettävä sovellus lisää viestin viestikanavaan. Kolmannessa vaiheessa (3) viesti välitetään: Viestinvälitysjärjestelmä siirtää viestin vastaanottavaan sovellukseen (tietokone 2) viestikanavan (*Channel*) kautta. Neljännessä vaiheessa (4) viesti vastaanotetaan: Vastaanottava sovellus lukee (ja poistaa) viestin viestikanavasta. Viidennessä vaiheessa (5) viesti käsitellään: Vastaanottava sovellus purkaa viestistä sen sisällön, dataosan (Hohpe & Woolf 2008: xxxii).

Järjestelmäintegrointiongelma: Miten voidaan lähettää viestejä usealle vastaanottajalle yhden viestikanavan kautta siten, että integraatio on mahdollisimman hyvin ja laadukkaasti toteutettu? Hohpe ja Woolf (2008: 508–509) ovat esitelleet integrointiratkaisumallin viestien välittämisen usealle vastaanottajalle keskitetyn viestinvälittäjän (*message dispatcher*) avulla. Tarkastellaan kuitenkin ensin yksinkertaista perustapausta, ja sen problematiikkaa.

Tarkastellaan järjestelmää, jossa sovellus A lähettää viestejä sovellukseen B. Viestit tulevat jonossa viestikanavaa pitkin sovellukseen B, joka käsittelee viestin ja suorittaa viestin mukaiset toiminnot.

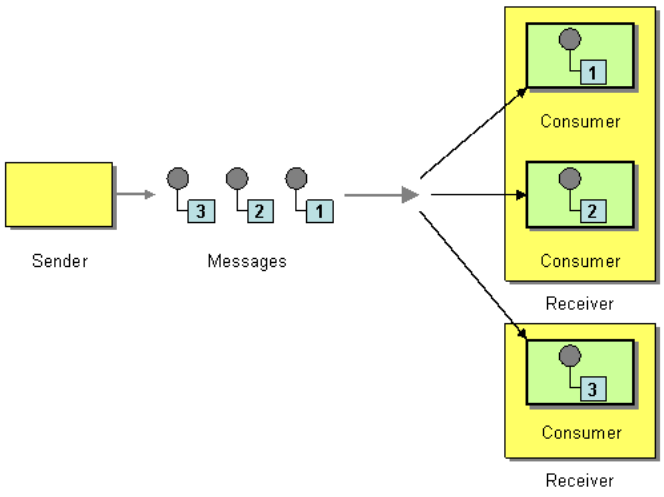


Kuva 10. Yksinkertainen viestinvälitysmalli (Hohpe & Woolf 2008: 103).

Kuvan 10 tapauksessa lähettäjäsovellus A (*Sender*) lähettää viestejä jonossa vastaanottajasovellukselle B (*Receiver*) kahdenvälistä viestikanavaa pitkin (*Point-to-Point Channel*). Tällä yksinkertaisella integraatioratkaisulla voidaan varmistaa, että vain yksi sovellus vastaanottaa ja käsittelee sovelluksen A lähettämän viestin. Vaikka vastaanottavia sovelluksia olisi useita, vain yksi niistä voi kerrallaan vastaanottaa lähetetyn viestin eli vastaanottavat sovellukset eivät voi kaikki vastaanottaa viestikanavaan lähetettyä tiettyä viestiä. Vastaanottajasovellusten ei tässä tapauksessa tarvitse mitenkään keskenään koordinoida viestien vastaanottamista, koska vain yksi niistä voi vastaanottaa lähetetyn viestin. Tämä tapaus on kuitenkin hyvin yksinkertainen eikä siis sellaisenaan sovellu järjestelmäintegraatioon muutoin kuin hyvin yksinkertaisissa tapauksissa.

Usean vastaanottajasovelluksen tapauksessa ongelmaksi muodostuu tyypillisesti se, että viestejä välittävä sovellus (*messaging system*) ei kykene prosessoimaan viestejä ja välittämään niitä usealle vastaanottajalle samaan tahtiin kuin se niitä vastaanottaa. Viestejä jonoutuu viestikanavaan, jolloin viestejä välittävästä sovelluksesta tulee järjestelmäintegraation kannalta pullonkaula, joka heikentää järjestelmän kokonaissuorituskykyä. Tämän ilmiön aiheutumiseen voi olla monta syytä: useat eri sovellukset voivat lähettää viestejä viestikanavaan samanaikaisesti, verkkoyhteyden katkokset voivat aiheuttaa ruuhkaa viestien lähettämiseen ja ne lähetetään kaikki kerralla viestikanavaan tai viestien prosessointi (käsittely viestejä välittävässä ja suorittaminen vastaanottavassa sovelluksessa) voi yksinkertaisesti kestää huomattavasti kauemmin kuin viestien lähettäminen viestikanavaan (Hohpe & Woolf 2008: 502).

Hophe ja Woolf (2008: 502) mukaan viestikanavaongelma voitaisiin ratkaista käyttämällä useita viestikanavia yhden sijaan, mutta edellä mainitut ongelmat voisivat siitä huolimatta kasautua yhteen viestikanavaan ja aiheuttaa viestien ruuhkautumisen eikä lähettäjäsovellus tietäisi, mitä vapaana olevia viestikanavia sen tulisi käyttää ruuhkautuneen sijaan. Useista viestikanavista olisi se etu, että ne mahdollistaisivat usean vastaanottajan käytön (yksi vastaanottajasovellus jokaista viestikanavaa kohden) ilman viestejä välittävää sovellusta, jolloin viestit voitaisiin käsitellä samanaikaisesti vastaanottajasovelluksissa. Vaikka ongelma onnistuttaisiinkin näin ratkaisemaan, välittäjäsovelluksen määrittelemä viestikanavien lukumäärä olisi järjestelmän kokonaissuorituskykyä rajoittava tekijä. Tarvitaan integraatioratkaisumalli, jossa yhtä (kahdenvälistä) viestikanavaa voi käyttää useat vastaanottajat samanaikaisesti.



Kuva 11. Usean vastaanottajan viestinvälitysmalli (Hohpe & Woolf 2008: 503).

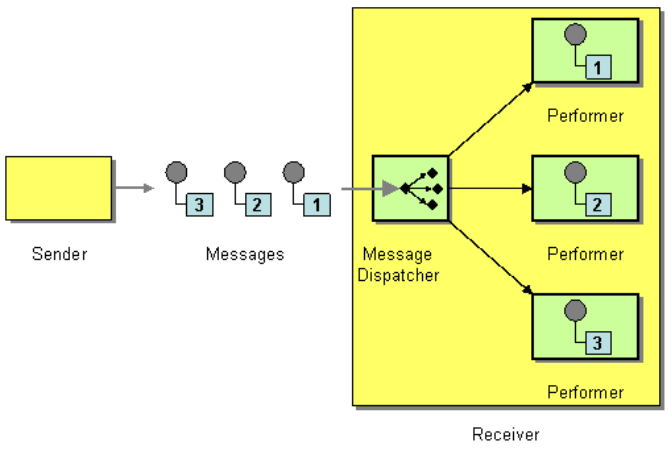
Kuvassa 11 on esitetty integraatioratkaisumalli viestinvälitykseen, jossa on useita vastaanottajasovelluksia. Nämä vastaanottajasovellukset (*Receiver*) ovat kes-

kenään samanarvoisia ja ne ”kilpailevat” keskenään siitä, mikä saapuvan viestin vastaanottaa. Yhdessä vastaanottajasovelluksessa voi olla useita viestin kuluttajia (*Consumer*). Tätä voisi kuvastaa siten, että viesti voidaan välittää useaan eri sovellukseen, ja sen voi sisältää käskyn tehdä useita eri asioita sovelluksessa, kuten esimerkiksi viestissä olevan datan tallentaminen tietokantaan tai XML-tiedostoon. Kilpailevat viestin kuluttajat (*competing consumers*) ovat luotu vastaanottamaan viestejä yhden kahdenvälisen viestikanavan kautta. Termi ”kilpailu” juontuu siitä, kun viesti välitetään viestikanavaan, mikä tahansa vastaanottajasovellus on viestin potentiaalinen vastaanottaja, ja tietyssä mielessä syntyy kilpailu viestin vastaanottamisesta ja kuluttamisesta (suorittamisesta). Viestinvälitysjärjestelmä määrittelee, mikä sovellus on viestin vastaanottaja ja mikä vastaanottajasovelluksen kuluttaja saa viestin käsiteltäväkseen. Kun viesti on vastaanotettu, viestin kuluttaja voi delegoida viestin käsittelyä ja suorittamista vastaanottajasovelluksen sisällä. Kukin viestin kuluttaja prosessoi eri viestiä rinnakkain, joten tämän integraatioratkaisun suorituskyky riippuu siitä, miten nopeasti viestinvälitysjärjestelmä kykenee välittämään viestejä vastaanottajille. Vastaanottajien määrää voidaan kasvattaa prosessointiresurssien (prosessori(e)n suorituskyky, muistin määrä, tallennuskapasiteetti) salliessa ilman, että siitä tulee integraatioratkaisun suorituskyvyn pullonkaula (Hohpe & Woolf 2008: 503–504).

Edellä kuvattu integraatioratkaisumalli tarjoaa ratkaisun aiemmin esitettyyn integraatio-ongelmaan, jossa tavoitteena oli voida lähettää viestejä usealle vastaanottajalle yhden viestikanavan kautta mahdollisimman laadukkaasti. Edellä kuvattu ratkaisu on hyvä malli perustapauksiin, mutta siinä on omat heikkoutensa. Viestien välittämisessä ja prosessoinnissa voi ilmetä konflikteja esimer-

kiksi sellaisessa tilanteessa, jossa useampi vastaanottaja syystä tai toisesta luulee olevansa viestin kuluttaja, ja siis yrittää käsitellä viestiä samanaikaisesti. Vähemmän laadukkaassa ratkaisussa sallitaan usean vastaanottajan ja kuluttajan yrittää vastaanottaa ja käsitellä samaa viestiä yhtä aikaa, mikä ei ole paras mahdollinen ratkaisu etenkin sellaisessa tapauksessa, jossa vastaanottajia ja viestiliikennettä on paljon (Hohpe & Woolf 2008: 504).

Hophe ja Woolf (2008: 504) kirjoittavat, että usean vastaanottajan viestinvälitysjärjestelmän laadukkaassa integraatioratkaisussa järjestelmä tunnistaa keskenään kilpailevat vastaanottajat ja kuluttajat, ja sisältää viestijakelijan. Tämä integraatioratkaisumalli auttaa ratkaisemaan ne ongelmat, jotka aiheutuvat tilanteesta, jossa usea vastaanottaja luulee olevansa viestin kuluttaja, ja yrittää yhtä aikaa käsitellä samaa viestiä.



Kuva 12. Usean vastaanottajan viestinvälitysmalli sisäisellä viestijakelijalla (Hohpe & Woolf 2008: 509).

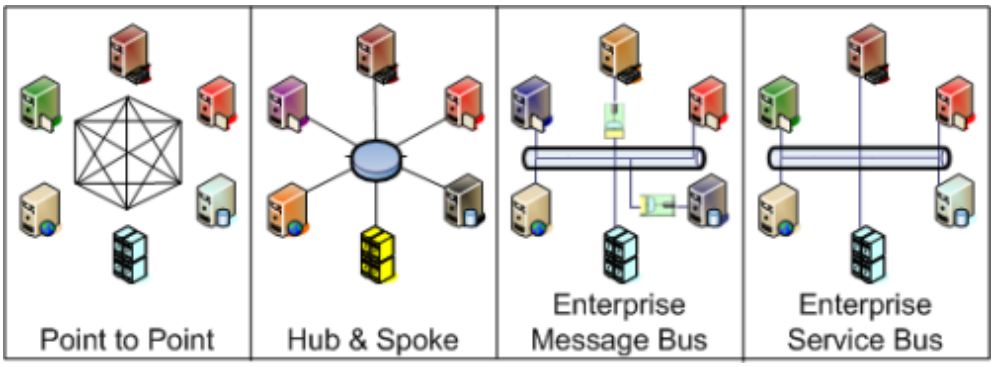
Kuvassa 12 esitetty integraatioratkaisumalli viestien välittämiseen usean vastaanottajan järjestelmässä, jossa käytetään yhtä viestikanavaa, jonka kautta viestit koordinoidaan ja prosessoidaan vastaanottajassa. Integraatioratkaisuun kuuluu kaksi osaa: viestijakelija (*Message Dispatcher*) ja viestin suorittaja (*Performer*). Viestijakelija on olio, joka kuluttaa viestit viestikanavasta saapumisjärjestyksessä, ja jakelee viestit eteenpäin viestin suorittajille. Viestin suorittaja on olio, jolle viesti välitetään viestijakelijan kautta, ja joka käsittelee viestin. Kun viestijakelija vastaanottaa viestin, se etsii sille sopivan suorittajan, ja välittää viestin suorittajalle, joka käsittelee viestin. Viestin suorittaja voi delegoida viestin käsittelyä sovelluksen sisällä, ja helpottaa sekä nopeuttaa viestin prosessointia. Viestin suorittaja voi olla viestijakelijan luoma uusi olio, tai se voi olla jokin aiemmin luoduista viestinsuorittajaolioista, joka ei sillä hetkellä käsittele mitään viestiä. Jokainen viestinsuorittajaolio voi toimia itsenäisesti, joten viestien suorittaminen voi tapahtua samanaikaisesti, rinnakkain. Jokainen suorittaja voi soveltua kaikenlaisten viestien suorittamiseen, tai tietty viestijakelija voi olla erikoistunut tietyn tyyppisten viestien käsittelemiseen ja suorittamiseen, jolloin viestijakelija voi välittää tietyt viestit soveltuvalle viestin suorittajalle viestin ominaisuuksien perusteella (Hohpe & Woolf 2008: 509–510).

Tässä integraatioratkaisussa viestijakelija toimii yhteytenä yhden viestikanavan ja usean viestin suorittajan välillä. Viestin suorittajat tekevät suurimman osa työstä; viestijakelija toimii ainoastaan välittäjänä viestien lähettäjän ja vastaanottajan, suorittajan välillä. Viestivälittäjä löytää jokaiselle viestille oikean ja vapaana olevan vastaanottajan, joka kykenee prosessoimaan ja suorittamaan viestin. Viestivälittäjä vastaanottaa viestin viestikanavasta, ja lähettää sen edelleen oikealle vastaanottajalle ilman, että se missään vaiheessa estää viestien kulkua

viestikanavassa. Viestien suorittajat voivat suorittaa viestejä niin nopeasti kuin viestivälittäjä kykenee niitä välittämään, riippumatta siitä miten kauan kunkin viestin prosessointiin menee aikaa. Tästä syystä viestit eivät ruuhkaudu viestikanavaan ja viestijakelijasta ei tule järjestelmän suorituskyvyn pullonkaula – viestit välitetään mahdollisimman tehokkaasti ja laadukkaasti oikealle vastaanottajalle, joka käsittelee ja suorittaa viestin (Hohpe & Woolf 2008: 511).

3.3.3 Järjestelmäintegraation arkkitehtuuri

Tietojärjestelmien integroinnissa on olemassa useita erilaisia teknologioita, ohjelmistoalustoja ja ympäristöjä, joilla integrointi voidaan toteuttaa. Järjestelmäintegroinnissa integraatioarkkitehtuuri voidaan jakaa kolmeen osaan, topologiaan: kahdenvälinen, monelta-monelle (tähti) ja väylä.



Kuva 13. Järjestelmäintegraation topologiat (Christudas 2007).

Kuvassa 13 on esitetty järjestelmäintegraation eri topologiat. Kahdenvälisessä (*point-to-point*) arkkitehtuurissa integrointiratkaisu määritellään kahden sovel-

luksen välille. Käytännössä tämä tarkoittaa sitä, että on olemassa kaksi integraatiopistettä eli sovellusta, jotka halutaan yhdistää toisiinsa eli integroida. Helpoin tapa (mikäli integroitavia sovelluksia on vähän) toteuttaa tämä käytännössä, on luoda dataformaattia muuttava sovitin tai protokolla joko vastaanottavaan tai lähettävään päähän (integraatiopisteeseen). Integroinnissa käytetään jotain tiettyä protokollaa tai ohjelmointirajapintaa, kuten esimerkiksi FTP (*File Transfer Protocol*), Microsoftin .NET tai Message Queuing –teknologiaa, tai eräajotyyppisiä käyttöliittymiä, jonka kautta sovellukset kommunikoivat keskenään ja joiden välityksellä dataa siirretään. Tällaisen kahdenvälisen integroinnin etu on se, että niiden välillä on niin sanottu tiukka kytkentä – molemmat osapuolet ovat tietoisia toisistaan ja ne ovat tiukasti toisiinsa sidottuja, mikä mahdollistaa tehokkaan tiedon ja sovelluslogiikan jakamisen. Tiukan kytkennän negatiivinen puoli on siinäkin, että se vaatii huomattavia muutoksia olemassa oleviin sovelluksiin ja tietokantoihin, ja kun yhteen järjestelmään tehdään muutoksia, täytyy ne tehdä myös kaikkiin muihin järjestelmiin (Christudas 2007).

Kahdenvälinen (pisteestä pisteeseen) integraatioarkkitehtuuri ei useinkaan sovellu tapauksiin, joissa halutaan integroida useita sovelluksia tai järjestelmiä. Kuvitellaan tilanne, jossa yritys haluaa integroida sovelluksensa siten, että jokaisen sovelluksen tulee voida kommunikoida keskenään. Yksinkertaistetaan asiaa siten, että yksi yhteys kykenee sekä lähettämään että vastaanottamaan dataa ja informaatiota (todellisuudessa tähän saatetaan tarvita kaksi erillistä yhteyttä). Kahdenvälisen integrointiratkaisun monimutkaisuutta tällaisessa tapauksessa kuvaa kaava, joka kertoo tarvittavien yhteyksien y lukumäärän:

$$y = \frac{n\,(n-1)}{2}, \qquad (1)$$

missä n on integroitavien järjestelmien tai sovellusten määrä. Jos integroitavien sovellusten lukumäärä on esimerkiksi kymmenen (n = 10), niiden integrointi kahdenvälisellä integraatioarkkitehtuurilla tarkoittaa neljääkymmentäviittä(!) yhteyttä sovellusten välillä. Tällaisen järjestelmäkokonaisuuden ylläpitäminen saattaa tulla yritykselle erittäin kalliiksi (Tähtinen 2005: 66).

Tähtimallin (monelta monelle) järjestelmäintegraatioarkkitehtuuri tarjoaa keskitetyn välittäjän (*broker*), johon jokainen sovellus on yhdistetty niin sanotulla löyhällä kytkennällä. Löyhän kytkennän etu on siinä, että integraatio voidaan toteuttaa hyvin pienillä muutoksia nykyisiin sovelluksiin. Integroituja sovelluksia voidaan myös vaihtaa toisiin huomattavasti pienemmällä vaivalla kuin tiukan kytkennän ratkaisussa – optimaalisessa tilanteessa ei tarvita mitään muutoksia itse sovelluksiin vaan välittäjä osaa kommunikoida suoraan uusien sovellusten kanssa. Viestien muutokset ja reititys tapahtuu välittäjässä (sanomanvälittäjä). Tähtitopologian integraatioarkkitehtuuri soveltuu parhaiten useita sovelluksia tai tietojärjestelmiä integroitaessa, koska tarvittavien yhteyksien lukumäärä vastaa suoraan integroitavien sovellusten lukumäärää. Tähtimallin integraatioratkaisun huonoin puoli on se, että mikäli keskitettyyn välittäjään tulee vika ja se lakkaa toimimasta, lamauttaa se koko integraation toiminnan, koska kaikki viestit kulkevat yhden välittäjän kautta. Tämä on kaikkien tähtimallin topologioiden suurin arkkitehtuurillinen heikkous ja riski (Motorola Inc. 2003: 5; Christudas 2007).

Väylätopologian järjestelmäintegraatioarkkitehtuuri tarjoaa yhteisen kommunikointi-infrastruktuurin, väylän, joka toimii alusta- ja ohjelmointikieliriippumattomana adapterina sovellusten välillä. Tämä kommunikaatioinfrastruktuuri voi sisältää sanomareitittäjä- ja/tai julkaise ja tilaa (*publish and subscribe*) -kanavia. Sovellukset ovat vuorovaikutuksessa keskenään sanomaväylän kautta vaadi vastaus (*request response*) -viestijonotekniikan avulla. Väyläarkkitehtuurissa viestit kulkevat aina jonossa (*message queue*), mutta se voi tapahtua joko synkronisesti tai asynkronisesti. Synkronisessa mallissa sovellus A lähettää viestin sovellukselle B, ja jää odottamaan vastausta sovellukselta B. Internetin HTTP-protokolla on tästä hyvä esimerkki: asiakas pyytää tiettyä web-sivua selaimen kautta, ja web-palvelin vastaa asiakkaan pyyntöön palauttamalla asiakkaan web-selaimeen oikean web-sivun. Asynkronisessa mallissa sovellus A voi esimerkiksi ilmoittaa jostain tapahtumasta sovellukselle B, mutta A:n ei tarvitse saada B:ltä minkäänlaista kuittausta tai paluuviestiä (tai ei tarvitse sitä välittömästi), jolloin A:n ei tarvitse jäädä odottamaan. Joissain tapauksissa sovellukset käyttävät adaptereita esimerkiksi tiettyjen skriptien hallintaan, jotka toimivat viestiliikenteen herätteenä. Tällaisilla adaptereilla voidaan luoda yhteys sovellusten ja väylän välille hyödyntämällä kaupallisten väyläsovellusten tai yhdistettävien sovellusten ohjelmointirajapintaa (API) (Christudas 2007; Linthicum 2000: 352, 358).

3.3.4 Järjestelmäintegraatioratkaisun toteuttamisen sudenkuopat

IT-ammattilaisille tarkoitetun Web-sivuston, ebizQ:n vuonna 2003 tehdyn selvityksen mukaan, noin 70 % järjestelmäintegraatiohankkeista epäonnistuu. Järjestelmäintegraatioprojektit epäonnistuvat, koska ne eivät valmistu aikataulussa, niiden budjetti ylittyy tai niiden tulokset eivät täytä niille asetettuja tavoitteita. Pahimmassa tapauksessa ne epäonnistuvat näissä kaikissa kolmessa kriteerissä. Selvityksen mukaan oli yllättävää, miten moni yritys joutuu tekemisiin samojen ongelmien kanssa järjestelmäintegraatioratkaisua implementoitaessa. Selvityksen tuloksena voitiin luetella muutamia pääsyitä siihen, minkä takia järjestelmäintegraatioprojektit epäonnistuvat (Trotta 2003):

Järjestelmäintegraatiossa muutos on pysyvää. Järjestelmäintegraatioratkaisut ovat hyvin dynaamisia, muuttuvat usein ja vaativat muutoksia useisiin järjestelmän osiin, komponentteihin. Järjestelmäintegraatioratkaisu koskettaa tyypillisesti yrityksen kaikkia liiketoimintaprosesseja yli organisaatiorajojen ja perinteisten arvoketjujen – alusta loppuun saakka. Perinteinen tietojärjestelmähankkeille tyypillinen budjetointi, joka päättyy tietyn projektin valmistumiseen aiheuttaa paljon ongelmia, kun sitä sovelletaan järjestelmäintegraatiohankkeisiin. Niissä palvelutasovaatimukset tyypillisesti kasvavat projektin implementoinnin jälkeen. Tämän vuoksi yritysten tulisi varautua projektin jälkeisiin investointikuluihin, jotka sallisivat muutakin kuin vain perustason ylläpidon (Trotta 2003).

Järjestelmäintegraation vaatima tietotaito on harvinaista. Järjestelmäintegraatio on monimutkaista, asynkronista ja rinnakkaista tiedon prosessointia, joka tapahtuu usein suljetun lähdekoodin sovelluksissa ja järjestelmissä. Tästä syystä erilaisten keskenään epäyhteensopivien dataformaattien yhteensovittaminen ja liiketoimintaprosessien jakaminen eri järjestelmien välillä voi olla erittäin vaativa ja haastava tehtävä ohjelmoijille ja järjestelmien ylläpidosta vastaaville henkilöille, jotka ovat tottuneet tarkastelemaan asioita yksinkertaisesti ja suoraviivaisesti. Yritysten pitäisi kuitenkin miettiä tarkoin kolmannen osapuolen valmistamien tuotteiden käyttämistä ja erityisesti pitää huolta tiedon jakamisesta sekä tietotaidon siirtymisestä yrityksen henkilöstön sisällä. Myös henkilöstön kielikysymyksiin ja muihin mahdollisiin tiedonkulun ja viestintään liittyviin ongelmiin tulisi kiinnittää huomiota (Trotta 2003).

Järjestelmäintegraation standardit eivät ole maailmanlaajuisia, yleispäteviä. Vaikka tämä saattaa vaikuttaa uskomattomalta, ohjelmistotoimittajat toisinaan poikkeavat standardeista sen vuoksi, etteivät eri tahojen määrittelemien standardien spesifikaatiot ole yksiselitteisiä. Tätä tapahtuu etenkin niiden standardien kohdalla, jotka ovat vielä kehitysasteella, mutta jotka ovat ajankohtaisia ja ovat hyvin suosittuja päivittäisessä käytössä, kuten esimerkiksi Web-palveluihin (*Web Services*) liittyvät standardit. Web-palveluihin liittyvää standardoinnin parissa työskentelee useita standardointiorganisaatioita, kuten W3C ja OASIS, ja niiden kehittämät standardit ovat keskenään yhä useammin ristiriidassa ja siis yhden standardin mukainen Web-palvelu ei ole välttämättä yhteensopiva toisen standardin mukaisen Web-palvelun kanssa, vaikka standardien pitäisi nimenomaan olla sitä varten, että niiden mukaiset teknologiat olisivat keskenään yhteensopivia. Tätä ongelmaa pahentaa usein se, että ohjelmisto-

toimittajat voivat olla hyvin vahvasti mukana tässä standardointityössä tavalla tai toisella, ja saattavat ajavat omia etujaan. Koska standardit eivät voi täysin taata yhteensopivuutta, yritysten tulisi järjestelmäintegraatiohankkeissa panostaa testaamiseen ja laadunvarmistukseen – vaikka integrointiratkaisut perustuisivatkin yleisiin standardeihin (Trotta 2003).

Järjestelmäintegraation väärinymmärtäminen. Järjestelmäintegraatio ymmärretään tyypillisesti yksittäisenä työkaluna, tekniikkana tai sovelluksena sen sijaan, että se nähtäisiin suurempana kokonaisuutena – toimintatapoina, filosofiana ja metodeina tukea yrityksen liiketoimintaa parhaalla mahdollisella tavalla tietojärjestelmien kautta. Tämä on hyvin tyypillinen ongelma järjestelmäintegraatiota käyttöönottavissa yrityksissä, mutta myös erittäin merkittävä hankkeen onnistumisen kannalta. IT-henkilöstö pyrkii tyypillisesti ratkaisemaan yrityksen IT-arkkitehtuurillisia ongelmia sekä liiketoiminnan vaatimuksia tietyn tekniikan ja esimerkiksi abstraktien rajapintamäärittelyjen avulla. Tällä tavoin kuitenkin keskitytään väärin asioihin, ja samalla unohdetaan järjestelmäintegraation tarkoitus: yhdistää yrityksen liiketoimintaprosessit ja liiketoimintakriittinen data (soveltamalla teknologiaa oikein).

Järjestelmäintegraatio voidaan ymmärtää myös väärin siten, ettei ymmärretä riittävästi sen liiketoimintakriittisyyttä ja että sen reaaliaikaisesta luonteesta johtuen, sitä voi olla vaikea hallita. Järjestelmäintegraatio voi myös vaikuttaa negatiivisesti muihin IT-projekteihin sekä yrityksen ja myös sen kumppaneiden palvelutasoon. Yrityksen IT-johdolta vaaditaan paljon kurinalaisuutta, oikeita työkaluja ja toimintatapoja, joita on mietitty ja suunniteltu etukäteen, jotta voidaan

hallita järjestelmäintegraatioon liittyviä tietoturvakysymyksiä, kapasiteetin hallintaa, kuormituksen tasapainotusta, muutosjohtamista ja seurantaa. Näiden toimeenpanemisesta vastaa yrityksen IT-johto, vaikka apuna käytettäisiin kumppaneita ja alihankintaa. Järjestelmäintegraatioasiantuntijoiden nimeäminen yrityksessä on hyvä alku, mutta ei ratkaise kaikkia ongelmia. Järjestelmäintegraatiosta täytyy jonkun vastata kokonaisuutena, ja yrityksen täytyy myös selvittää, kuka raportoi kenellekin järjestelmäintegraatioon liittyvissä asioissa (Trotta 2003).

Yksityiskohtien unohtaminen ajan myötä. Kun järjestelmäintegraatioratkaisu ajan kuluessa laajenee, tämän päivän merkityksetön informaatio saattaa olla huomenna kriittistä liiketoiminnan kannalta. Operatiiviset vaatimukset tyypillisesti muuttuvat sen jälkeen, kun järjestelmäintegraatioprojektissa on edetty projektinsuunnitteluvaihetta pidemmälle. Yksi ratkaisu tähän ongelmaan on se, että projektin edetessä dokumentoidaan kaikki tietojärjestelmien määrittelyt, datarakenteet, liittymät ja tietovuot. Näistä tulisi myös kerätä ja tallennetaan jonkinlaista statistiikkaa, jota voidaan tarvittaessa raportoida kaikille järjestelmäintegraatiohankkeeseen kuuluville osapuolille. Tällä tavoin voidaan ennalta ehkäistä monia ongelmia, jotka johtuvat siitä että kunnollista dokumentaatiota ei ole tehty alusta alkaen (Trotta 2003).

Vastuujako on epäselvä ja kokonaisuudenhallinta puuttuu. Järjestelmäintegraatiohankkeet ovat tyypillisesti hyvin laajoja ja monimutkaisia hankkeita, ja koskevat koko organisaatiota ja usein myös niiden sidosryhmiä, kuten asiakkaita. Tästä syystä hyvin helposti hämärtyy se, kuka on vastuussa järjestelmäinte-

graatiosta – järjestelmäintegraatioratkaisun toimittaja, yrityksen IT-hallinto vai kenties joku muu taho? Kun haastavan ja vaikean projektin läpiviemisen aikana ilmenee käytännössä aina lukuisia erilaisia ongelmia, olisi hyvä tehdä alusta alkaen selväksi kaikille hankkeen osapuolille, kuka vastaa mistäkin osa-alueesta, jotta aikaa ei mene turhaan vastuun vierittämiseen muille tahoille. Ongelmanratkaisu tulee olla jotenkin koordinoitua, jotta ongelmat voidaan ratkaista mahdollisimman tehokkaasti (Trotta 2003).

4. ESIMERKKITAPAUS: MYYNTIKONFIGURAATTORIN INTEGROINTI TOIMINNANOHJAUSJÄRJESTELMÄÄN

Tässä luvussa esittelen lyhyesti Summium-myyntikonfiguraattorin sekä sen integrointia MS Dynamics NAV -toiminnanohjausjärjestelmään. Kerron lyhyesti integrointiprojektista, tarkastelen kahta eri vaihtoehtoa integroinnin toteuttamiseen ja esittelen niiden hyviä ja huonoja puolia.

4.1 Summium-myyntikonfiguraattori

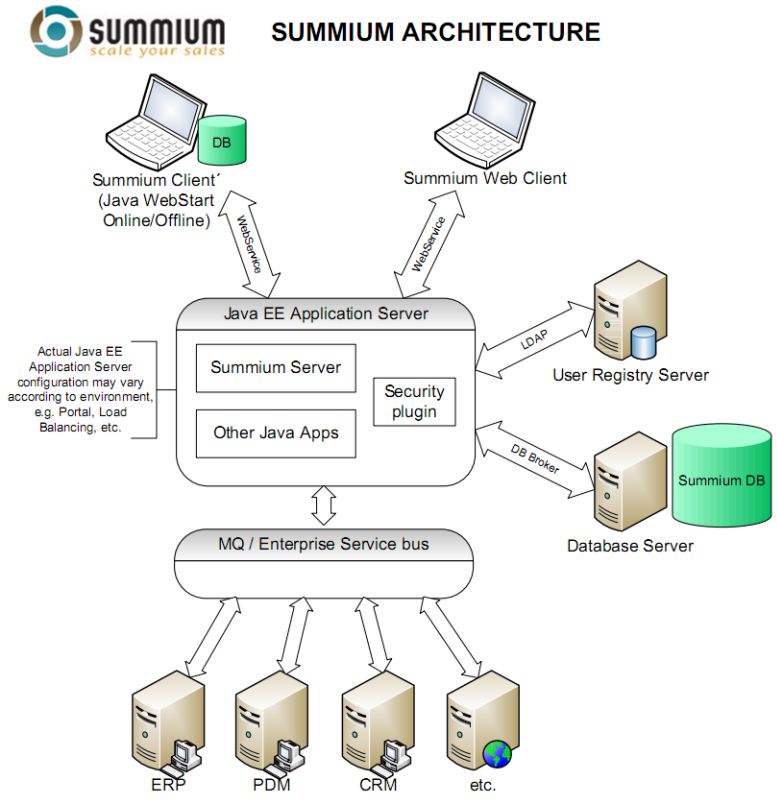
Konfiguraattori on sovellus, jolla hallitaan tuotteen rakennetta (myöhemmin käytetään käsitettä tuoterakenne) sekä sen eri variaatioita – hallitaan tuotteen erilaisia konfiguraatioita. Alan kirjallisuudessa erotellaan toisistaan kaksi eri käsitettä: myyntikonfiguraattori ja tuoterakennekonfiguraattori. Myyntikonfiguraattorilla kontrolloidaan tuotteen myyntiominaisuuksia sekä sääntöjä myyntiominaisuuksiin liittyen. Säännöt määrittelevät sallitut tuoteyhdistelmät (tuoterakenteen eri variaatiot) ja estävät kiellettyjen tuoteyhdistelmien tekemisen. Myyntikonfiguraattori on ensisijaisesti myyjän ja asiakkaan työkalu, jolla määritellään tuotteen ulkomuoto ja toiminnallisuus, josta asiakas on kiinnostunut. Tuoterakennekonfiguraattori on ensisijaisesti tuotteen valmistuksessa käytettävä työkalu, ja sillä määritellään tuotteen kokoonpanossa tarvittava dokumentaatio, kuten osaluettelo (Immonen & Sääksvuori 2002: 67–68).

Myyntikonfiguraattoria käytetään lähinnä massaräätälöitävien tuotteiden konfigurointiin. Tuotteen ominaisuudet määrittelevät sen, että minkälaisia tuotekonfiguraatioita halutaan sallia. Myyntikonfiguraattorin tarkoitus on auttaa ja nopeuttaa myyntityötä siten, että sen avulla myyjän on helppo räätälöidä kullekin asiakkaalle juuri heidän tarpeisiinsa sopiva tuote. Myyntikonfiguraattorin tarkoitus on myös estää myyjää tekemästä virheellisiä tai epätoivottuja kokoonpanoja asiakkaalle räätälöivästä tuotteesta.

Summium on Wapice Oy:n kehittämä myyntikonfiguraattori, jota käyttää yli 4000 käyttäjää maailmanlaajuisesti. Summium-myyntikonfiguraattori (myöhemmin käytetään termiä Summium tai myyntikonfiguraattori) on myyjän työkalu, jolla myyntivalintoja tekemällä konfiguroidaan massaräätälöitäviä tuotteita. Myyntivalintojen perusteella Summium muodostaa konfiguroidulle tuotteelle tarvittavan hintatiedon reaaliajassa sekä tuotteen valmistuksessa tarvittavan moduuli- tai tuoterakenteen tarvittavilla tiedoilla. Hintatiedot voivat muodostua myyntivalintojen, tuoterakenteen tai tuotteen muiden ominaisuuksien mukaan monipuolisten säännöstöjen avulla, jotka ovat käyttäjän hallittavissa (Wapice Oy 2009).

Summium-myyntikonfiguraattori muodostaa hintatietojen ja tuoterakenteen lisäksi tarvittavan kaupallisen ja teknisen asiakasdokumentaation sekä yrityksen sisäisen dokumentaation automaattisesti tehdyn tuotekonfiguraation perusteella. Summium voi integroitua muihin tietojärjestelmiin myynti- tai tilausprosessin eri vaiheista saadun informaation mukaan. Konfiguroinnin tuloksena syntyneen myyntivalintojen, tuoterakenteen, hintatietojen sekä teknisten tieto-

jen välinen logiikka ja säännöt hallitaan Summiumiin sisäänrakennetun sääntömoottorin avulla. Summiumin sääntömoottorin hyödyt korostuvat erityisesti, kun monimutkaisten sääntöjen hallittavuus vaikeutuu (Wapice Oy 2009).



Kuva 14. Summium-myyntikonfiguraattorin arkkitehtuuri (Wapice Oy 2009).

Kuvassa 14 on esitetty Summium-myyntikonfiguraattorin arkkitehtuuria. Myyntikonfiguraattoria voi käyttää joko Web-käyttöliittymän (*Summium Web Client*) kautta tai Java-sovelluksena (*Summium Java WebStart Client*) omalle koneelle asennettuna. Se voidaan myös räätälöidä osaksi asiakkaan Web-sivuja, jolloin konfiguraattorin ulkonäkö vastaa asiakasta. Summiumin arkkitehtuuri perustuu asiakas–palvelin arkkitehtuuriin, jossa kaikki data, kuten esimerkiksi

tarjoukset, tilaukset ja tuoterakenteet, tallennetaan Summium-palvelimella (J2EE-sovelluspalvelin, *Java Platform Enterprise Edition*) olevaan tietokantaan (*Database Server, Summium DataBase*). Palvelinohjelmisto ja tietokanta voi olla joko Wapicen ylläpitämä tai se voidaan asentaa asiakkaan ympäristöön. Tietokanta voi olla asiakasympäristöstä riippuen Microsoft SQL Server, MySQL, Oracle tai PostgreSQL.

Summiumia voi käyttää myös Offline-tilassa eli ilman verkkoyhteyttä, jolloin myyjä voi esimerkiksi valmistella asiakkaalle tarjouksen tuotteesta ja tallentaa tarjouksen omalle tietokoneelleen. Tarjous tallentuu automaattisesti Summium-tietokantaan, kun sovellus on yhteydessä Internetiin (Online-tila). Tällöin tiedot synkronisoidaan tietokannassa ajan tasalle. Offline-tilassa ei kuitenkaan ole käytettävissä kaikkea dataa, vaan vain pieni osa siitä, esimerkiksi joitain yksittäisiä tarjouksia. Riippumatta siitä, käytetäänkö Summiumia asiakassovelluksena vai Web-käyttöliittymän kautta, kaikki data tallentuu yhteen tietokantaan. Tällä varmistetaan se, että esimerkiksi Web-käyttöliittymän ja Java-asiakassovelluksen data on aina yhtenäistä. Vaikka tulevaisuudessa Summium-arkkitehtuuriin tehtäisiinkin muutoksia, kaikki data on tallennettuna yhdessä tietokannassa, joten esimerkiksi sovelluksen käyttöliittymään tai palvelinalustaan tehtävät muutokset eivät vaikuta varsinaiseen dataan. Summiumin sisäinen dataformaatti on XML, ja esimerkiksi tietokantaan tallennetut tuoteperheet (tuoterakenne) voidaan tallentaa myös XML-dokumentiksi suoraan sovelluksen käyttöliittymästä. XML-formaatti soveltuu erinomaisesti rakenteisen tiedon käsittelyyn (tuoterakenteiden hallinta), ja siitä on huomattavasti hyötyä silloin, kun myyntikonfiguraattorin dataa halutaan käyttää jossain toisessa sovelluksessa.

Summium-palvelinohjelmistoon sisältyy tietoturvasta huolehtiminen (*Security Plugin*) sekä käyttäjän tunnistaminen käyttäjätunnuksen ja salasanan avulla. Vaihtoehtoisesti käyttäjän tunnistaminen voidaan integroida osaksi asiakkaan omaa käyttäjätunnistusta, jolloin esimerkiksi asiakkaan LDAP-palvelin (*Light-weight Directory Access Protocol*) palauttaa tiedon, oliko annettu käyttäjätunnus ja salasana oikein, ja sen perusteella Summium joko hyväksyy tai hylkää käyttäjän kirjautumisen myyntikonfiguaattoriin. Palvelinohjelmistoon voi lisäksi kuulua muitakin Java-sovelluksia asiakkaan tarpeista riippuen. Palvelinohjelmiston vastuulla on datan tallentaminen tietokantaan, joka tapahtuu tietokantavälittäjän kautta (*DataBase broker*). Palvelinohjelmisto kommunikoi asiakassovelluksen kanssa kahteen suuntaan asiakas-palvelinarkkitehtuurin mukaisesti: Asiakas lähettää kyselyjä palvelimelle (HTTP-request), joka vastaa asiakkaan kyselyihin (HTTP-response) HTTP-protokollan kautta (Wapice Oy 2009).

Summium-myyntikonfiguraattori voidaan myös liittää asiakkaan olemassa oleviin tietojärjestelmiin, kuten esimerkiksi toiminnanohjaus-, tuotetiedonhallinta- tai asiakkuudenhallintajärjestelmään. Summium-palvelinohjelmisto voidaan integroida suoraan asiakkaan tietojärjestelmiin esimerkiksi sanomavälittäjätekniikalla viestijonoihin perustuen joko kahdenvälisellä tai väylätopologian arkkitehtuuriratkaisulla. Jos asiakkaalla on jo käytössä jokin järjestelmäintegraatioratkaisu, kuten esimerkiksi MS BizTalk, Summium voidaan liittää suoraan siihen, jolloin kommunikointi asiakkaan sovellusten kanssa tapahtuu järjestelmäintegraatio-ohjelmiston kautta.

4.2 Summium-integrointiprojekti

Summium-integrointiprojekti aloitettiin Wapicella alkuvuodesta 2009 integrointitiimissä. Tiimiin kuului projektipäällikkö ja kaksi teknistä asiantuntijaa minun lisäkseni. Jokaisella projektijäsenellä oli paljon muitakin tehtäviä eikä integrointiprojekti ollut kenenkään ykkösprioriteetti, mikä vaikutti projektin etenemiseen. Allekirjoittaneen rooli oli lähinnä projektin dokumentoinnissa diplomityöhön liittyen sekä osittain myös projektin etenemisestä huolehtiminen yhdessä projektipäällikön kanssa. Projektin tavoite oli integroida Summium-myyntikonfiguraattori Microsoftin Dynamics NAV -toiminnanohjausjärjestelmään (jatkossa käytetään termejä NAV, toiminnanohjausjärjestelmä). Vaikka projektissa asetettiin tavoitteeksi integroida Summium nimenomaan NAV-toiminnanohjausjärjestelmään, tausta-ajatuksena oli löytää riittävän geneerinen integrointiratkaisu, jota voitaisiin tarvittaessa hyödyntää missä tahansa asiakasprojektissa, jossa myyntikonfiguraattori halutaan integroida asiakkaan olemassa oleviin tietojärjestelmiin. Tällä tavoin integrointiratkaisun sovellettavuus tuleviin asiakasprojekteihin olisi mahdollisimman hyvä, ja onnistuessaan projektilla olisi myös pidemmän aikavälin positiivisia vaikutuksia.

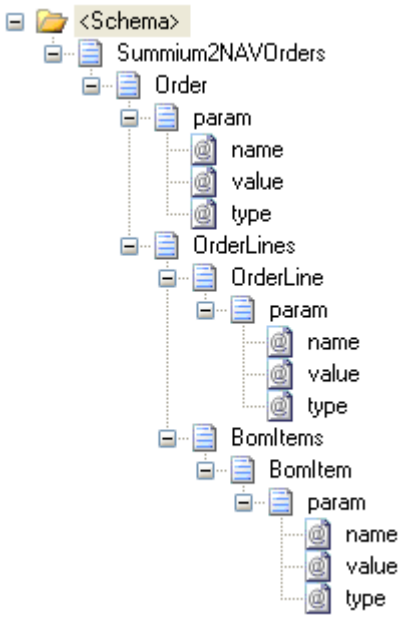
Myyntikonfiguraattorin ja toiminnanohjausjärjestelmän roolijako on se, että konfiguraattorissa tehdään kaikki tuotetta ja sen rakennetta koskevat määrittelyt (tuotteen konfigurointi, tuotevariaatioiden säännöt), ja sen perusteella tuotteelle muodostuu hinta sekä haluttu tuoterakenne (osaluettelo). Myyntikonfiguraattorin data siirretään toiminnanohjausjärjestelmään, ja toiminnanohjausjärjestelmä muodostaa tuoterakenteen perusteella osaluettelon halutun tuotteen

valmistamiseksi. Toiminnanohjausjärjestelmään voidaan määritellä osien valmistamiseen ja kokoonpanoon liittyviä asioita, kuten esimerkiksi osien hankinnan alihankkijoilta, valmistuksen vaiheistus (mitä tehdään missäkin järjestyksessä), aikatauluttaminen ja niin edelleen. Jotta tuote voidaan valmistaa, tuotteeseen kuuluvat osat täytyy olla toiminnanohjausjärjestelmässä tai ne täytyy sinne luoda myyntikonfiguraattorista tulevan datan perusteella.

Integrointiprojektissa päätettiin, että tehdään ensin yksinkertainen esimerkki konfiguroitavasta tuotteesta toiminnanohjausjärjestelmään, joka voidaan myöhemmin mallintaa myyntikonfiguraattorissa. Tämä päätettiin tehdä sen takia, että tiedetään, minkälainen sanoma NAV:iin täytyy tulla, jotta toiminnanohjausjärjestelmä kykenee tarvittaessa luomaan tuotteen osaluettelon Summiumista tulevan XML-sanoman perusteella.

Kun asiaa oli selvitetty ensin Dynamics NAV:issa, voitiin päättää yhdessä, minkälainen XML-sanoma Summiumista halutaan tuoda toiminnanohjausjärjestelmään. Summiumin XML-datassa on paljon ylimääräistä dataa, jota NAV ei tarvitse, ja joka voidaan poistaa, jolloin XML-sanomat ovat mahdollisimman yksinkertaisia ja selkeitä. XML-sanomaan päätettiin tehdä kolme tasoa (elementtiä): tilaustaso (Order), tilausrivitaso (OrderLine) ja tuoterakennetaso (BomItem, *Bill of Material Item*). Nämä kaikki sisältävät param-lapsielementin, jonka attribuutteja ovat name, value ja type, joihin tallennetaan parametrin nimi, arvo ja tyyppi. XML-sanoman rakenne haluttiin määritellä tällaiseksi sen vuoksi, ettei XML-skeemaan tarvitse tehdä muutoksia esimerkiksi asiakaskohtaisten vaatimusten takia (tietyn nimiset kentät). Tällä tavoin pyrittiin XML-sanoma pitä-

mään mahdollisimman geneerisenä, jolloin integraatioratkaisua on helpompi tarvittaessa muokata asiakastapauksen vaatimusten mukaan.



Kuva 15. XML-sanoman rakenne MS BizTalkissa (Wapice Oy 2009).

Kuvassa 15 on havainnollistettu Summiumista toiminnanohjausjärjestelmään lähetettävän XML-sanoman rakennetta. XML-sanoman tyyppimäärittely (XML-skeema) tulee olemaan tämän rakenteen mukainen. Jokainen Summiumista NAV:iin lähetettävä XML-sanoma täytyy vastata XML-skeemaa (ks. Liite 2), jotta toiminnanohjausjärjestelmä voi vastaanottaa ja käsitellä viestin. Kun tyyppimäärittelyn mukainen XML-sanoma on vastaanotettu toiminnanohjausjärjestelmässä, viesti käsitellään, ja tuotteen valmistus voi alkaa toiminnanohjausjärjestelmän prosessien mukaisesti.

Integrointiprojektin esimerkkitapauksessa päädyttiin käyttämään yksinkertaista polkupyörän kokoonpanoa (ks. Liite 1). Polkupyörässä on tietyt perusosat, kuten runko, pyörät ja ohjaustanko, mutta esimerkiksi pyöriä voi olla valittavissa useita erilaisia. Tuoterakenne ja säännöt, kuten sallitut tuotevariaatiot, ovat kaikki määritelty myyntikonfiguraattoriin, joten toiminnanohjausjärjestelmän ei tarvitse tietää, mitkä ovat sallittuja tuotevariaatioita. Myyntikonfiguraattorin sääntömoottori varmistaa sen, ettei toiminnanohjausjärjestelmään tule kiellettyjä tuoteyhdistelmiä.

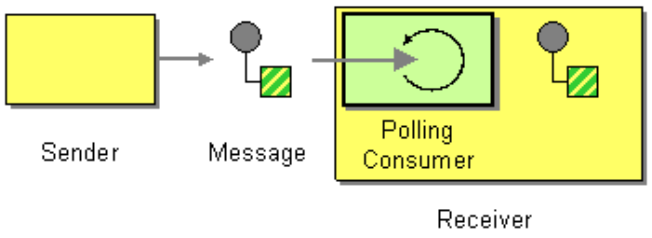
4.3 Ratkaisuehdotus integroinnin toteuttamismallista

Integrointitiimissä keskusteltiin erilaisista vaihtoehdoista, millä periaatetasolle integrointi tulisi toteuttaa. Kaksi vaihtoehtoa nousi keskusteluissa muiden ylitse. Tarkastelen edellisessä luvussa 4.2 kuvattua Summium-myyntikonfugiraattorin ja MS Dynamics NAV -toiminnanohjausjärjestelmän integrointiratkaisua kahden järjestelmäintegraation suunnittelumallin kautta, joista toinen on esitetty tarkemmin luvussa 3.3.2. Esittelen molempien vaihtoehtojen hyviä ja huonoja puolia, sekä oman ehdotukseni varsinaisesta integrointiratkaisusta.

Integrointiratkaisun vaihtoehto 1:

Kun myyntikonfiguraattorissa on tehty kaikki tarvittava tuotteen konfigurointiin liittyen, tuotedatasta voidaan muodostaa XML-sanoma toiminnanohjausjär-

jestelmää varten. Myyntikonfiguraattorin data tallennetaan ensin Summiumin omaan tietokantaan, kuten tehtäisiin ilman integraatioratkaisua. Tämän jälkeen Summium-tietokantaan tehdään kysely (*polling*), jotta voidaan tietää onko uutta dataa tallennettu tietokantaan tai olemassa olevaan dataan tehty muutoksia. Tämän jälkeen tietokannassa olevan datan perusteella muodostetaan luvussa 4.2 esitetyn XML-skeeman mukainen XML-sanoma toiminnanohjausjärjestelmään varten, ja lähetetään se NAV:iin, joka käsittelee viestin. Kun viesti on käsitelty, toiminnanohjausjärjestelmä voi aloittaa tuotteen valmistuksen toiminnanohjausjärjestelmän prosessien mukaisesti.



Kuva 16. Viestikyselyyn perustuva integrointiratkaisu (Hohpe & Woolf 2008: 450).

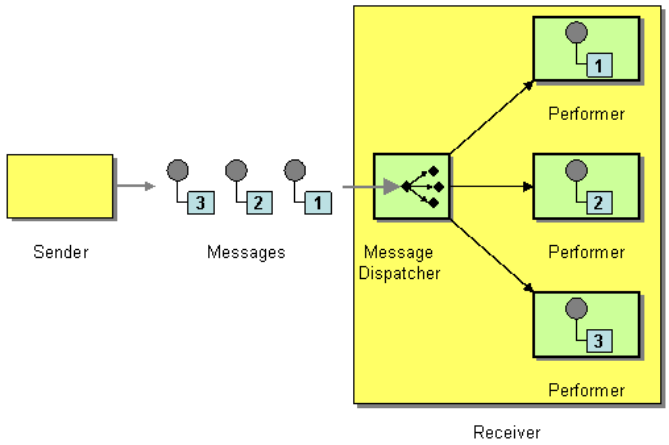
Kuvassa 16 on esitetty integrointiratkaisu, joka perustuu viestikyselyyn. Myyntikonfiguraattosissa (*Sender*) tehty tarjous tai tilaus (*Message*) tallennetaan ensin Summiumin tietokantaan. Vastaanottajasovellus (*Receiver*) tekee kyselyjä (*Polling Consumer*) Summium-tietokantaan, jotta konfiguraattorin kautta tietokantaan tallennetusta datasta voidaan tehdä XML-sanoma, joka käsitellään ja kulutetaan edelleen toiminnanohjausjärjestelmässä. Integrointiratkaisu ei voi tietää, milloin Summium-tietokantaan on dataa tallennettu, joten kyselyjä joudutaan tekemään usein, esimerkiksi ajastetusti tietyn ennalta määritellyt ajan välein.

Tämän integrointiratkaisun hyviä puolia on se, että data on varmasti ajan tasalla, koska kyselyt tehdään suoraan Summium-tietokannassa olevaan dataan, joka on tallennettu suoraan myyntikonfiguraattorista. Integrointiratkaisun huonoja puolia on se, että kyselyjä joudutaan tekemään jatkuvasti, koska ei tiedetä, milloin uutta dataa on tallennettu tietokantaan siellä jo ennestään olevan datan lisäksi. Tämä heikentää integrointiratkaisun suorituskykyä riippuen siitä, miten reaaliaikaista XML-sanoman siirtymisestä myyntikonfiguraattorista toiminnanohjausjärjestelmään halutaan. Datan siirto voitaisiin myös tehdä eräajotyyppisenä tiettynä ajankohtana, mutta tällöin tilauksen siirtyminen toiminnanohjausjärjestelmään olisi hidasta eikä se todennäköisesti täyttäisi asiakkaan liiketoiminnan vaatimuksia. Suorituskyky on tämän integrointiratkaisun huonoin puoli.

Integrointiratkaisun vaihtoehto 2:

Myyntikonfugiraattorin ja toiminnanohjausjärjestelmän integroinnissa voitaisiin soveltaa kappaleessa 3.3.2 esitettyä usean vastaanottajan viestinvälitysmallia sisäisellä viestijakelijalla (*Message Dispatcher*). Tämä ratkaisuvaihtoehto eroaa oleellisesti vaihtoehdosta 1 siinä, että XML-sanoma tallennetaan Summium tietokantaan ja otetaan samanaikaisesti käsittelyyn toiminnanohjausjärjestelmään. Datan käsittely on tällöin rinnakkaista eikä tietokantaan tarvitse tehdä kyselyjä, jotta tiedetään onko sinne tallennettu uutta dataa. Rinnakkainen datan käsittely on hyvin tehokasta, ja Summiumin XML-datasta muodostetaan samalla kertaa sekä NAV:iin menevä XML-skeeman mukainen XML-sanoma että tallennetaan tuotedata Summium-tietokantaan. Sisäinen viestijakelija on tässä integrointiratkaisussa se, joka vastaa XML-sanoman välittämisestä eri vastaanottajille (Performer), jotka käsittelevät ja kuluttavat viestin. Vastaanottajat voivat olla eri so-

velluksia, kuten esimerkiksi Summium-tietokanta tai toiminnanohjausjärjestelmä, tai ne voivat olla myös saman vastaanottajasovelluksen eri osia, jotka toimivat viestin suorittajina (*Performer*).



Kuva 17. Sisäiseen viestijakelijaan perustuva integrointiratkaisu (Hohpe & Woolf 2008: 509).

Kuvassa 17 esitetyssä integrointiratkaisussa Summium (*Sender*) lähettää XML-sanomia (*Messages*) yhtä viestikanavaa pitkin vastaanottajasovellukseen (*Receiver*), joka vastaa viestien koordinoinnista ja prosessoinnista. Vastaanottajasovelluksen sisäinen viestijakelija (*Message Dispatcher*) on olio, joka ottaa viestit käsittelyyn samassa järjestyksessä kuin ne saapuvat viestikanavaan, ja jakelee viestit eteenpäin viestin suorittajille (*Performer*). Viestin suorittajia voi olla Summium-tietokanta (tietokantaan tallentaminen tapahtuu Summium-arkkitehtuurissa DataBase Brokerin kautta), Dynamics NAV-toiminnanohjausjärjestelmä tai mikä tahansa muu asiakkaan tietojärjestelmä, esimerkiksi asiakkuudenhallintajärjestelmä tai jokin räätälöity liiketoimintasovellus. Viestin suorittajien lukumäärää voidaan tässä integrointiratkaisussa hel-

posti kasvattaa, kunhan integrointiratkaisu saadaan ensin toteutettua kahden suorittajan tapauksessa: Summium ja Dynamics NAV. Toiminnanohjausjärjestelmään tuleva XML-sanoma täytyy ensin muuttaa NAV:in hyväksymään muotoon luvussa 4.2 esitetyn XML-skeeman mukaisesti. Tämä XML-sanoman käsittelytyö voidaan antaa kyseisen viestin suorittajan tehtäväksi, jotta viestijakelija voi keskittyä ainoastaan viestin jakelemiseen eikä siitä tällöin tule järjestelmän suorituskyvyn kannalta pullonkaula jos se pysty vastaanottamaan ja jakelemaan viestejä tehokkaasti eteenpäin.

Tässä integrointiratkaisussa viestijakelija ja viestin suorittaja ovat olioita, jotka toimivat omissa säikeissään (*thread*), jotta ne voivat toimia itsenäisesti ja rinnakkain. Tämä on huomattavasti tehokkaampi ja laadukkaampi tapa koordinoida ja prosessoida viestejä kuin ensimmäisessä integrointiratkaisuvaihtoehdossa. Tällöin sanomat voidaan käsitellä mahdollisimman nopeasti ja rinnakkain ilman, että Summium-tietokantaan täytyy tehdä kyselyjä jatkuvasti eikä integrointiratkaisusta tule suorituskyvyn pullonkaula.

Edellä mainituista syistä johtuen, tämä on allekirjoittaneen ratkaisuehdotus Summium-myyntikonfiguraattorin integrointiratkaisuksi Dynamics NAV – toiminnanohjausjärjestelmään, ja myös kaikkiin muihin mahdollisiin asiakkaan tietojärjestelmiin, joihin myyntikonfiguraattori halutaan integroida. Toki integrointiratkaisun toteutuksessa vaaditaan ohjelmointitaitoa ja ongelmanratkaisukykyä, jotta sitä voitaisiin testata käytännössä. Integrointiratkaisun ohjelmointi voidaan toteuttaa jollain olio-ohjelmointikielellä, esimerkiksi .NET- tai Java-kielellä. Hohpe ja Woolf (2008: 513–514) ovat esittäneet yksinkertaisen ohjelma-

koodiesimerkin kyseisen integrointiratkaisun toteuttamisesta Java-ohjelmointikielellä (ks. Liite 3), jota voidaan soveltaa ja käyttää ohjelmointityön pohjana integrointiratkaisun toteuttamisessa.

Valitettavasti tätä käytännön työtä ei ehditty koskaan tehdä minun aikanani, joten tätä ratkaisuehdotusta ja sen soveltuvuutta tähän tapaukseen ei ole voitu testata. Allekirjoittaneen työsuhde Wapicella päättyi ennen kuin integrointiprojekti oli edennyt käytännön ohjelmointivaiheeseen. Se ei kuitenkaan tarkoita sitä, että tämän työn tulokset olisivat käyttökelvottomia – niitä ei vain ole toistaiseksi sovellettu käytäntöön. Ennen kuin riviäkään ohjelmakoodia on viisasta kirjoittaa, täytyy jokaisessa integrointiprojektissa nähdäkseni joka tapauksessa miettiä ja tutkia erilaisia vaihtoehtoja integroinnin toteuttamiseen. Niiden hyvistä ja huonoista puolista on hyvä olla tietoinen jo ennen kuin integrointiratkaisua aletaan toteuttaa ja ohjelmoida. Hyvä ja laadukas integrointiratkaisu voi parhaimmassa tapauksessa kantaa hedelmää myös tulevaisuudessa, huonosti valittu ja lyhytnäköinen ratkaisu saattaa pahimmillaan tulla erittäin kalliiksi vaikka se olisi toteutettu (ohjelmoitu) miten hyvin tahansa.

5. YHTEENVETO

Tässä tutkielmassa on esitelty tietojärjestelmien integroinnin taustoja, tarpeita ja ratkaisumalleja. Alun perin yrityksen sovellukset palvelivat vain yhtä tarkoitusta, mutta liiketoiminnan kasvaessa myös tietojärjestelmiin ja sovelluksiin kohdistuvat tarpeet ja vaatimukset kasvoivat. Yrityksillä on tänä päivänä aito tarve saada sovelluksensa toimimaan keskenään siten, ettei samoja asioita tarvitse tehdä moneen kertaan eri järjestelmissä ja että olemassa olevaa dataa ja informaatiota voidaan hyödyntää kaikissa yrityksen sovelluksissa.

Tämän diplomityön tavoite oli tarkastella yleisellä tasolla sitä, miten yrityksen tietojärjestelmien integrointi voitaisiin toteuttaa siten, että sovellukset voisivat kommunikoida keskenään, jakaa informaatiota ja liiketoimintalogiikkaa, jotta ne tukisivat yrityksen liiketoimintaa mahdollisimman hyvin. Olen tutkinut tietojärjestelmien integrointia kirjallisuuteen pohjautuen, tavoitteena löytää hyviä malleja ja kriteerejä integroinnin laadukkaaseen toteuttamiseen. Tutkielmassa olen keskittynyt kirjallisuudessa usein esiintyvään termiin EAI (*Enterprise Application Integration*) tietojärjestelmien integroinnin yhteydessä. Olen pyrkinyt selvittämään, mitä tähän käsitteeseen liittyy ja minkälaisia malleja yrityksen tietojärjestelmien integrointiin on olemassa.

Yritysten tietojärjestelmien integrointi on aiheena laaja ja vielä melko uusi, joten se on aiheuttanut omia haasteita aiheen käsittelyssä. Alan kirjallisuutta lukemalla ja tutkimalla eri lähteitä olen ymmärtänyt, ettei ole olemassa mitään ”hopealuotiratkaisua” tietojärjestelmien integrointiin. Järjestelmäintegraatio pyrkii

auttamaan yrityksen IT-arkkitehtuurista vastaavia ihmisiä suunnittelemaan ja implementoimaan tietojärjestelmät sellaisiksi, että ne olisi mahdollisimman helppoa integroida muihin järjestelmiin ja sovelluksiin, jotta informaatiota ja liiketoimintaprosesseja voidaan hyödyntää eri sovelluksissa koko organisaatiossa. Yrityksen johdon näkökulmasta tämä tarkoittaa käytännössä sitä, että tietojärjestelmät tukevat yrityksen liiketoimintaa mahdollisimman hyvin ja johdolla on saatavilla oikea informaatio päätöksenteon tueksi.

Yrityksillä on usein käytössä historian myötä lukuisia perinnejärjestelmiä (*legacy system*), joiden integrointi muihin järjestelmiin vaatii jonkinlaista integrointiratkaisua. Järjestelmäintegraatio tarjoaa tähän erilaisia integrointimalleja ja -tekniikoita, joita yritys voi soveltaa omaan ympäristöönsä. Esimerkiksi kahdenvälinen integrointiratkaisu (*point-to-point*) saattaa olla hyvä ratkaisu kahden tai kolmen integroitavan sovelluksen tapauksessa, mutta se ei sovellu monimutkaisiin ympäristöihin, joissa on useita eri sovelluksia. Tällaisiin tapauksiin soveltuu parhaiten tähtimallin (monelta monelle) integrointiratkaisu, koska sovellukset ovat yhteydessä toisiinsa yhden integrointipisteen, keskitetyn välittäjän (*Broker*) kautta, jolloin integraatioarkkitehtuuri pysyy hallittavana ja yksinkertaisena. Tämän arkkitehtuuriratkaisun huonoja puolia on se, että jos välittäjä on jostain syystä epäkunnossa tai pois käytöstä, koko integraatio ei toimi eli yksikään viesti ei kulje sovelluksesta toiseen. Yrityksen tietojärjestelmäpäälliköiden ja asiantuntijoiden tehtäväksi jää soveltaa järjestelmäintegraatiota heidän toimintaympäristöönsä siten, että se palvelee yrityksen liiketoimintaa parhaalla mahdollisella tavalla.

Nykyään lähes jokaisen yrityksen tavoite on kasvattaa liiketoimintaa, ja kun liiketoiminta tämän seurauksena monimutkaistuu, kansainvälistyy ja erilaisten liiketoimintasovellusten määrä kasvaa, yrityksen on ennemmin tai myöhemmin lähestulkoon välttämätöntä panostaa järjestelmäintegraatioratkaisuun. Järjestelmäintegraatioprojektit alkavat usein hyvin yksinkertaisista kahden sovelluksen välisistä integroinneista, joista kuitenkin tulee nopeasti hyvin hankalia hallita ja ylläpitää integroitavien sovellusten määrän kasvaessa. Jossain vaiheessa yrityksen johdon täytyy ottaa kantaa siihen, voidaanko liiketoimintaa ja sen kasvattamista jatkaa ilman, että panostetaan järjestelmäintegraatioratkaisuun sen haitoista huolimatta ja että onko järjestelmäintegraatioratkaisusta enemmän hyötyä yrityksen liiketoiminnalle kuin siitä on haittaa, ettei siihen panosteta. Nämä ovat tapauskohtaisia asioita, eikä näihin ole yksiselitteistä ja suoraa vastausta, vaan vaativat jokaisen tapauksen tarkastelua kokonaisvaltaisesti. Tämän tutkielman puitteissa olen pyrkinyt tähän ajankohtaiseen ja hyvin mielenkiintoiseen asiaan tuomaan edes vähän omaa näkemystäni.

Summium-myyntikonfiguraattorin integrointi toiminnanohjausjärjestelmään oli mielenkiintoinen käytännön esimerkki järjestelmäintegrointiratkaisusta. Valitettavasti allekirjoittaneen osalta integrointiprojekti jäi kesken, kun määräaikainen työsuhteeni Wapice Oy:ssä päättyi vuoden 2009 lopussa. Tässä tutkielmassa esitetty integrointiratkaisu on tehty niiden tietojen pohjalta, mitä minulla on ollut käytettävissä edellä mainittu asia huomioiden. Ratkaisuehdotus on tästä syystä esitetty hyvin yleisellä tasolla eikä sitä ole voitu toteuttaa ja testata käytännössä. Tämä luonnollisesti vähentää tutkielman käyttöarvoa, mutta olen pyrkinyt kuitenkin löytämään ja esittämään järjestelmäintegrointiratkaisuja riittävän yleisellä tasolla (arkkitehtuuritasolla), joita voidaan soveltaa tapauskohtaisesti.

LÄHDELUETTELO

Chandra, David; Liu, Anna; Roxburgh, Ulrich; Mason, Andrew; Nadhan, E.G. & Slater, Paul (2003). *Guidelines for Application Integration*. Microsoft Patterns & Practices. 140 sivua. Saatavana World Wide Webissä: <URL: http://www.microsoft.com/downloads/details.aspx?FamilyId=3F182469-529E-4472-95E8-151E0CC8D3A5&displaylang=en>.

Christudas, Binildas A. (2007). *Aggregate Services in ServiceMix JBI ESB*. Packt Publishers. [online] [siteerattu 10.09.2009]. Saatavana World Wide Webissä: <URL: http://www.packtpub.com/article/aggregate-services-in-servicemix-jbi-esb>.

Eckstein, Robert (1999). *XML Pocket Reference*. Sebastopol: O'Reilly & Associates. 107 sivua. ISBN: 1-56592-709-5.

Gable, Julie (2002). *Enterprise Application Integration*. Information Management Journal. [online] [siteerattu 16.02.2010]. Saatavana World Wide Webissä: <URL: http://findarticles.com/p/articles/mi_qa3937/is_200203/ai_n9019202/>.

Haikala, Ilkka & Märijärvi, Jukka (2006). *Ohjelmistotuotanto*. Jyväskylä: Gummerus Kirjapaino Oy. 440 sivua. ISBN: 952-14-0850-2.

Hohpe, Gregor; Woolf, Bobby (2008). *Enterprise Integration Patterns.* Massachusetts: Addison-Wesley. 683 sivua. ISBN: 0-321-20068-3.

Holman, Ken G. (2002). *Definitive XSLT and XPath.* New Jersey: Prentice Hall PTR. 373 sivua. ISBN: 0-13-065196-6.

Kroenke, David M. (2007). *Using MIS.* New Jersey: Pearson Prentice Hall. 395 sivua. ISBN: 0-13-143372-5.

Laine, Harri (2001). *Ohjelmistoarkkitehtuurien luentomateriaali.* Helsingin yliopisto. [online] [siteerattu 11.10.2009]. Saatavana World Wide Webissä: <URL: http://www.cs.helsinki.fi/u/laine/arkki/k01/ark01_7.pdf>.

Linthicum, David S. (2001). *B2B Application Integration: e-Business – Enable Your Enterprise.* Massachusetts: Addison-Wesley. 408 sivua. ISBN: 0-201-70936-8.

Linthicum, David S. (2000). *Enterprise Application Integration.* Massachusetts: Addison-Wesley. 375 sivua. ISBN: 0-201-61583-5.

Lipitsäinen, Arvo (2001). *XML - Extensible Markup Language*. Helsigin liiketalouden ammattikorkeakoulu. [online] [siteerattu 29.01.2010]. Saatavana

World Wide Webissä: <URL:
http://myy.helia.fi/~atk42d/Xml/xmlopas#johdanto>.

McLeod, Raymond Jr. & George P. Schell (2007). *Management Information Systems – Tenth Edition.* New Jersey: Pearson Prentice Hall. 447 sivua. ISBN: 0-13-188918-4.

Microsoft Oy (2009). *Microsoft Dynamics NAV 5.0 -myyntikalvot.* 75 sivua.

Microsoft Office Infopath (2010). *Tietoja XML:stä*. [online] [siteerattu 29.01.2010]. Saatavana World Wide Webissä: <URL:
http://office.microsoft.com/fi-fi/infopath/HP010967281035.aspx>.

Motorola Inc. (2003). *ENTERPRISE INTEGRATION: A Key To Successful Public Sector Customer Service – White Paper*. 12 sivua. Saatavana World Wide Webissä: <URL:
http://www.motorola.com/governmentandenterprise/contentdir/en_US/Files/SolutionInformation/Enterprise_Integration_White_Paper.pdf>.

Pohjonen, Risto (2002). *Tietojärjestelmien kehittäminen.* Jyväskylä: Docendo Finland Oy. 178 sivua. ISBN: 951-846-146-5.

Sääksvuori, Antti & Anselmi Immonen (2004). *Product Lifecycle Management.* New York: Springer. 222 sivua. ISBN: 3-540-40373-6.

Tietotekniikan liitto ry (2004). *ATK-sanakirja.* Jyväskylä: Talentum Media Oy. 720 sivua. ISBN: 952-14-0869-3.

Trotta, Gian (2003). *Dancing Around EAI 'Bear Traps'*. EbizQ community. [online] [siteerattu 16.03.2010]. Saatavana World Wide Webissä: <URL: http://www.ebizq.net/topics/int_sbp/features/3463.html>.

Tähtinen, Sami (2005). *Järjestelmäintegraatio. Tarve, vaihtoehdot, toteutus.* Jyväskylä: Gummerus Kirjapaino Oy. 217 sivua. ISBN: 952-14-0854-5.

W3C Suomen toimisto. *XML 10 kohdan tiivistelmänä*. [online] [siteerattu 10.12.2009]. Saatavana World Wide Webissä: <URL: http://www.w3c.tut.fi/translations/xml/xmlin10pts/>.

Wapice Oy (2009). *Summium-myyntikonfiguraattorin kotisivut*. [online] [siteerattu 9.12.2009]. Saatavana World Wide Webissä: <URL: http://www.summium.com/fi/summium.html>.

World Wide Web Consortium 2009. *Extensible Markup Language (XML) 1.0 (Fifth Edition) – W3C Recommendation 26 November 2008*. [online] [siteerattu 11.12.2009]. Saatavana World Wide Webissä: <URL: http://www.w3.org/TR/2008/REC-xml-20081126/>.

LIITTEET

LIITE 1. Polkupyörän nimikeluettelo toiminnanohjausjärjestelmää varten.

Nimikeluettelo

Nro	Kuvaus	Korvaavia olemassa	Tuotantokenne	Tuotannon tuoteraken teen nro	Reititys nro	Perusmitta yksikkö	Kustannusta on muutettu	Yksikkökustannus	Yksikköhinta	Toimittajan nro	Hakunimi
1000	Polkupyörä	Ei	Ei	1000	1000	KPL	Ei	350,60	4 000,00		POLKUPYÖRÄ
1001	Retkipyörä	Ei	Ei	1000	1000	KPL	Kyllä	350,60	4 000,00		RETKIPYÖRÄ
1100	Etupyörä	Ei	Ei	1100	1100	KPL	Ei	129,67	1 000,00	20000	ETUPYÖRÄ
1110	Vanne	Ei	Ei			KPL	Kyllä	1,05	0,00	1587796	VANNE
1120	Pinnat	Ei	Ei			KPL	Kyllä	2,00	0,00	1587796	PINNAT
1150	Etukeskiö	Ei	Ei	1150	1150	KPL	Kyllä	12,44	500,00		ETUKESKIÖ
1151	Eturenkaan akseli	Ei	Ei			KPL	Kyllä	0,45	0,00	32456123	ETURENKAAN AKSELI
1155	Etuistukka	Ei	Ei			KPL	Kyllä	0,77	0,00	32456123	ETUISTUKKA
1160	Rengas	Ei	Ei			KPL	Kyllä	1,23	0,00	1587796	RENGAS
1170	Putki	Ei	Ei			KPL	Kyllä	1,75	0,00	1587796	PUTKI
1200	Takarengas	Ei	Ei	1200	1200	KPL	Ei	129,68	1 200,00		TAKARENGAS
1250	Takakeskiö	Ei	Ei	1250	1150	KPL	Kyllä	12,45	1 100,00		TAKAKESKIÖ
1251	Takarenkaan akseli	Ei	Ei			KPL	Kyllä	0,33	0,00	1587796	TAKARENKAAN AKSELI
1255	Takaistukka	Ei	Ei			KPL	Kyllä	0,90	0,00	1587796	TAKAISTUKKA
1300	Ketjun kokoonpano	Ei	Ei	1300		KPL	Ei	13,16	800,00		KETJUN KOKOONPANO
1310	Ketju	Ei	Ei			KPL	Kyllä	1,99	0,00	32456123	KETJU
1320	Etupyörän ketju	Ei	Ei			KPL	Kyllä	4,66	0,00	32456123	ETUPYÖRÄN KETJU
1330	Takapyörän ketju	Ei	Ei			KPL	Kyllä	5,88	0,00	32456123	TAKAPYÖRÄN KETJU
1400	Etulokasuoja	Ei	Ei			KPL	Ei	3,90	0,00	32456123	ETULOKASUOJA
1450	Takalokasuoja	Ei	Ei			KPL	Ei	3,90	0,00	32456123	TAKALOKASUOJA
1500	Lamppu	Ei	Ei			KPL	Ei	5,20	0,00	45774477	LAMPPU
1600	Kello	Ei	Ei			KPL	Ei	2,70	0,00	32456123	KELLO
1700	Jarrut	Ei	Ei	1700		KPL	Ei	9,77	600,00		JARRUT
1710	Käsitakajarru	Ei	Ei			KPL	Kyllä	4,50	0,00	32456123	KÄSITAKAJARRU
1720	Käsietujarru	Ei	Ei			KPL	Kyllä	4,80	0,00	1587796	KÄSIETUJARRU
1800	Ohjaustanko	Ei	Ei			KPL	Ei	2,12	0,00	1587796	OHJAUSTANKO
1850	Satula	Ei	Ei			KPL	Ei	7,20	0,00	1587796	SATULA

LIITE 2. XML-skeema Summiumista toiminnanohjausjärjestelmään lähetettäviä XML-sanomia varten.

```
<?xml version="1.0" encoding="utf-16" ?>
- <xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003" xmlns="http://BizTalk_Server_Project1.Schema1" targetNamespace="http://BizTalk_Server_Project1.Schema1"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  - <xs:element name="Summium2NAVOrders">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="Order">
          - <xs:complexType>
            - <xs:sequence>
              - <xs:element name="param">
                - <xs:complexType>
                    <xs:attribute name="name" type="xs:string" />
                    <xs:attribute name="value" type="xs:string" />
                    <xs:attribute name="type" type="xs:string" />
                  </xs:complexType>
                </xs:element>
              - <xs:element name="OrderLines">
                - <xs:complexType>
                  - <xs:sequence>
                    - <xs:element name="OrderLine">
                      - <xs:complexType>
                        - <xs:sequence>
                          - <xs:element name="param">
                            - <xs:complexType>
                                <xs:attribute name="name" type="xs:string" />
                                <xs:attribute name="value" type="xs:string" />
                                <xs:attribute name="type" type="xs:string" />
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                        </xs:complexType>
                      </xs:element>
                    - <xs:element name="BomItems">
                      + <xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

LIITE 3. Java-ohjelmakoodiesimerkki Summium-myyntikonfiguraattorin ja MS Dynamics NAV -toiminnanohjausjärjestelmän integrointiratkaisuehdotukseen (sisäinen viestijakelija).

Example: *Simple Java Dispatcher*

This is a simple example of how to implement a dispatcher and performer in Java. A more sophisticated dispatcher implementation might pool several performers, keep track of which ones are currently available to process messages, and make use of a thread pool. This simple example skips those details but does run each performer in its own thread so that they can run concurrently.

The driver/manager that controls the dispatcher (not shown) will run receiveSync() repeatedly. Each time, the dispatcher will receive() the next message, instantiate a new performer instance to process the message, and then start the performer in its own thread.

```
import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import javax.naming.NamingException;

public class MessageDispatcher {

    MessageConsumer consumer;
    int nextID = 1;

    protected MessageDispatcher() {
        super();
    }

    public static MessageDispatcher newDispatcher(Connection connection,
      String queueName)
        throws JMSException, NamingException {
        MessageDispatcher dispatcher = new MessageDispatcher();
        dispatcher.initialize(connection, queueName);
        return dispatcher;
    }

    protected void initialize(Connection connection, String queueName)
        throws JMSException, NamingException {
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Destination dispatcherQueue = JndiUtil.getDestination(queueName);
        consumer = session.createConsumer(dispatcherQueue);
    }
```

```
    public void receiveSync() throws JMSException {
        Message message = consumer.receive();
        Performer performer = new Performer(nextID++, message);
        new Thread(performer).start();
    }
}
```

The performer must implement Runnable so that it can run in its own thread. The runnable's run() method simply calls processMessage(). When this is complete, the performer becomes eligible for garbage collection.

```
import javax.jms.JMSException;
import javax.jms.Message;

public class Performer implements Runnable {

    private int performerID;
    private Message message;

    public Performer(int id, Message message) {
        performerID = id;
        this.message = message;
    }

    public void run() {
        try {
            processMessage();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void processMessage() throws JMSException, InterruptedException {
        int id = message.getIntProperty("cust_id");

        System.out.println(System.currentTimeMillis() + ": Performer #"
          + performerID + " starting; message ID " + id);
        Thread.sleep(500);
        System.out.println(System.currentTimeMillis() + ": Performer #"
          + performerID + " processing.");
        Thread.sleep(500);
        System.out.println(System.currentTimeMillis() + ": Performer #"
          + performerID + " finished.");
    }
}
```

Implementing a simple dispatcher and performer is easy. The main trick is to make the performer a Runnable and run it in its own thread.