TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANTTI UUSIMÄKI

TOIMINNANOHJAUSJÄRJESTELMÄN RAKENNE JA KEHITYS

Diplomityö

Tarkastaja: professori Hannu-Matti Järvinen Tarkastaja ja aihe hyväksytty 1. marraskuuta 2017

TIIVISTELMÄ

ANTTI UUSIMÄKI: Toiminnanohjausjärjestelmän rakenne ja kehitys
Tampereen teknillinen yliopisto
Diplomityö, 50 sivua
Huhtikuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Hannu-Matti Järvinen

Avainsanat: ERP, toiminnanohjaus, järjestelmä, e1, rakenne, kehitys, kerros, arkkitehtuuri, antisuunnittelumalli, ylläpidettävyys, asiakas, palvelin

Toiminnanohjausjärjestelmä on koko yrityksen resurssisuunnittelussa käytetty tietojärjestelmä, joka integroi datan kaikista yksiköistä keskitettyyn tietokantaan. Tässä työssä esitellään lyhyesti e1-järjestelmä, joka on toiminnanohjausjärjestelmäksi kategorisoitu tietojärjestelmä. E1-järjestelmä ei täytä toiminnanohjausjärjestelmän määritelmää, koska yksiköiden välillä on erilliset tietokannat.

E1-järjestelmä on kehitetty Windows-ympäristöön käyttäen Microsoftin tuottamia teknologioita. Rakenne on suunniteltu 3-kerrosarkkitehtuuriin perustuen, mutta toteutuksessa ei ole noudatettu 3-kerrosarkkitehtuurille ominaisia periaatteita. E1-järjestelmällä on myös asiakas-palvelin-arkkitehtuurin piirteitä.

Työssä havaittiin, että e1-järjestelmä ei ole toteutukseltaan helposti ylläpidettävä järjestelmä. Ylläpidettävyyttä voidaan jossain määrin kehittää, mutta rakenteellisten ratkaisujen vuoksi järjestelmää ei voida kehittää helposti ylläpidettäväksi. e1-järjestelmässä komponenteilla ei ole myöskään selkeitä vastuualueita ja kaikki kerrokset ovat vahvasti riippuvaisia toisistaan. Tietokanta on toteutettu antisuunnittelumalleihin perustuen, joten e1-järjestelmässä ei ole yhtään uudelleen käytettävää komponenttia.

ABSTRACT

ANTTI UUSIMÄKI: Structure and Design of Enterprise Resource Management System
Tampere University of Technology
Master of Science Thesis, 50 pages
April 2018
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Hannu-Matti Järvinen

Keywords: ERP, Enterprise Resource Management, system, e1, structure, development, layer, tier, architecture, anti pattern, maintenace, client, server

Enterprise resource planning system manages resource planning for entire organization and integrates data from all organization units to centralized database. This work introduces briefly e1 system, which is categorized as ERP system. E1 does not fufill requirements of ERP system, because organization units have separated databases.

E1 system is designed in Windows environment and implemented with technologies that are produced by Microsoft. Structure of e1 design is based on 3-tier-architecture, but implementation of e1 does not follow 3-tier-architecture principles. E1 system have also features from client-server-architecture.

This work revealed that e1 system is not maintainable system at all, but there are some development that may improve e1 maintenance. Structural design of e1 is so messy, that it is impossible to make e1 system easy to maintain. Software components of e1 do not have clear responsibilities and layers of e1 system are tightly depented on each other. Also e1 database is designed by using anti patterns, so e1 system does not have any reusable component.

ALKUSANAT

Kiitokset professori Hannu-Matti Järviselle diplomityön ohjauksesta.

Tampereella, 5.4.2018

Antti Uusimäki

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

->	Vierasavainviittaus
ADOP	Always Depend on One's Parent
CLR	Common Language Runtime
ERP	Enterprise Resource Planning
FCL	Framework Class Library
IIS	Internet Information Services
OSFA	One Size Fits All
PDA	Personal Digital Assistant
SLT	Single Lookup Table
SQL	Structured Query Language
STI	Single Table Inheritance
XML	Extensible Markup Language

1. JOHDANTO

ERP on lyhenne englannin kielen sanoista Enterprise Resource Management, jota käytetään kategorisoimaan yrityksien resurssisuunnitteluun tarkoitettuja tietojärjestelmiä. ERP on suomeksi käännettynä ERP-järjestelmä eli toiminnanohjausjärjestelmä (Toiminnanohjausjärjestelmä).

ERP-järjestelmä on hyvin laaja käsite ja sen määritelmä voi muuttua hyvin paljon eri toimialojen välillä, mutta oleellista on tiedon keskittäminen kaikkien organisaation yksiköiden välillä yhteiseen tietokantaan. ERP-järjestelmä huolehtii kokonaan yrityksen liiketoiminnalle tärkeistä prosesseista kuten esimerkiksi tilaustenhallinnasta, varastonhallinnasta, laskutuksesta ja työvoimasta integroiden tämän keskitettyyn tietokantaan. ERP-järjestelmät ovat nykypäivää ja yhä useampi keskisuuri yritys ottaa ERP-järjestelmän käyttöön liiketoiminnan tueksi. ERP-järjestelmän tavoitteena on lisätä tuottavuutta ja tarjota reaaliaikaista dataa koko yrityksen toiminnasta (What is ERP).

Usein ERP-järjestelmän hankinta perustuu olemassa olevaan tuotteeseen, josta otetaan käyttöön yrityksen tarvitsemat moduulit, jotka voivat olla materiaalinohjaukseen, varastonhallintaan, talouteen, ostoihin, henkilöstönhallintaan ja tilaustenhallintaan liittyviä ominaisuuksia. Tämän tyyppisissä ERP-ratkaisuissa moduulit otetaan yleensä käyttöön vaiheittain, ja tarpeen mukaan niitä voidaan myös valita lisää. Yrityksen kaikkia tarvitsemia ominaisuuksia ei välttämättä löydy ERP-ratkaisun moduuleista, jolloin keskitetty tietokanta tarjoaa datan erillisjärjestelmälle, joka toteuttaa nämä ominaisuudet (Toiminnanohjausjärjestelmä).

ERP-järjestelmien käyttöönotto on usein hyvin raskas prosessi, jonka aikana ilmenee erilaisia ongelmia ja pahimmassa tapauksessa tämä voi aiheuttaa yritykselle konkurssin. Usein vaatimusten muuttuminen käyttöönoton aikana viivästyttää projektin etenemistä (ERP-j rjestelm n hankinta).

Työssä käsitellään toiminnanohjausjärjestelmää, jonka ratkaisu perustuu edellä mainittuun modulaariseen rakenteeseen. ERP-järjestelmän käyttöönoton vaiheet käydään lyhyesti läpi, koska ERP-järjestelmille on ominaista hyvin raskas käyttöönottoprosessi. Toiminnanohjausjärjestelmien elinkaari voi olla hyvin pitkä, joten työssä esitellään lyhyesti ERP-järjestelmän kehityshistoria.

ERP-järjestelmän esittely tehdään toisessa luvussa, jossa esitellään myös suunnitelma toiminnanohjausjärjestelmän rakenteesta. Toisessa luvussa esitellään myös korkean tason hajautus nykyisen ERP-järjestelmän laitteista ja ohjelmista, sekä esitellään lyhyesti ERP-

järjestelmässä käytetyt keskeisimmät teknologiat. Kolmannessa luvussa esitellään teoriaa järjestelmästä löydetyistä korkeimman tason ratkaisuista ja tietokannan suunnittelussa käytetyistä ratkaisuista.

Työn tavoitteena on rekonstruoida ERP-järjestelmän rakenne, koska järjestelmästä ei ole kattavaa dokumentaatiota. Rekonstruointi tehdään korkeimman tason päätösten mukaan koko järjestelmän laajuisesti. Tietokannan suunnittelussa käytetyt ratkaisut rekonstruoidaan yksityiskohtaisemmin, koska yrityksen keskitetty tietokanta on ERP-järjestelmän keskeinen ominaisuus.

Rekonstruoinnin yhteydessä kiinnitetään erityisesti huomiota komponenttien vastuualueisiin ja rajapintojen käyttöön, koska työssä on myös tavoitteena tarkastella nykyisen ERP-järjestelmän ylläpidettävyyttä ja arvioida ylläpidettävyyden parantamiseen vaadittavaa työn määrää. Rekonstruointi tehdään luvussa 4 ja ERP-järjestelmän ylläpidettävyyden tarkastelu tehdään luvussa 5.

Luvussa 6 ehdotetaan järjestelmän jatkokehitysajatuksia työn pohjalta. Viimeisessä luvussa on yhteenveto työstä.

2. JÄRJESTELMÄN ESITTELY

Tässä luvussa esitellään järjestelmän käyttökonteksti ja järjestelmää kehittäneen yrityksen tavoitteet järjestelmän arkkitehtuurissa. Järjestelmän ylläpidettävyyden kannalta on myös tärkeätä esitellä järjestelmän kehityshistoria ja käyttöönotonvaiheet sekä järjestelmän elinkaaren aikana tapahtuneet muutokset järjestelmää kehittäneissä ja ylläpitäneissä organisaatioissa.

2.1 Järjestelmän tavoitteet

E1 on Efigen Oy:n kehittämä ERP-tuote, joka Efigen Oy:n mukaan koostuu moduuleista eli itsenäisistä ohjelman osista, jotka otetaan käyttöön asiakkaan tarpeiden mukaan. E1-tuotteen hinnoittelu perustuu vuokrattujen moduulien määrään. Efigen Oy:n mukaan E1-järjestelmä on kohdistettu tuotantoyrityksille, joiden tarpeissa on haluttu kiinnittää huomiota erityisesti käyttöönoton helppouteen ja järjestelmän joustavuuteen sekä tuotannon raportointiin (Karvonen 2004).

Oulun yliopiston apulaisprofessori Päivi Iskanius mainitsee raportissaan e1-järjestelmän olevan helposti integroitavissa yrityksen muihin tietojärjestelmiin, tiedonkeruulaitteisiin, mobiililaitteisiin ja tuotantoautomaatioon. E1-järjestelmän vahvuuksiin kuulu kattava nimikejärjestelmä. E1-tuotetta käytettiin vuonna 2009 esimerkiksi paperiteollisuuden, metalliteollisuuden, kemian teollisuuden ja koneteollisuuden yrityksissä (Iskanius & Juuso 2009).

Tässä työssä käsitellään e1-tuotteen versiota, jota käytetään suomalaisen paperin- ja kartonginjalostusteollisuuden yrityksen kolmessa yksikössä. Järjestelmän vastuualueisiin kuuluu esimerkiksi tilaustenkäsittely, tuotantoprosessin ohjaus, varastonhallinta, logistiikka ja raportointi.

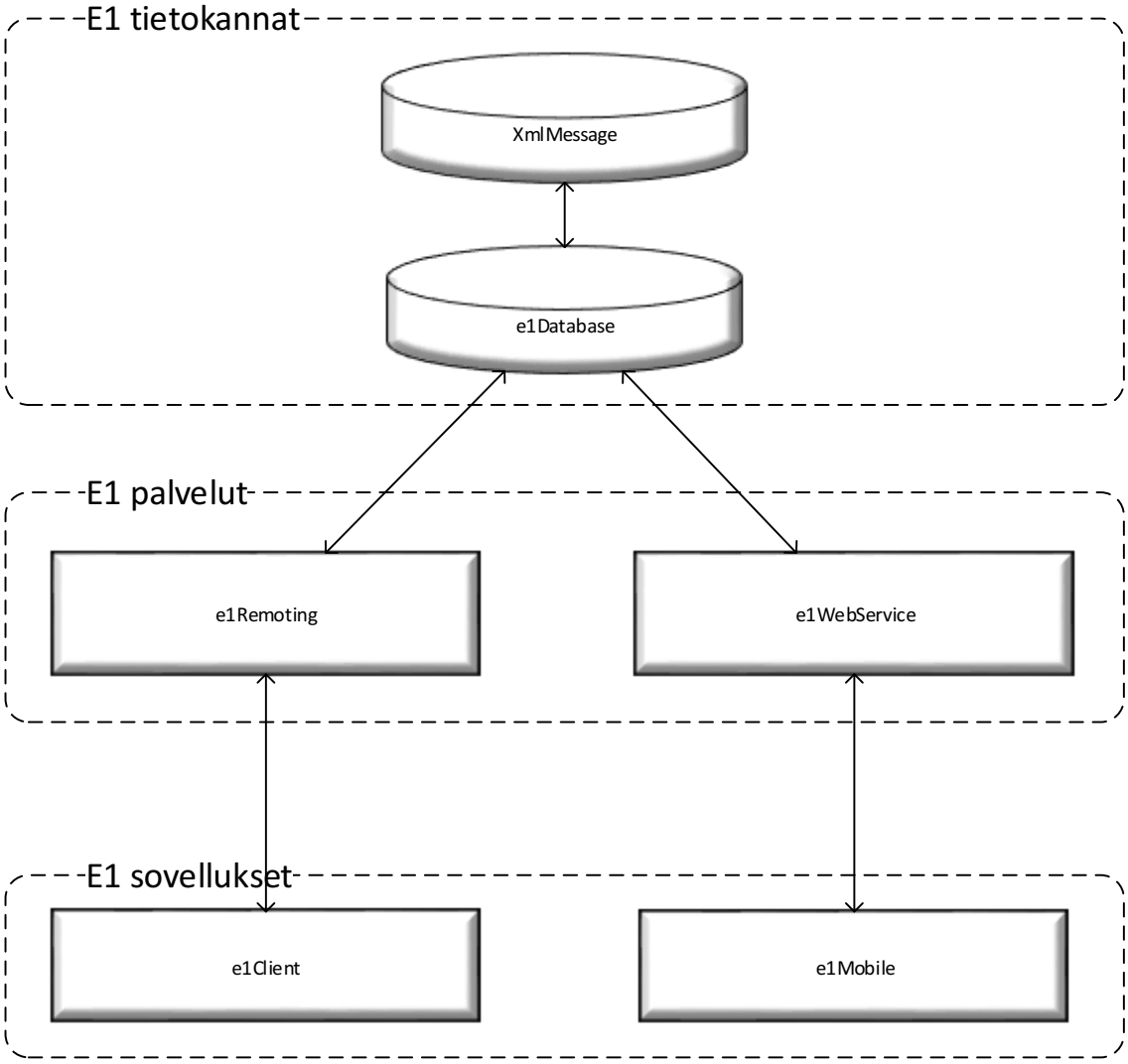
2.2 E1-järjestelmän kehityshistoria

Efigen Oy on tehnyt ohjelmistoratkaisuja myös aputoiminimellä Suomen ATK-osasto (Efigen Oy). CDM on Suomen ATK-osaston kehittämä e1-tuotetta edeltänyt ERP-järjestelmä. E1-tuotteen kehitys on aloitettu vuonna 2003 käyttäen uudelleen osaa CDM-järjestelmän komponenteista.

E1-järjestelmän ensimmäinen versio otettiin käyttöön vuonna 2007 kartonginjalostusteollisuuden yrityksen pääyksikössä ja vuonna 2008 toisessa yksikössä. Kehitystöiden jälkeen e1-järjestelmä otettiin käyttöön vuonna 2010 kolmannessa yksikössä. Efigen Oy:n mennessä konkurssiin vuonna 2012, e1-järjestelmän ylläpito ja jatkokehitys on siirtynyt eteenpäin. Vuonna 2016 Eatech Oy aloitti e1-järjestelmän ylläpidon ja jatkokehityksen.

2.3 Suunniteltu e1-järjestelmä

E1-järjestelmästä ei ole olemassa kattavaa dokumentaatiota, mutta korkean tason kuvaus järjestelmän hajautuksesta löytyy. Kuvassa 1 on esitetty jäljennös järjestelmän alkuperäisten kehittäjien dokumentoimasta e1-järjestelmästä.

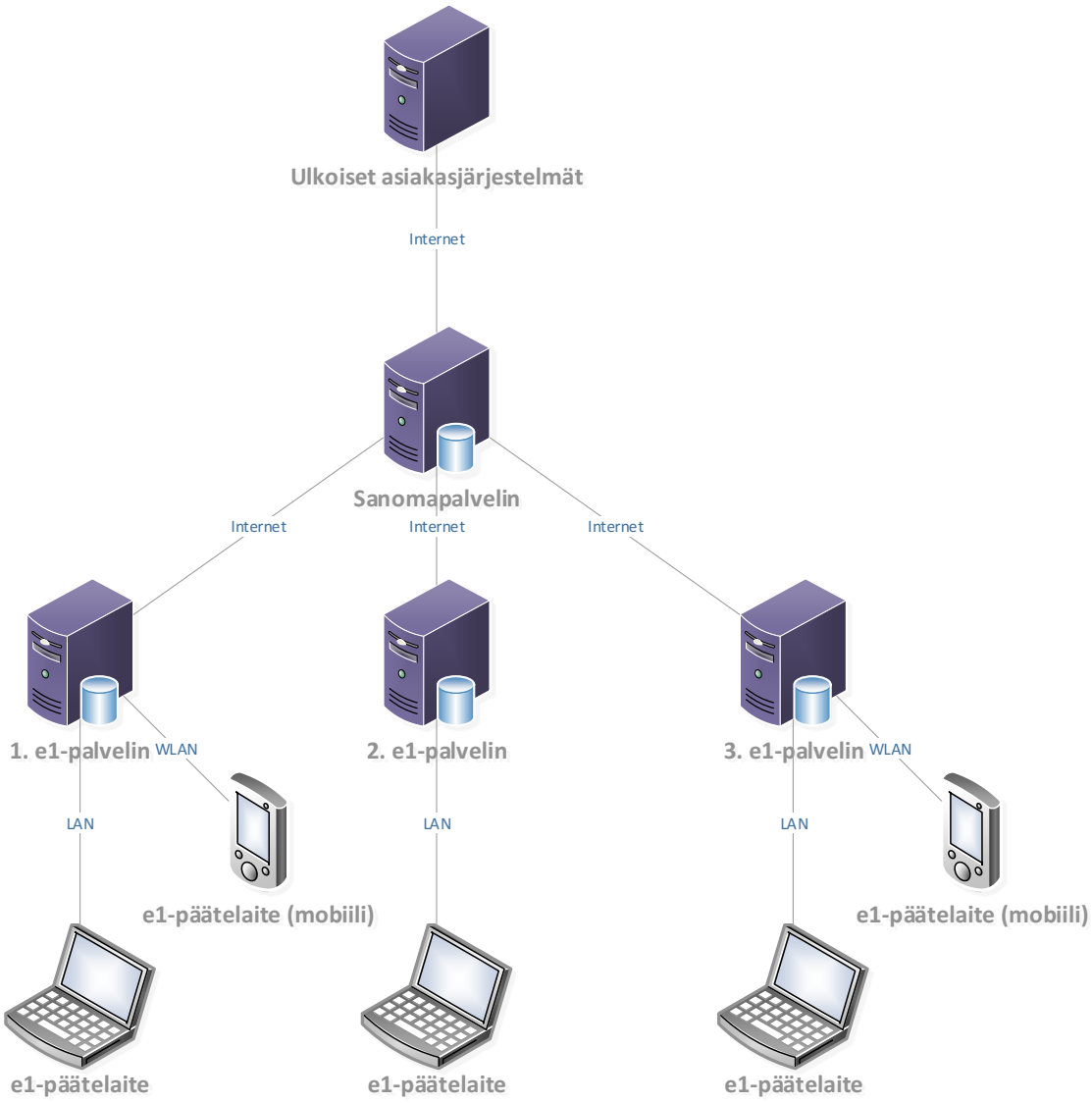


Kuva 1. *Dokumentoitu e1-järjestelmä*

Kuvasta 1 voidaan nähdä, että suunnitelmana on ollut kolmeen erilliseen osaan jaettu kerrosrakenne. E1-sovelluksia ovat e1Client- ja e1Mobile-asiakasohjelmat, jotka ovat toisistaan erillisiä. E1-palvelut ovat myös toisistaan erilliset. e1Client-asiakasohjelma käyttää e1Remoting-palvelua viestinnässä, eikä se ole suoraan yhteydessä tietokantaan. e1Mobile-asiakasohjelma käyttää e1WebService-palvelua viestinnässä. E1-palvelut käyttävät e1Database-tietokantaa, joka on yhteydessä XmlMessage-tietokantaan.

2.4 Nykyinen e1-järjestelmä

Nykyisessä e1-järjestelmässä on kolme e1-palvelinta ja yksi sanomapalvelin. E1-palvelimet muodostavat paikallisen alijärjestelmän, jotka muodostuvat e1-palvelimeen liitetyistä päätelaitteista ja integraatioista tuotantoympäristöön. E1-päätelaitteisiin kuuluu erilaiset tietokoneet ja PDA-laitteet. PDA-laitteita käytetään viivakoodinlukijoina. Kuvassa 2 on esitelty e1-järjestelmään kuuluvat palvelimet ja päätelaitteet.



Kuva 2. *e1-järjestelmään kuuluvat laitteet*

Yksiköiden paikalliset e1-palvelimet on nimetty järjestyksessä e1-järjestelmän käyttöönoton mukaan, esimerkiksi järjestysnumerolla yksi viitataan vuonna 2007 käyttöönotetun paikallisen e1-alijärjestelmän e1-palvelimeen. Kuvasta 2 voidaan nähdä, ettei nykyinen e1-järjestelmä noudata ERP-järjestelmän määritelmää, koska resurssisuunnittelussa ei ole keskitettyä tietokantaa yksiköiden välillä.

2.4.1 Sanomapalvelin

Sanomapalvelimen vastuulla on internetin välityksellä sekä vastaanottaa asiakkailta, että lähettää asiakkaille erityyppisiä sanomia. Sanomapalvelimella on XmlMessage-tietokanta, joka pitää tallessa järjestelmässä kulkevia sanomia. Sanomapalvelin käyttää e1-alijärjestelmien tietokantoja, ja sen erillinen ohjelmisto toteuttaa integraation paikallisten e1-alijärjestelmien ja asiakkaiden välillä. Sanomapalvelimen tietokanta ei siis ole e1-järjestelmän keskitetty tietokanta resurssisuunnittelua varten.

2.4.2 E1-palvelimet

E1-järjestelmässä on kolme e1-palvelinta, joissa on erilliset tietokannat paikallisten e1-alijärjestelmien resurssisuunnittelua varten. Nykyisessä e1-järjestelmässä on kolme e1Remoting-palvelua ja kaksi e1WebService-palvelua.

2.4.3 E1-asiakasohjelmat

Tietokoneiden e1Client-asiakasohjelmalla hallinnoidaan paikallista alijärjestelmää e1-palvelimella sijaitsevan e1Remoting-palvelun kautta. PDA-laitteissa on suppea e1Mobile-asiakasohjelma, joka on tarkoitettu varastonhallinnan tueksi. PDA-laitteiden e1Mibile-asiakasohjelma kommunikoi alijärjestelmän kanssa e1-palvelimella sijaitsevan erillisen e1WebService-palvelun kautta.

2.5 Teknologiat

E1-järjestelmä on rakennettu Microsoftin Windows-ympäristöön ja järjestelmän komponentit on toteutettu pääasiassa Microsoftin tuottamilla teknologioilla. Tässä luvussa esitellään lyhyesti e1-järjestelmän keskeisimmät teknologiat.

2.5.1 .NET-sovelluskehys 3.5

.NET on Microsoftin kehittämä sovelluskehys, joka tarjoaa laajan FCL-luokkakirjaston ohjelmistokehittäjille. FCL-luokkakirjastot koostuvat esimerkiksi uudelleen käytettävistä luokista, rajapinnoista ja tietotyypeistä. Sovelluskehyksen runkona on CLR-virtuaalikone, joka huolehtii esimerkiksi .NET-sovelluskehyksellä tuotettujen ohjelmien suorituksesta, muistinhallinnasta ja poikkeuksien käsittelystä (Troelsen 2012).

2.5.2 C#-ohjelmointikieli

C# on Microsoftin kehittämä C-perheen ohjelmointikielistä ja Java-ohjelmointikielestä vaikutteita saanut olioparadigmaa noudattava ohjelmointikieli. C#-ohjelmointikielen ta-

voitteena on lisätä tuottavuutta ja sen käyttökohteisiin kuuluu esimerkiksi ohjelmakomponenttien tuottaminen hajautettuihin järjestelmiin. Lisäksi C#-ohjelmointikielellä on pääsy koko FCL-luokkakirjastoon, jolloin sen käyttöä suositellaan erityisesti sovelluslogiikan toteutuksessa (Troelsen 2012).

2.5.3 Windows Forms

Windows Forms on .NET-ympäristöön kehitetty teknologia asiakasohjelmien käyttöliittymien toteutukseen, se tarjoaa erilaisia elementtejä tiedon esitystä varten. Windows Forms -elementeillä on erilaisia tapahtumia, jotka aktivoituvat käyttäjän toiminnoista, kuten painikkeen painamisesta. Tapahtumankäsittelijät ovat sidottu tapahtumiin ja ne vastaavat niiden toiminnallisuudesta. Windows Forms -elementtien arvot voidaan sitoa olioiden ominaisuuksiin ja listoihin, jotka toteuttavat IList-rajapinnan (Windows Forms).

2.5.4 Microsoft SQL Server

Microsoft SQL Server on Microsoftin kehittämä tietokannanhallintajärjestelmä SQL-tietokannalle, joka tarjoaa rajapinnan tietokantaan pääsyä varten. SQL-tietokanta on relaatiotietokannan malli, jossa tieto tallennetaan yhteen tai useampaan tauluun. Tietokantataulut koostuvat sarakkeista ja riveistä. Tietokantataulut mallinnetaan reaalimaailman entiteeteistä, joiden ominaisuuksia eli attribuutteja sarakkeet kuvaavat. Tietokantataulun rivit kuvaavat entiteetin attribuuttien varsinaisia arvoja. Tietokantataulussa on yksi tai useampi sarake, jotka toimivat taulun pääavaimena. Pääavain on entiteetin rivien yksikäsitteinen tunniste. Tietokantatauluilla voi olla myös vierasavaimia, jotka viittaavat muiden taulujen pääavaimiin. Vierasavaimia käytetään tietokantataulujen välisissä liitoksissa (Syverson & Murach 2016).

Tietokantaproseduuri on tietokantaan tallennettu SQL-kielellä tehty ohjelma, joka pystyy tekemään tietokannan dataa muuttavia kyselyitä, kuten SQL-kielen INSERT-, UPDATE- ja DELETE-kyselyitä. Tietokantafunktion erona tietokantaproseduuriin on tietokannan dataa muuttavien operaatioiden puuttuminen, jolloin on vain SELECT-kyselyn käyttäminen mahdollista (Syverson & Murach 2016).

Tietokantaproseduureille ja tietokantafunktioille määritellään vaaditut parametrit ohjelman suoritusta varten. Tietokantafunktion ja tietokantaproseduurin paluuarvo voi olla yksittäinen arvo tai jokin määrä rivejä. Tietokantafunktion rakenteessa on pakotettu yksittäisen paluuarvon tyypin tai palautettavan taulun rakenteen määrittäminen. Tietokantaproseduurin paluuarvo voi muuttua ohjelman suorituksesta riippuen yksittäisestä arvosta useaan riviin (Syverson & Murach 2016).

2.5.5 Extensible Markup Language

XML on rakenteinen dokumentti, joka on suunniteltu sekä ihmisen, että koneen ymmärrettäväksi menetelmäksi tiedon tallentamista ja kuljettamista varten. XML-dokumentteja voidaan pitää tallessa ajoaikaisessa muistissa tai ne voidaan tallentaa xml-päätteiseen tiedostoformaattiin (XML Tutorial).

2.5.6 DataSet

DataSet on .NET-sovelluskehyksen tietoaineisto, joka on kuvaus relaatiotietokannan rakenteesta. DataSet-tietoaineistoon kuuluu yksi tai useampi DataTable-taulu, joka on ohjelman kuvaus jostain relaatiotietokannan taulusta. DataTable-taulu pitää tallessa kokoelmaa DataRow-riveistä, jotka ovat kuvauksia relaatiotietokannan taulun riveistä (Troelsen 2012).

2.5.7 Windows Service

Windows Service on Windows-ympäristön taustalla ajossa oleva palvelu, jolle on ominaista automaattinen käynnistyminen tietokoneen käynnistämisen yhteydessä. Windows Service -palvelut ovat usein pitkäaikaisesti ajossa eivätkä ne vaati käyttäjän vuorovaikutusta toimiakseen. Windows Service -palvelu voi toimia esimerkiksi osana järjestelmää, jonka metodeja asiakasohjelmat kutsuvat .NET Remoting -rajapintojen kautta (Troelsen 2012).

2.5.8 .NET Remoting

.NET Remoting on teknologia, joka mahdollistaa hajautetussa järjestelmässä komponenttien käyttämisen etänä. Etäkomponentti aktivoidaan URL-osoitteen kautta, joka luo aktivointipyynnön antajalle etäkomponentin edustajan. Edustajakomponentti huolehtii pyyntöjen välityksestä oikealle etäkomponentti-instanssille. Aktivointipyynnön jälkeen etäkomponentin metodeja voidaan kutsua samoin kuin paikallisen komponentin metodeja (Microsoft .NET Remoting: A Technical Overview).

2.5.9 Web Service

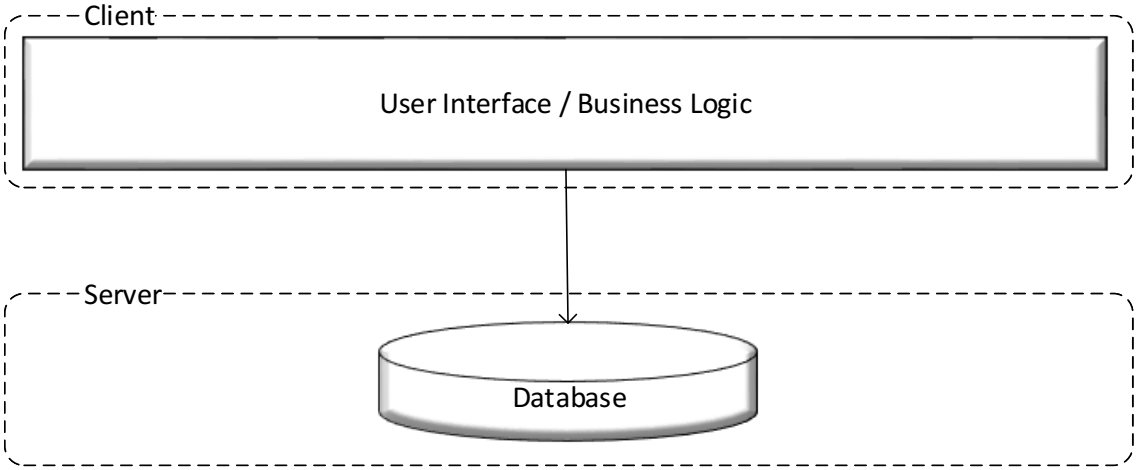
Web Service on Microsoftin kehittämä Web-palvelu, joka tarjoaa Web-rajapinnan ja käyttää kommunikoinnissa standardeja Web-protokollia. Web-palvelut ovat ajossa IIS-palvelimella. Web-palvelut käyttävät tiedon siirrossa XML-formaattia, joka standardeja Web-protokollia käyttäen mahdollistaa riippumattomuuden käyttöjärjestelmästä ja ohjelmointikielestä järjestelmän osien välillä (Troelsen 2012).

3. SUUNNITTELUMALLIT

Ohjelmistoarkkitehtuurille ei ole yksimielisesti hyväksyttyä määritelmää, mutta usein ohjelmistoarkkitehtuurin kontekstiin liitetään korkeimman tason jaottelu järjestelmän osista ja rakenteelliset päätökset, joita on hankala muuttaa myöhemmin. Järjestelmässä voi olla myös usea arkkitehtuurinen ratkaisu samaan aikaan. Suunnittelumallit kuvaavat ratkaisuja toistuvasti esiintyneisiin ongelmiin (Fowler 2002a). Antisuunnittelumallit kuvaavat puolestaan toistuvasti esiintyneisiin ongelmiin ei toivottuja ratkaisuja, joista seuraa lähes aina haittaa ohjelmiston elinkaaren aikana (Karwin 2014).

3.1 Asiakas-palvelin-arkkitehtuuri

Asiakas-palvelin-arkkitehtuuri on hajautetuissa järjestelmissä käytetty rakenne, jossa asiakasohjelman vastuualueena on käyttöliittymä ja palvelin on usein relaatiotietokanta. Asiakas-palvelin-arkkitehtuuri sopii hyvin, jos tiedon määrä on hyvin vähäistä ja tehdään yksinkertaisia toimintoja (Fowler 2002a). Kuvassa 3 on esitelty asiakas-palvelin-arkkituuri, jossa asiakasohjelman vastuualueisiin kuuluvat käyttöliittymä ja sovelluslogiikka.



Kuva 3. *Asiakas-palvelin-arkkitehtuuri*

Asiakasohjelman toteutuksessa laadukkaaseen ohjelmointiin kuuluu käyttöliittymän ja sovelluslogiikan toteuttaminen erillisinä komponentteina, jolloin toistuvat rakenteet voidaan käyttää uudelleen. Asiakas-palvelin-arkkitehtuurin ongelmana on, että usein sovelluslogiikka kirjoitetaan käyttöliittymään kiinni, jonka seurauksena toistuvia rakenteita ei voida käyttää uudelleen ja ohjelman toteutus tehdään kopioimalla vanhaa ohjelmakoodia.

Ohjelmakoodin kopiointi tekee ohjelmakoodin laajentamisesta hankalaa, koska muutokset täytyy tehdä kaikkialle ohjelmaa missä kopioitua koodia on käytetty (Fowler 2002a).

Asiakas-palvelin-arkkitehtuurin järjestelmissä on myös vaihtoehtoisesti käytetty rakennetta, jossa sovelluslogiikka on toteutettu tietokantaproseduureilla. Tämä on usein johtanut hyvin epäselvään rakenteeseen (Fowler 2002a).

3.2 Kerrosarkkitehtuuri

Kerrosarkkitehtuurissa järjestelmän rakenne jaetaan kerroksiin, jossa ylempi kerros käyttää alemman kerroksen palveluita. Kerroksiin jakaminen tehdään välillä ihminen ja laite, joten esimerkiksi hajautetun järjestelmän ylin kerros on käyttöliittymä ja alin kerros on tietokanta (Koskimies & Mikkonen 2005).

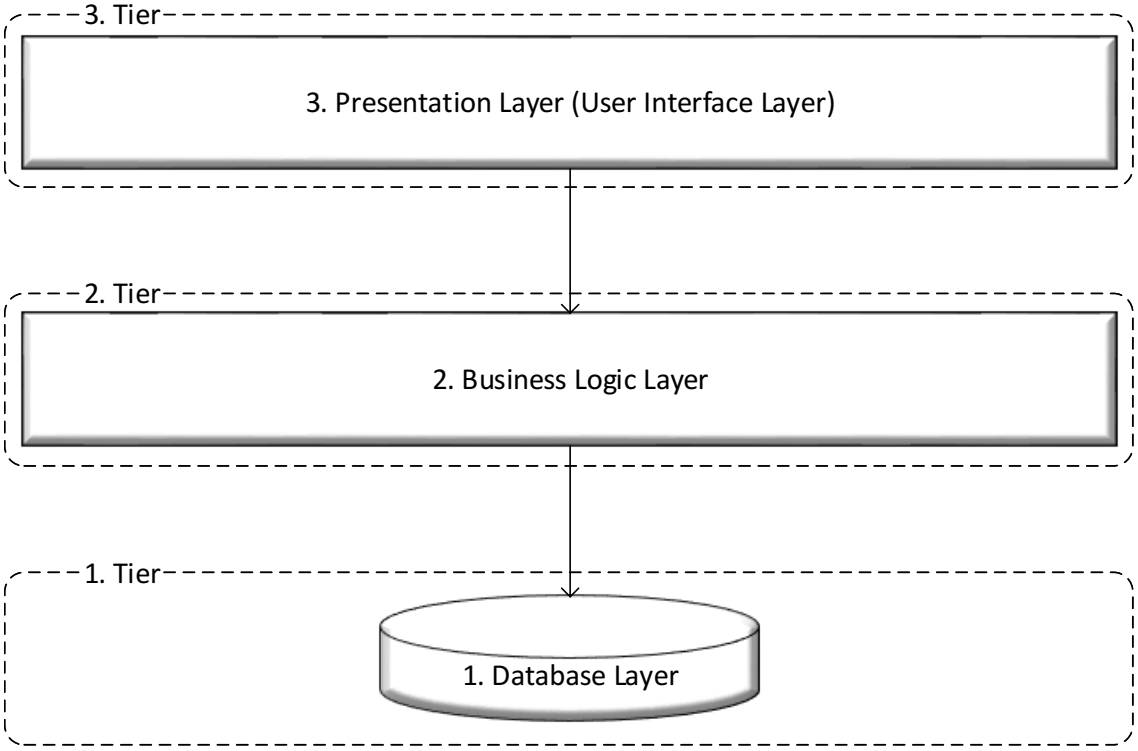
3-kerrosarkkitehtuurissa järjestelmä jaetaan kolmeen kerrokseen, jotka tyypillisesti ovat käyttöliittymä, sovelluslogiikka ja tietokanta. Käyttöliittymä on ohjelman osa, jonka ihminen näkee järjestelmästä ja sen tehtävänä on esittää tieto ihmiselle ymmärrettävässä muodossa sekä tarjota rajapinta järjestelmän käyttöön. Käyttöliittymä ei ole suoraan yhteydessä tietokantaan vaan kommunikointi tapahtuu sovelluslogiikkakerroksen välityksellä. Sovelluslogiikkakerroksessa tehdään tiedon validointi, järjestelmän raskaat laskutoimitukset sekä tietokannan käsittely (Fowler 2002a).

Kerrosarkkitehtuuria koskee kerroksien eristyneisyyden periaate ”*Layers of Isolation*”, jonka mukaan palvelukutsut välittyvät vain ylemmältä kerrokselta sitä heti alemmalle kerrokselle, jolloin kerroksien välinen riippuvuus on mahdollisimman pieni. Periaatteen rikkominen johtaa kerrosarkkitehtuurin hyötyjen heikkenemiseen, jonka seurauksena voi olla erittäin vaikeasti ylläpidettävä järjestelmä (Richards 2015).

Kerrosarkkitehtuurin hyötynä on kerroksien käsittely itsenäisinä ohjelman osina ja modulaarisuuden lisääminen, jolloin käyttöliittymä on vain riippuvainen sovelluslogiikkakerroksesta ja sovelluslogiikka on vain riippuvainen tietokannasta. Kerroksiin jakamisen tavoitteita ovat myös rakenteen ryhmittely helpommin ymmärrettäväksi kokonaisuudeksi ja tehdä järjestelmän kehitystyö helpommaksi, koska kerroksiin jakaminen usein sovitetaan myös organisaation eri osaamisalueisiin. Kerrosarkkitehtuurin periaatteisiin kuuluu

hyvin suunnitellut rajapinnat sekä selkeä vastuunjako eri kerroksien komponenttien välillä (Richards 2015).

Kerrosarkkitehtuurissa fyysinen kerroksien hajauttaminen ”*Tier*” tarkoittaa, ett ohjelmakerrokset ”*Layer*” ovat laitetasolla e n toisistaan (Fowler 2002a). Kuvassa 4 on esitelty malli 3-kerrosarkkitehtuurista, jossa hajauttaminen on tehty myös laitetasolla.



Kuva 4. *3-kerrosarkkitehtuuri*

3-kerrosarkkitehtuurin tavoitteena on, että sovelluslogiikkakerros ja tietokantakerros ovat myös laitetasolla erillään toisistaan, mutta usein pienemmissä järjestelmissä sovelluslogiikkakerros sijaitsee samalla palvelimella kuin tietokantakerros. 3-kerrosarkkitehtuurissa erillisen sovelluslogiikkakerroksen tavoitteena on ratkaista asiakas-palvelin-arkkitehtuuriin liittyvät ongelmat (Fowler 2002a).

Kerroksien välisiä riippuvuuksia analysoidaan riippuvuusmatriisilla. Riippuvuusmatriisissa kerrokset asetetaan ylimmästä lähtien järjestyksessä matriisin vasemman reunan sarakkeeseen. Kerrokset asetetaan myös järjestyksessä taulukon 1 mukaisesti matriisin ylimmälle riville. Analysointi tehdään matriisin vasemman reunan sarakkeen mukaan, jolloin kyseisen kerroksen riippuvuuksiin laitetaan rastit sarakkeiden kohdalle, joista kerros on riippuvainen. Rastit muodostavat lävistäjän matriisiin, jolloin väärään suuntaan

menevät riippuvuudet näkyvät lävistäjän vasemmalla puolella (Koskimies & Mikkonen 2005).

Taulukko 1. *Kerroksien riippuvuusmatriisi*

Rivikerros riippuu sarakekerroksesta	**User Interface**	**Business Logic**	**Database**
User Interface		X	
Business Logic			X
Database			

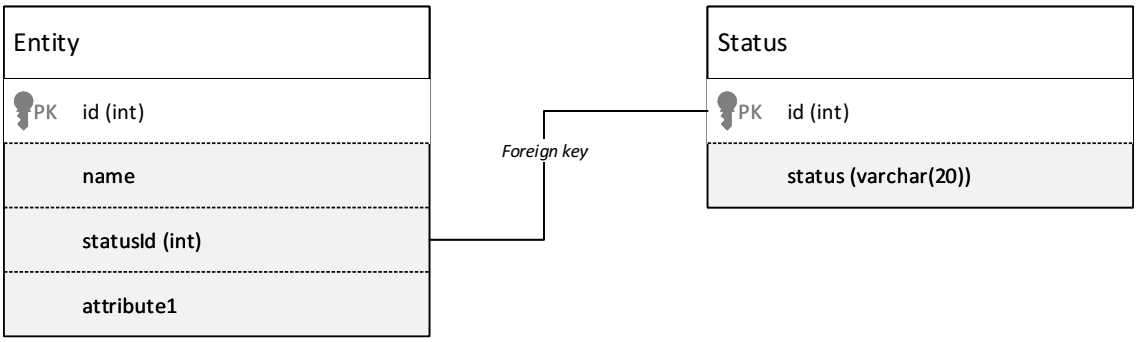
Taulukossa 1 on esitelty käyttöliittymän, sovelluslogiikan ja tietokannan riippuvuusmatriisi, jossa kerrokset ovat riippuvaisia vain sitä heti alemmasta kerroksesta. Taulukossa 1 käyttöliittymä on riippuvainen vain sovelluslogiikasta ja sovelluslogiikka on riippuvainen vain tietokannasta, eikä tietokannalla ole riippuvuuksia. Väärään suuntaan menevä riippuvuus näkyisi taulukossa 1 esimerkiksi Business Logic -rivillä sarakkeen User Interface kohdalla.

3.3 One Size Fits All -antisuunnittelumalli

OSFA-antisuunnittelumalli ”*One Size Fits All*” on tietokantasuunnittelussa esiintyvä ratkaisu, jossa tietokantataulujen pääavaimena on aina kokonaislukutyyppinen surrogaattiavain. Surrogaattiavaimella tarkoitetaan tietokannan automaattisesti generoimaa arvoa pääavaimelle. Tietokannat usein generoivat surrogaattiavaimen automaattisesti kasvavana kokonaislukuarvona, joka on ohjelmistokehitystä helpottava ominaisuus, jos tietokantataululle ei löydy selkeää yksikäsitteistä tunnistetta. Surrogaattiavaimen käyttäminen

aiheuttaa ongelmia, jos tavoitteena on pitää myös muissa sarakkeissa yksikäsitteisiä arvoja tallessa (Karwin 2014).

Esimerkiksi entiteetin tiloja kuvaavalla Status-taululle ei ole tarvetta määrittää erikseen surrogaattiavainta. Kuvassa 5 on annettu esimerkki tarpeettomasta surrogaattiavaimen käytöstä.



Kuva 5. *Tarpeeton surrogaattiavain pääavaimena*

Taulukossa 2 on annettu esimerkki Entity-taulun riveistä. Taulukosta 2 nähdään, että status-kentän arvot ovat 1 ja 2, jotka ovat Status-taulun surrogaattiavaimen generoimia arvoja. Surrogaattiavaimella ei ole ilmaisuvoimaista arvoa tässä kontekstissa.

Taulukko 2. *Tarpeeton surrogaattiavain ja entiteetin rivit*

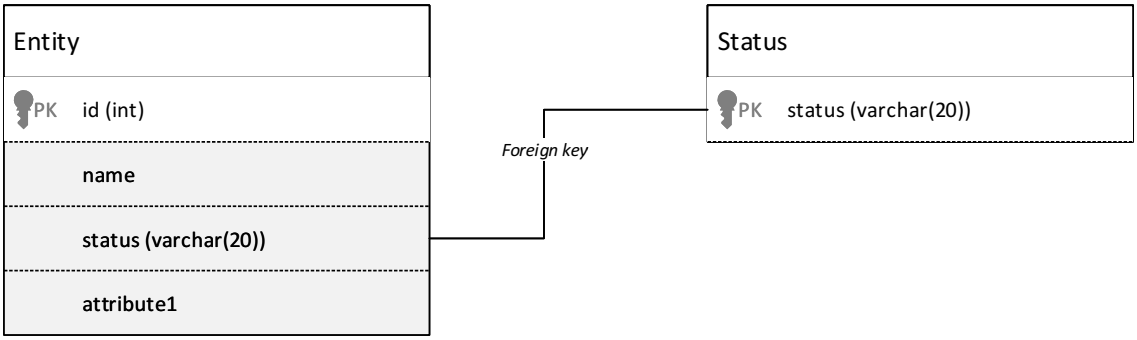
id	name	statusId	attribute1
1	Instance1	1	NULL
2	Instance2	2	NULL

Surrogaattiavaimen käyttäminen tässä kontekstissa mahdollistaa saman arvon toistumisen useaan kertaan, jos arvoa ei erikseen määritellä yksikäsitteiseksi UNIQUE-rajoitteella (Karwin 2014). Esimerkiksi, jos Status-tauluun tehdään kaksi kertaa INSERT-lause, jossa on sama arvo status-sarakkeella, niin sama arvo toistuu useaan kertaan eri surrogaattiavaimen arvolla. Ohjelmassa 1 esitetään SQL-kysely, jossa tehdään liitos entiteetin tilan näyttämiseksi.

```
SELECT e.id, e.name, s.status, e.attribute1 FROM Entity e
INNER JOIN Status s ON e.statusId = s.id
```

Ohjelma 1. *Tarpeeton surrogaattiavain ja taulujen liitos*

Käyttäessä tässä kontekstissa surrogaattiavainta, Entity-taulun rivien tilaa ei voida nähdä ennen liitosta Status-tauluun. Kuvassa 6 on esitelty vertailun vuoksi tietokantakaavio, jossa Status-taulun pääavaimena ei ole käytetty surrogaattiavainta.



Kuva 6. *Status-arvo pääavaimena*

Status-taulun tilaa ylläpitävä status-sarake on taulun pääavain, jolloin samojen arvojen toistuminen ei ole mahdollista pääavaimen yksikäsitteisyyden vuoksi (Karwin 2014). Ohjelmassa 2 on vertailun vuoksi tehty SQL-kysely Entity-tauluun.

```
SELECT id, name, status, attribute1 FROM Entity
```

Ohjelma 2. *Status-arvo vierasavainviittauksena*

Entity-taulun vierasavaimesta voidaan nyt suoraan nähdä entiteetin rivien tilat. SQL-kysely on lyhyempi kuin käyttäessä surrogaattiavainta. Taulukossa 3 on esitelty ohjelman 2 mukaisen SQL-kyselyn rivit.

Taulukko 3. *Status-arvo vierasavaimena ja entiteetin rivit*

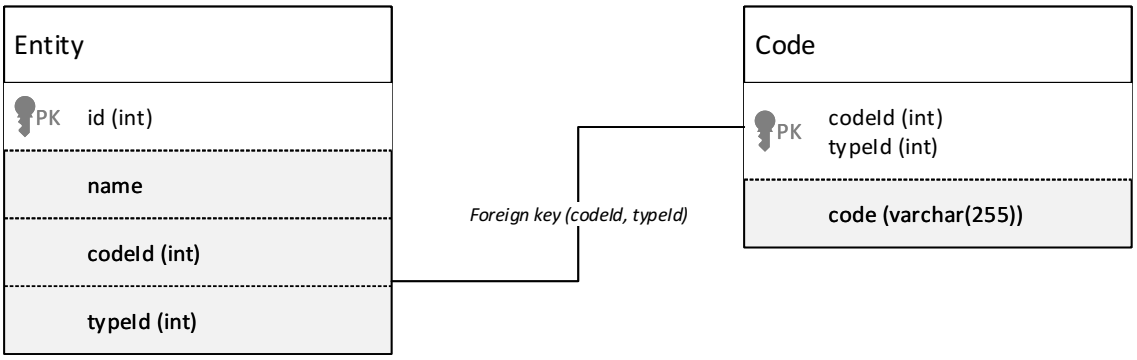
id	name	status	attribute1
1	Instance1	OLD	NULL
2	Instance2	NEW	NULL

Kaavion 4.3 mukaista ratkaisua pidetään antisuunnittelumallina, koska surrogaattiavaimella ei ole ilmaisuvoimaista arvoa ja arvon hakemiseen täytyy tehdä aina liitos. Lisäksi surrogaattiavaimen käyttäminen ei takaa arvon yksikäsitteisyyttä ilman ylimääräisiä rajoitteita ja SQL-kyselyt voivat palauttaa yllättävän määrän rivejä, jos yksikäsitteiseksi tarkoitetulla arvolla on ylimääräisä rivejä eri surrogaattiavaimella. (Karwin 2014).

3.4 Single Lookup Table -antisuunnittelumalli

SLT-antisuunnittelumallissa ”*Single Lookup Table*” tietokannassa on yksi kooditaulu, jonka nimi on usein Code. Kooditaulu pitää tallessa kaikkien entiteettien tyyppiä ja tilaa koskevia sallittuja arvoja. Code-taulussa entiteettiä koskevat arvot erotellaan toisistaan

typeId-attribuutilla, joka kategorisoi arvot entiteetin tyypin mukaan. Kooditaulussa on usein kaksiosainen pääavain, joka koostuu typeId-attribuutista ja kooditaulun surrogaattiavaimesta codeId (Peterson 2006). Kuvassa 7 on esitelty SLT-antisuunnittelumallin tietokantakaavio.



Kuva 7. *Single Lookup Table -antisuunnittelumalli*

Entiteettien typeId-kentän generointi entiteetille on usein toteutettu tietokantataulun Constraint-rajoitteen tai triggerin avulla (Peterson 2006). Taulukossa 4 on esitelty esimerkki kooditaulun riveistä.

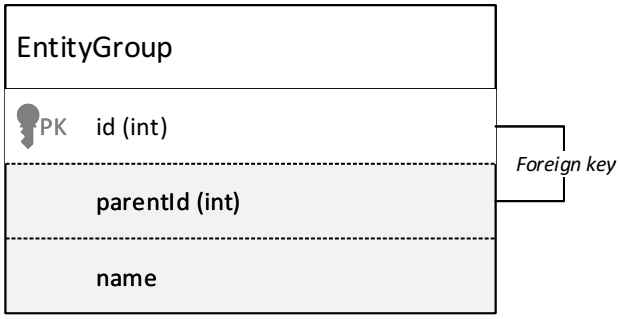
Taulukko 4. *Single Lookup Table rivit*

codeId	typeId	code
1	1	OLD
2	2	NEW
3	1	NEW
4	3	1

Taulukosta 4 nähdään, että code-sarakkeen arvona voi olla esimerkiksi yksittäinen numero ja samat arvot voivat toistua eri typeId-attribuutilla. Kooditaulun code-sarake on usein 255 merkkiä pitkä merkkijono, koska tavoitteena on varmistaa tilan riittäminen kaiken pituisille arvoille. SLT-taulun toteutuksessa voidaan käyttää joskus apuna erillistä tietokantataulua typeId-kentän arvojen määrittämistä varten (Peterson 2006).

3.5 Always Depend on One's Parent -antisuunnittelumalli

ADOP-antisuunnittelumalli "*Always Depend on One's Parent*" on ei toivottu ratkaisu toteuttaa hierarkkisia puurakenteita relaatiotietokannassa, jos tietokannalla ei ole tukea rekursiivisille kyselyille. ADOP-antisuunnittelumallissa tietokantataululla on vierasavain-viittaus itseensä. ADOP-antisuunnittelumallille ominainen piirre on parentId-niminen sarake rekursiivista viittausta varten (Karwin 2014). Kuvassa 8 on esitetty ADOP-antisuunnittelumallin mukaisen taulun tietokantakaavio.



Kuva 8. *Always Depend on One's Parent -antisuunnittelumalli*

Puurakenteen osat ovat solmuja ”*Node*”. Hierarkiassa edelt solmu on is ”*Parent*” ja seuraava solmu on lapsi ”*Child*”. Puurakenteessa juureksi ”*Root*” kutsutaan solmua, jolla ei ole ja lehdiksi ”*Leaves*” kutsutaan solmuja, joilla ei ole lapsia (Karwin 2014). Taulukossa 5 on esitetty ryhmittelyyn tarkoitetun tietokantataulun mahdollinen sisältö käyttäen ADOP-antisuunnittelumallia, josta nähdään, että solmujen määrä voi vaihdella puurakenteen juurien ja lehtien välillä.

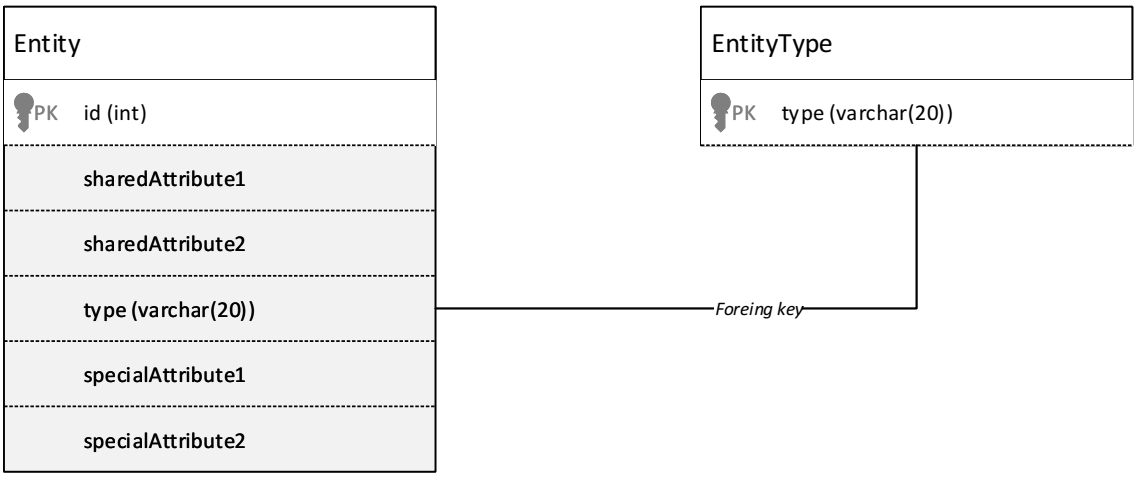
Taulukko 5. *Always Depend on One's Parent solmut*

Id	**parentId**	**name**
1	null	MainGroup1
2	null	MainGroup2
3	1	SubGroup1
4	1	SubGroup2
5	2	SubGroup1
6	4	SubGroup1
7	5	SubGroup1
8	5	SubGroup2
9	7	SubGroup1
10	8	SubGroup1

ADOP-antisuunnittelumallin mukainen rakenne vaatii ilman rekursiivista kyselyä aina SQL-koodiin taululle uuden liitoksen itseensä, kun solmujen määrä kasvaa puurakenteessa ja halutaan hakea juuren kaikki e t ”*Descendant*” tai lehden kaikki edelt ”*Ancestor*”. Solmun poistaminen puurakenteesta on hyvin raskas operaatio, koska kaikki lapset täytyy liittää solmun isään operaation yhteydessä. Solmun siirtäminen puurakenteessa vaatii myös uudessa sijainnissaan solmujen liitoksiin muutoksia. ADOP-antisuunnittelumallin käyttäminen saattaa vaatia myös ylimääräistä työtä ennen yksinkertaisten kyselyiden suorittamista (Karwin 2014).

3.6 Single Table Inheritance -suunnittelumalli

STI-suunnittelumalli ”*Single Table Inheritance*” on tapa toteuttaa periytymishierarkioita tietokannassa. STI-suunnittelumallissa toisiinsa liittyvät tyypit tallennetaan samaan tietokantatauluun. Periytetyt tyypit erotellaan toisistaan tyyppitunnisteena toimivan attribuutin arvolla, jonka arvojen ylläpitämiseen voidaan käyttää esimerkiksi erillistä tietokantataulua. Periytetyillä tyypeillä on tyyppikohtaisia attribuutteja, jotka eivät koske muita periytettyjä tyyppejä. Uusien periytettyjen tyyppien lisääminen tietokantatauluun vaatii usein myös uusia tyyppikohtaisia attribuutteja tietokantatauluun (Karwin 2014). Kuvassa 9 on esitelty STI-suunnittelumallin mukainen periytyminen.



Kuva 9. *Single Table Inheritance -antisuunnittelumalli*

STI-suunnittelumallille on ominaista, että tyypille kuulumattomat attribuutit jätetään arvoon NULL (Karwin 2014). Taulukossa 6 on esitelty kuvan 9 mukaisen Entity-taulun mahdolliset rivit, jossa tyypille kuulumattomat attribuutit on jätetty arvoon NULL.

Taulukko 6. *Single Table Inheritance rivit*

id	shared Attribute1	shared Attribute2	type	special Attribute1	special Attribute2
1	A	B	Type1	Type1 Attribute	NULL
2	B	A	Type2	NULL	Type2 Attribute

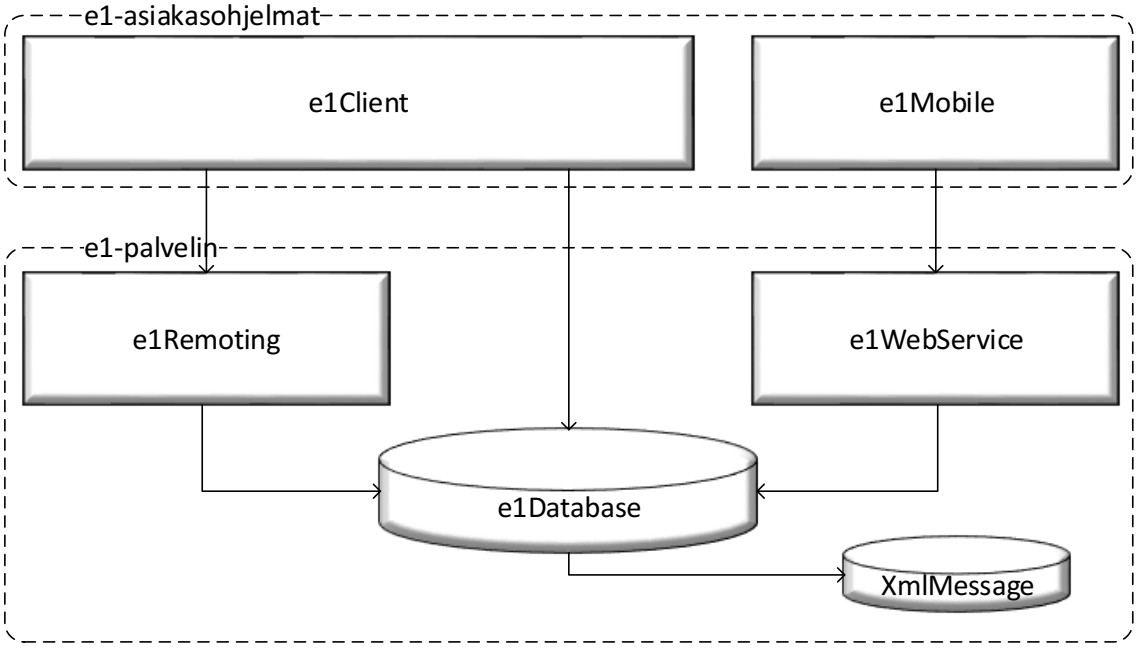
STI-suunnittelumallin käyttökelpoisuudelle on oleellista, että kaikilla tyypeillä on useita yhteisiä attribuutteja ja periytettyjen tyyppien määrä pysyy vähäisenä. Tyyppikohtaisten attribuuttien määrän on myös tärkeätä pysyä vähäisenä, koska periytetylle tyypille tietokannassa ei ole tietoa mille periytetylle tyypille erikoistetut attribuutit kuuluvat. Periytettyjen tyyppien tyyppikohtaisista attribuuteista huolehtiminen on kehittäjän vastuulla, joten suuret STI-taulut voivat olla vaikeasti ylläpidettäviä (Karwin 2014).

4. RAKENTEEN REKONSTRUOINTI

Rakenteen rekonstruointi tehdään paikalliseen e1-alijärjestelmään, joista on tällä hetkellä kolme eri versiota käytössä. Rakenne rekonstruoidaan ensin arkkitehtuuristen ratkaisujen mukaan. Tämän jälkeen tehdään e1-tietokannan rekonstruointi yksityiskohtaisemmin, koska ERP-järjestelmässä tietokanta on hyvin keskeisessä asemassa.

4.1 E1-alijärjestelmän kerrokset

E1-alijärjestelmässä on kaksi fyysisesti hajautettua kerrosta. Ylimmässä fyysisessä kerroksessa ovat e1Client- ja e1Mobile-asiakasohjelmat. e1-palvelimella on e1Remoting- ja e1WebService-palvelut, jotka muodostavat ohjelmallisen kerroksen e1-asiakasohjelmien ja e1-tietokannan välille. e1Client-asiakasohjelma käyttää myös suoraan e1Database-tietokannan palveluita. Kuvassa 10 on esitelty e1-alijärjestelmän fyysinen hajautus ja niiden ohjelmalliset osat.



Kuva 10. *e1-alijärjestelmän kerrokset*

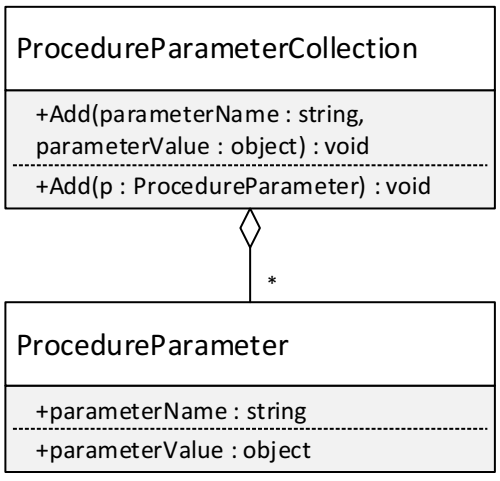
XmlMessage-tietokanta sisältää yhden tietokantataulun, joka pitää tallessa pyyntöä lähtevästä sanomasta. Taulu on identtinen kopio varsinaisen sanomatietokannan taulusta. Tietokannassa on ajastettu työ, joka kopioi uudet rivit tietokantataulusta sanomatietokannan tauluun.

Kuvan 10 nuolilla kuvataan käyttösuhteita ohjelmallisten osien välillä. Kerrosten väliset riippuvuudet esitellään aliluvussa 4.2 e1-alijärjestelmän osien esittelyn jälkeen. Rajapinnat esitellään luokkakaavioilla. Rajapinnan käyttö esitellään tarkemmin ohjelmakoodin pätkillä, koska metodien käyttöä ei voida suoraan ymmärtää luokkakaavioista.

4.1.1 e1Client

e1Client-asiakasohjelmalla hallinnoidaan esimerkiksi tuotantoyksikön tilauksia, varastoa, logistiikkaa, tuotantoprosessia, henkilöstöä ja raportointia. e1Client-asiakasohjelma on aluksi tehty NET-sovelluskehyksen versiolla 2.0, mutta myöhemmin tämä on siirretty .NET-sovelluskehyksen 3.5 versioon. e1Client-asiakasohjelma on toteutettu C#-ohjelmointikielellä, jossa käyttöliittymä on toteutettu Windows Forms -teknologialla. Käyttöliittymän elementit ovat sidottu DataSet-tietoaineistojen DataTable-taulujen sarakkeisiin. Näkymillä on tyypitettyjä tietoaineistoja, jotka sijaitsevat e1Client-asiakasohjelmassa. Osalla näkymistä ei ole tyypitettyä tietoaineistoa, jolloin näille luodaan dynaamisesti tietoaineisto, johon käyttöliittymän elementit sidotaan.

Näkymien pääasiallisena rajapintana e1-järjestelmän sovelluslogiikkaan on e1Remoting-palvelun BlBroker-välittäjäluokan ExecuteBusinessLogicProcedure-metodi, joka suorittaa tietokantaproseduurin. Tämä metodi vaatii applicationName-parametrin, joka kuvaa suoritettavan sovelluksen nimeä ja applicationMethodName-parametrin, joka kuvaa suoritettavan metodin nimeä. Parametrit applicationName ja applicationMethodName ovat molemmat merkkijonotyyppiä. Näiden lisäksi ExecuteBusinessLogicProcedure-metodi vaatii ProcedureParameterCollection-tyyppisen parametrin, jolla välitetään suoritettavalle tietokantaproseduurille kokoelma sen käyttämistä parametreista. Kuvassa 11 on esitelty parametrikokoelman luokkakaavio.

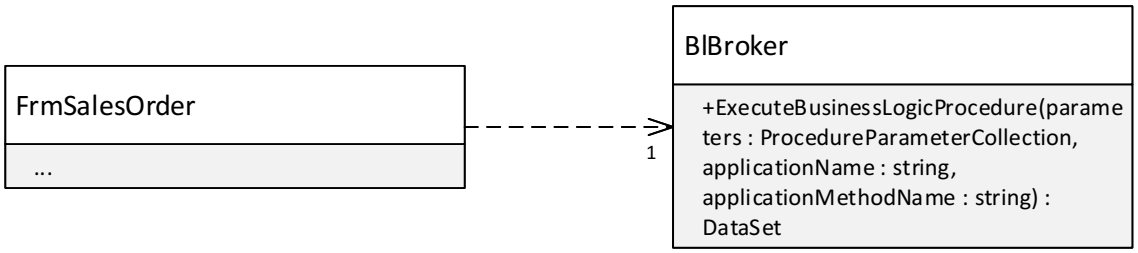


Kuva 11. *ProcedureParameterCollection-luokka*

ProcedureParameter-luokka koostuu kahdesta attribuutista, jossa parametrin nimeä pidetään tallessa merkkijonotyyppisessä ParameterName-attribuutissa ja arvoa pidetään tallessa object-kantaluokkatyyppisessä ParameterValue-attribuutissa. Parametrikokoelman

parametrien arvot ovat tyyppiriippumattomia ja usein yksi kokoelman parametreista on DataSet-tyyppi.

Parametrien applicationName ja applicationMethodName arvoilla määritellään minkä tietokantaproseduurin ExecuteBusinessLogicProcedure-metodi suorittaa. Kuvassa 12 on esitelty FrmSalesOrder-näkymän käyttämä rajapinta, jossa kaikki toiminnallisuudet tehdään ExecuteBusinessLogicProcedure-metodin kautta.



Kuva 12. *ExecuteBusinessLogicProcedure-rajapinta*

FrmSalesOrder-näkymä alustetaan kutsumalla ExecuteBusinessLogicProcedure-metodia applicationName-parametrin arvolla ”SalesOrder” ja applicationMethodName-parametrin arvolla ”Load”. Parametrikokoelmassa annetaan myyntitilauksen tunniste, jonka tiedot haetaan. Ohjelmassa 3 on esitelty ExecuteBusinessLogicProcedure-metodin käyttö FrmSalesOrder-näkymän alustamiseen, jossa myyntitilauksen tunnus on 1.

```
DataSetSalesOrder datasetSalesOrder = new DataSetSalesOrder();

ProcedureParameterCollection parameters =
new ProcedureParameterCollection();

int salesOrderId = 1;
parameters.Add(”SalesOrderID”, salesOrderId);

datasetSalesOrder.Merge(BlBroker.ExecuteBusinessLogicProcedure
(
  parameters,
  “SalesOrder”,
  “Load”
));
```

Ohjelma 3. *FrmSalesOrder-näkymän alustaminen*

Myyntitilauksen tallentaminen tehdään applicationName-parametrin arvolla ”SalesOrder” ja applicationMethodName-parametrin arvolla ”Save”. Parametrikokoelmassa tällöin annetaan myyntitilauksen tunnus salesOrderId ja FrmSalesOrder-näkymän tyypitettyyn DataSetSalesOrder-tietoaineistoon tehdyt muutokset. Tyypitetyt tietoaineistot muutetaan ennen parametrikokoelmaan lisäämistä DataSet-tyypiksi käyttöliittymän AddParameter-metodissa. DataSetSalesOrder-tietoaineistossa on 40 kappaletta erilaisia DataTable-tauluja myyntitilauksen käsittelyä varten.

Ohjelmassa 4 on esitelty myyntitilauksen tallentaminen ExecuteBusinessLogicProcedure-metodilla, jonka suoritus palauttaa dynaamisesti luodun tietoaineiston. Suorituksen jälkeen käyttöliittymän salesOrderId-attribuutin arvoksi sijoitetaan tietoaineiston TableResult-taulun ensimmäiseltä riviltä SalesOrderID-sarake.

```
ProcedureParameterCollection parameters =
new ProcedureParameterCollection();

Client.AddParameter(ref parameters, ”DataSet”, datasetSalesOrder.
GetChanges());

parameters.Add(”SalesOrderID”, salesOrderId)

DataSet ds = BlBroker.ExecuteBusinessLogicProcedure
(
  parameters,
  ”SalesOrder”
  ”Save”
);

SalesOrderID = (int)ds.Tables[”TableResult”].Row[0][”SalesOrderID”];
```

Ohjelma 4. *FrmSalesOrder-näkymän tallentaminen*

Myyntitilausnäkymässä käsitellään myös myyntitilauksesta erillisiä käsitteitä. Myyntitilausta kohden voi olla nolla tai enemmän tuotantoeriä, joita voidaan lisätä ja poistaa FrmSalesOrder-näkymässä. Tuotantoeriä voidaan lisätä tai poistaa myös muista näkymistä, joissa on erilaiset tyypitetyt tietoaineistot ja erilliset tietokantaproseduurit toimintoa varten.

FrmSalesOrder-näkymässä tuotantoerän lisääminen ensin tallentaa myyntitilauksen, jonka jälkeen tämä avaa uuden ikkunan tuotantoerän tietojen syöttämistä varten. Tuotantoerän poistaminen tehdään tallentamalla myyntitilaus ohjelman 4 mukaisesti. Tieto poistettavista tuotantoeristä lisätään ennen tallentamista DataSetSalesOrder-tietoaineiston tauluun TableRemovedBatchRows. Tuotantoerän poistaminen tallentaa samalla kaikki muutokset myös myyntitilaukseen, jos niitä on tehty ennen tuotantoerän poistamista.

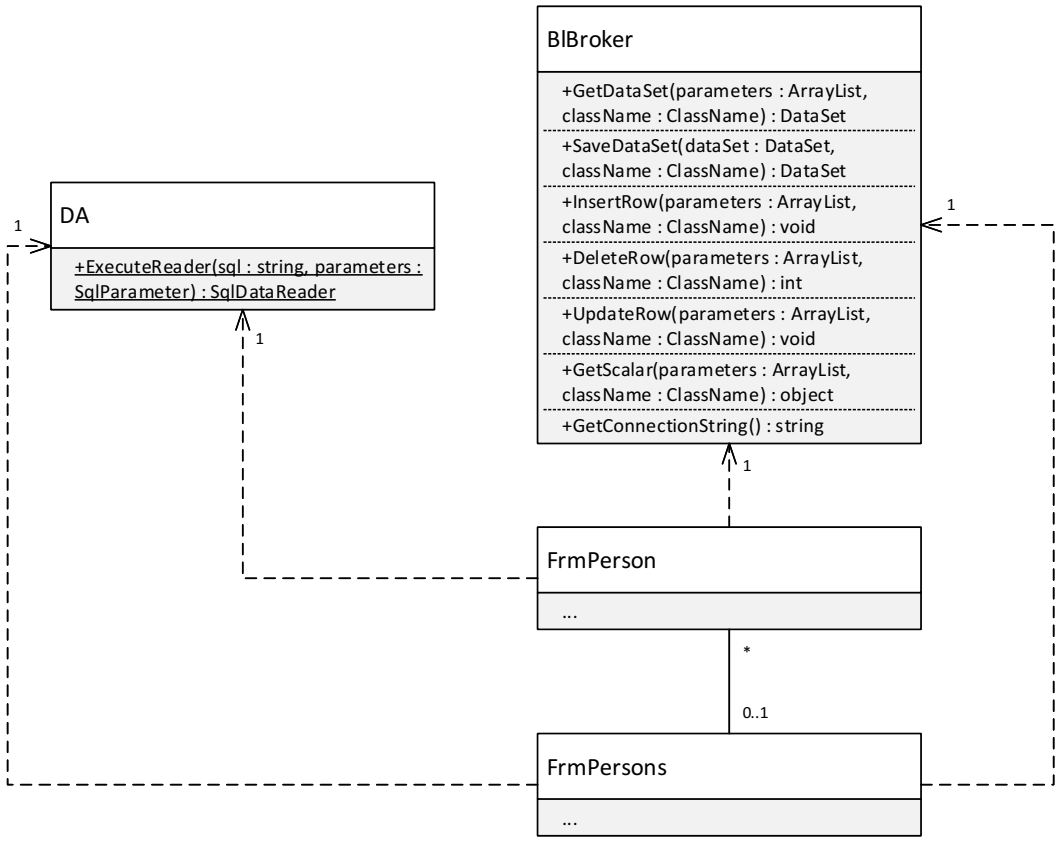
e1Client-asiakasohjelman näkymien vastuualueeseen kuuluu käyttöliittymän lisäksi osa sovelluslogiikan laskennasta, koska näkymät suorittavat sovelluslogiikan laskentaa e1Remoting-palvelun ExecuteBusinessLogicProcedure-metodikutsujen välillä. Näkymät tekevät esimerkiksi pieniä muutoksia tietoaineistoon tietokantaproseduurikutsujen välissä, jotka välitetään suorituksessa seuraavalle tietokantaproseduurille käyttöliittymästä. Joissakin näkymissä tietoaineistoa käsitellään kuin tietokantaa ja muutokset tehdään jo käyttöliittymässä tietoaineistoon. Tietoaineistossa uusien rivien pääavaimille annetaan usein arvoksi -1, jolloin koko tietoaineisto välitetään tietokantaproseduurille tallennettavaksi ExecuteBusinessLogicProcedure-metodilla.

e1Client-asiakasohjelman näkymistä osa ei käytä ollenkaan e1Remoting-palvelun ExecuteBusinessLogicProcedure-metodia. Nämä näkymät käyttävät rajapintana välittäjäluokan metodeja, joiden vaihtoehtoista toiminnallisuutta ohjataan numeerisella tyypillä ClassName. Ohjelmassa 5 on esitelty näkymien FrmPersons ja FrmPerson henkilöiden käsittelyyn tarkoitetut numeeriset tyypit.

```
enum ClassName
{
  Person,
  Persons,
  Person_PersonGroup,
  Person_PersonSkill,
  Person_PersonName,
  Person_AddWage,
  Person_RemoveWage
}
```

Ohjelma 5. *ClassName Person-arvot*

GetDataSet-metodi on tarkoitettu näkymän DataSet-tietoaineiston alustamiseen ja SaveDataSet-metodi tallentaa koko näkymän DataSet-tietoaineiston. InsertRow- ja UpdateRow-metodit ovat tarkoitettu näkymien käsitteiden käsittelyyn, joilla on vain yksi attribuutti näkymässä. GetScalar-metodilla haetaan käsitteiden yksittäisiä arvoja. DeleteRow-metodilla poistetaan käsitteitä näkymissä. Kuvassa 13 on esitelty FrmPerson- ja FrmPersons-näkymien rajapinta, jonka metodien toimintaa ohjataan numeerisen ClassName-tyypin arvoilla.



Kuva 13. *Sovelluslogiikan muu rajapinta*

Näkymät käsittelevät tietokantaa myös merkkijonoon tallennetuilla SQL-kyselyillä, jotka suoritetaan asiakasohjelmassa DA-luokan SQL-kyselyn suorittavalla ExecuteReader-metodilla, jolle e1Remoting-palvelu palauttaa tiedot tietokantayhteyttä varten GetConnectionString-metodilla.

Suurimmalle osalle e1Client-asiakasohjelman näkymistä on määritelty tyypitetty DataSet-tietoaineisto, jonka taulujen sarakkeisiin näkymän elementit sidotaan. FrmPerson- ja FrmPersons-näkymille ei ole määritelty tyypitettyä DataSet-tietoaineistoa asiakasohjelmassa, jolloin se rakennetaan dynaamisesti GetDataSet-metodin paluuarvosta. FrmPersons-näkymän alustaminen tehdään antamalla GetDataSet-metodin className-parametrin arvoksi ”Persons” ja parameters-listassa näkymän hakusuodattimet oikeassa järjestyksessä. Ohjelmassa 6 on esitelty FrmPersons-näkymän alustaminen.

```
ArrayList al = new ArrayList();

al.Add(radioButtonActivePersons.Checked);
al.Add(checkBoxSummerEmployee.Checked);

DataSet ds = BlBroker.GetDataSet(al, ClassName.Persons);
```

Ohjelma 6. *FrmPersons-näkymän alustaminen*

FrmPersons-näkymän alustaminen hakee kaikki henkilöryhmät ja niihin liitetyt henkilöt, jolloin DataSet-tietoaineistoon muodostuu relaatio henkilöryhmien ja henkilöiden välille.

FrmPersons-näkymässä uusi henkilöryhmä lisätään InsertRow-metodilla, jossa className-parametrin arvo on ”PersonGroup” ja parameters-listassa annetaan henkilöryhmän nimi. Ohjelmassa 7 on esitelty henkilöryhmän lisääminen.

```
ArrayList al = new ArrayList();

string name = ”Name”;
al.Add(name);

BlBroker.InsertRow(al, ClassName.PersonGroup);
```

Ohjelma 7. *FrmPersons henkilöryhmän lisääminen*

Henkilöryhmän nimi muutetaan UpdateRow-metodilla antamalla className-parametrin arvoksi ”PersonGroup” sekä parameters-listassa henkilöryhmän uuden nimen ja tunnisteen oikeassa järjestyksessä. Ohjelmassa 8 on esitelty henkilöryhmän nimen päivittäminen henkilöryhmälle, jonka personGroupId on 1.

```
ArrayList al = new ArrayList();

int personGroupId = 1;
al.Add(personGroupId);
string name = ”NewName”;
al.Add(name);

BlBroker.UpdateRow(al, ClassName.PersonGroup);
```

Ohjelma 8. *FrmPersons henkilöryhmän päivittäminen*

Henkilöryhmän poistaminen tehdään DeleteRow-metodilla antamalla className-parametrin arvoksi ”PersonGroup” ja parameters-listassa henkilöryhmän tunniste. Ohjelmassa 9 on esitelty henkilöryhmän poistaminen, jonka personGroupId on 1.

```
ArrayList al = new ArrayList();

int personGroupId = 1;
al.Add(personGroupId);

BlBroker.DeleteRow(al, ClassName.PersonGroup);
```

Ohjelma 9. *FrmPersons henkilöryhmän poistaminen*

Henkilön poistaminen tehdään myös FrmPersons-näkymässä samoin kuin henkilöryhmän poistaminen. className-parametrin arvo on tällöin ”Person” ja parameters-listassa annetaan henkilön tunniste.

FrmPerson-näkymä alustetaan antamalla GetDataSet-metodille className parametrin arvoksi ”Person” ja parameters-listassa henkilön tunnisteen. Ohjelmassa 10 on esitelty FrmPerson-näkymän alustaminen henkilölle, jonka personId on 1.

```
ArrayList al = new ArrayList();

int personId = 1;
al.Add(personId);

DataSet ds = BlBroker.GetDataSet(al, ClassName.Person);
```

Ohjelma 10. *FrmPerson-näkymän alustaminen*

FrmPerson-näkymän alustaminen hakee henkilön tiedot, vakanssit ja palkkalajit, jolloin nämä muodostavat FrmPerson-näkymälle dynaamisesti tietoaineiston. Henkilön kohdistamattomat vakanssit haetaan uudelleen näkymän alustuksen yhteydessä suoralla SQL-kyselyllä, jonka avulla estetään henkilölle saman vakanssin lisääminen uudelleen. Suorat SQL-kyselyt suoritetaan e1Client-asiakasohjelman DA-luokan ExecuteReader-metodilla. Ohjelmassa 11 on esitelty henkilön vakanssien hakeminen henkilölle, jonka personId on 1.

```
int personId = 1;
string sql = @”SELECT ...”;

SqlDataReader dr =
DA.ExecuteReader(sql, new SqlParameter(”@personID”, personId));
```

Ohjelma 11. *FrmPerson henkilön vakanssien hakeminen*

FrmPerson-näkymässä haetaan suoralla SQL-kyselyllä myös tietokannan kaikki vakanssit comboBox-elementin listausta varten. FrmVacancies-näkymässä haetaan tietokannan kaikki vakanssit GetDataSet-metodilla antamalla className-parametrin arvoksi ”Vacancies”. Metodeilla on täysin sama toiminnallisuus, mutta metodeja ei käytetä uudelleen, koska GetDataSet-metodi palauttaa näkymän tietoaineiston.

Uusi henkilö lisätään e1-alijärjestelmään SaveDataSet-metodilla antamalla className-parametrin arvoksi ”Person” ja dataSet-parametrina GetDataSet-metodin dynaamisesti luoma tietoaineisto. Henkilö päivitetään myös SaveDataSet-metodilla antamalla className-parametrin arvoksi ”Person” ja dataSet-parametrina dynaamisesti luotu tietoaineisto. Henkilölle myös lisätään uusi vakanssi SaveDataSet-metodilla samalla kun henkilö lisätään tai päivitetään. Ohjelmassa 12 on esitelty FrmPerson-näkymän tallentaminen. SaveDataSet-metodi palauttaa tietoaineiston, josta haetaan henkilön personId.

```
ds = BlBroker.SaveDataSet(ds, ClassName.Person);

if (personId == -1)
{
  personId = (int)ds.Tables[0].Rows[0][”personId];
}
```

Ohjelma 12. *FrmPerson-näkymän tallentaminen*

Henkilön vakansseja poistetaan kutsumalla deleteRow-metodia antamalla ClassName-parametrin arvoksi ”Person_PersonSkill”. Henkilön vakanssien poistaminen tehdään

henkilön ja henkilöryhmän poistamisesta poikkeavasti, koska yksittäisen rivin poistamiseen tarkoitetulla metodilla poistetaan yksi tai useampi rivi kerrallaan. Parametrilistassa annetaan tietokantataulujen tblPerson ja tblVacancy välissä käytetyn tblPersonSkill-liitostaulun pääavaimet. Nämä annetaan pilkuilla eroteltuna yhdessä merkkijonossa, jotta tämä voidaan upottaa SQL-kyselyn IN-lauseen sisälle. Merkkijono rakennetaan for-silmukan avulla, mutta metodikutsun yksinkertaistamisen vuoksi merkkijono esitellään valmiiksi rakennettuna. Ohjelmassa 13 on esitelty henkilön vakanssien poistaminen, jossa tblPersonSkill-taulun pääavaimien arvot ovat 1, 2 ja 3.

```
string ids = ”1, 2, 3”;

ArrayList al = new ArrayList();
al.Add(ids);

BlBroker.DeleteRow(al, ClassName.Person_PersonSkill);
```

Ohjelma 13. *FrmPerson henkilön vakanssien poistaminen*

Henkilön etunimen ja sukunimen yhdistelmä haetaan FrmVacation-näkymässä kutsumalla GetScalar-metodia antamalla className-parametrin arvoksi ”Person_PersonName” ja parameters-listassa henkilön tunniste. Samaa metodia kuin FrmPerson-näkymässä henkilön tietojen hakemiseen ei siis käytetä. Ohjelmassa 14 on esitelty henkilön etunimen ja sukunimen yhdistelmän hakeminen otsikkoa varten FrmVacation-näkymässä.

```
int personId = 1;
ArrayList al = new ArrayList();
al.Add(personId);

string personName =
BlBroker.GetScalar(al, ClassName.Person_PersonName).ToString();
```

Ohjelma 14. *FrmPerson henkilön etunimen ja sukunimen yhdistelmä*

Henkilöillä on myös palkkalajeja, joita voidaan valita käyttöön FrmPerson-näkymästä. Tietokannan kaikki palkkalajit haetaan vakanssien tavoin alustamisen yhteydessä suoralla SQL-kyselyllä. Henkilölle lisätään palkkalaji InsertRow-metodilla antamalla className-parametrin arvoksi ”Person_AddWage” ja parameters-listassa henkilön tunniste sekä palkkalajin tunniste. Ohjelmassa 15 on esitelty palkkalajin lisääminen henkilölle.

```
int personId = 1;
int wageId = 1;

ArrayList al = new ArrayList();
al.Add(personId);
al.Add(wageId);

BlBroker.InsertRow(al, ClassName.Person_AddWage);
```

Ohjelma 15. *FrmPerson palkkalajin lisääminen henkilölle*

Henkilön palkkalaji poistetaan DeleteRow-metodilla antamalla className-parametrin arvoksi ”Person_RemoveWage”. Palkkalajin poistaminen tehdään myös poikkeavasti, koska poistamisessa ei käytetä liitostaulun pääavainta. Henkilön ja palkkalajin tunnisteet annetaan parameters-listassa. Ohjelmassa 16 on esitelty palkkalajin poistaminen henkilöltä.

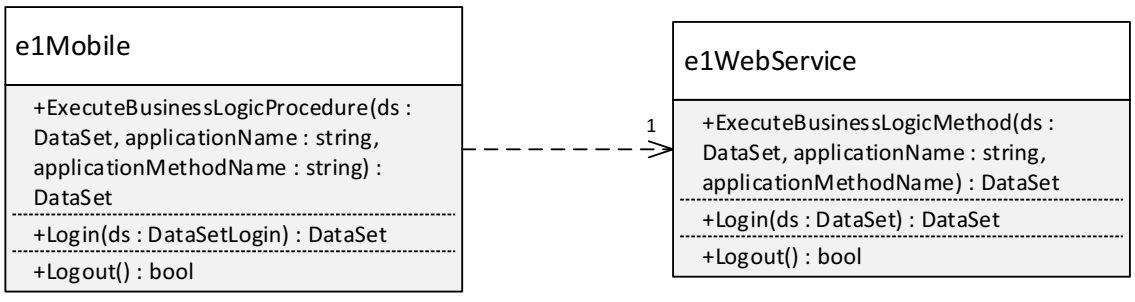
```
int personId = 1;
int wageId = 1;

ArrayList al = new ArrayList();
al.Add(personId);
al.Add(wageId);

BlBroker.DeleteRow(al, ClassName.Person_AddWage);
```

Ohjelma 16. *FrmPerson palkkalajin poistaminen henkilöltä*

Vertailtaessa henkilön vakanssien ja palkkalajin poistamista voidaan nähdä, että DeleteRow-metodi poistaa tässä tapauksessa useita rivejä ja parameters-listassa annetaan liitostaulun pääavaimet. Palkkalajin poistamisesta voidaan nähdä, että DeleteRow-metodilla poistetaan yksittäinen arvo liitostaulusta henkilön ja palkkalajien tunnisteiden perusteella.

4.1.2 e1Mobile

e1Mobile-asiakasohjelmaa käytetään apuna varastonhallinnassa, kuten kirjaamaan järjestelmään saapuneet ja valmistuneet tuotteet, siirtämään tuotteita varastossa sekä yhdistämään osatuotteita. e1Mobile on C#-ohjelmointikielellä toteutettu asiakasohjelma Windows CE -ympäristöön. Käyttöliittymä on toteutettu myös Windows Forms -teknologialla. Käyttöliittymän elementit ovat myös sidottu näkymälle tarkoitetun tietoaineiston DataTable-taulujen sarakkeisiin. Näkymille tarkoitetut tietoaineistot sijaitsevat e1Mobile-asiakasohjelmassa. Kuvassa 14 on esitelty e1Mobile-asiakasohjelman käyttämä rajapinta.



Kuva 14. *e1Mobile-rajapinta*

Sisäänkirjautumista varten on Login-metodi, jolle annetaan parametrina FrmLogin-näkymän tyypitetty DataSetLogin-tietoaineisto. e1Mobile-asiakasohjelman sisäinen Login-metodi muuttaa tyypitetyn DataSetLogin-tietoaineiston DataSet-tyypiksi, joka tämän jäl-

keen välitetään parametrina e1WebService-palvelun Login-metodille. e1Mobile-asiakasohjelman Logout-metodi kirjaa käyttäjän ulos asiakasohjelmasta ja tämän jälkeen kutsuu e1WebService-palvelun Logout-metodia.

Kaikkeen muuhun toiminnallisuuteen on käytetty e1Mobile-asiakasohjelman ExecuteBusinessLogicProcedure-metodia, jolle annetaan parametrina näkymän tyypitetty DataSet-tietoaineisto. e1Remoting-palvelun ExecuteBusinessLogicProcedure-metodin tavoin applicationName- ja applicationMethodName-parametrit määrittävät mikä tietokantaproseduuri suoritetaan.

Lavojen varastopaikan siirtämisessä käytetään FrmContainerMove-näkymää, joka alustetaan kutsumalla ExecuteBusinessLogicProcedure-metodia, jossa applicationName-parametrin arvo on ”MobileContainerMove” ja applicationMethodName-parametrin arvo on ”Load”. Alustettaessa näkymää TableContainer- ja TableParameter-taulut ovat tyhjiä. Luettaessa viivakoodi TableContainer-taulun sisältö yhdistetään tietoaineistoon ja annetaan parametrina samalle tietokantaproseduurille, joka alustaa näkymän uudelleen ja hakee lavan tiedot. Varastopaikan valitsemisen yhteydessä varastopaikka on TableParameter-taulussa, jolloin näkymä alustetaan myös uudelleen ja haetaan varastopaikkaan liittyvää tietoa. Ohjelmassa 17 on esitelty metodin käyttöä siirtonäkymän alustuksessa.

```
DataSetStorageMove dataSetStorageMove = new DataSetStorageMove();

DataSetStorageMove tempDs = new DataSetStorageMove();
tempDs.Merge(dataSetStorageMove.TableContainer);
tempDs.Merge(dataSetStorageMove.TableParameter);

DataSet ds = ExecuteBusinessLogicProcedure(tempDs,
”MobileContainerMove”, ”Load”);

if (!ds.Tables.Contains(”TableError”))
{
  dataSetStorageMove.Merge(ds);

}
```

Ohjelma 17. *e1Mobile siirtonäkymän alustaminen*

ExecuteBusinessLogicProcedure-metodi antaa e1Mobile-asiakasohjelmalle paluuarvona aina DataSet-tietoaineiston, joka yhdistetään näkymän tyypitettyyn DataSet-tietoaineistoon. Mikäli palautettu DataSet-tietoaineisto sisältää TableError-taulun, niin toiminto on tällöin epäonnistunut.

Näkymässä tehdyt muutokset tallennetaan antamalla näkymän tyypitetty DataSet-aineisto parametrina ExecuteBusinessLogicProcedure-metodille. Ohjelmassa 18 on esitelty siirtonäkymässä tehtyjen muutoksien tallentaminen kutsumalla ExecuteBusinessLogicProcedure-metodia applicationName-parametrin arvolla ”MobileContainerMove” ja applicationMethodName-parametrin arvolla ”Save”.

```
dataSetStorage.AcceptChanges();

DataSet ds = ExecuteBusinessLogicProcedure(dataSetStorageMove,
”MobileContainerMove”, ”Save”);
```

Ohjelma 18. *e1Mobile siirtonäkymän tallentaminen*

ExecuteBusinessLogicProcedure-metodi muuttaa tyypitetyt tietoaineistot DataSet-tyypiksi ja välittää tämän jälkeen parametrit e1WebService-palvelun ExecuteBusinessLogicMethod-metodille. e1Mobile-asiakasohjelman ja e1WebService-palvelun kaikki viestintä tehdään tietoaineistojen kautta.

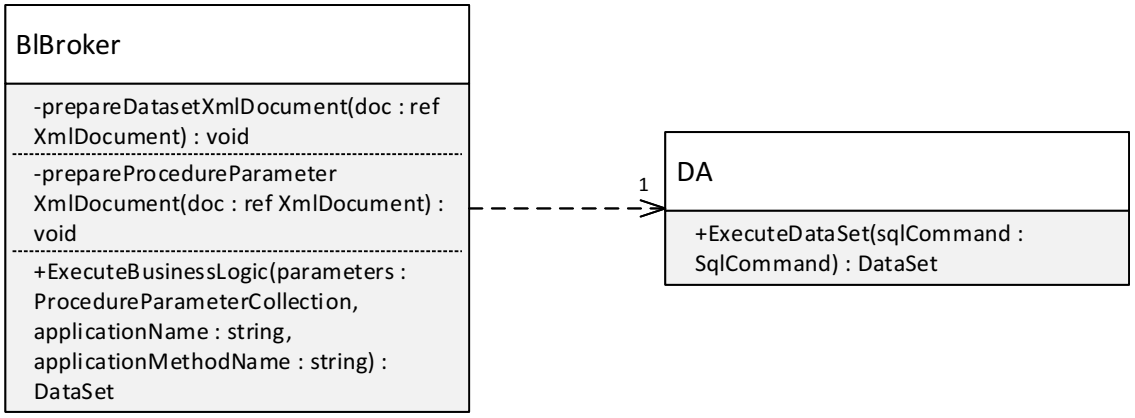
4.1.3 e1Remoting

e1Remoting on C#-ohjelmointikielellä toteutettu Windows Service, joka tarjoaa rajapinnan e1Client-asikasohjelmalle BlBroker-välittäjäluokan kautta. e1Client-asiakasohjelma kutsuu etänä BlBroker-luokan metodeja käyttäen .NET Remoting -rajapintaa.

Suurin osa näkymistä käyttää vain e1Remoting-palvelun ExecuteBusinessLogicProcedure-metodia, joka suorittaa tietokantaproseduurin. Tietokantaproseduurien nimien alkuun täytyy laitaa tunniste ”spBl”, jotta tietokantaproseduuri voidaan suorittaa ExecuteBusinessLogicProcedure-metodin kautta. Parametrina annettu applicationName on jokin osa tietokantaproseduurin nimestä tunnisteen ”spBl” jälkeen ja parametrina annettu applicationMethodName on jokin osa tietokantaproseduurin nimen lopusta.

Päivittäessä myyntitilaus valmistuneeksi annetaan applicationName-parametrin arvoksi ”Packing”. e1- rjestelm ”FinnishPacking” tarkoittaa, että pakkaaminen on viimeistelty, jolloin applicationMethodName-parametrin arvoksi annetaan ”FinnishPacking”. Suoritettava tietokantaproseduuri on tällöin ”spBlPackingFinnishPacking”.

ExecuteBusinessLogicProcedure-metodi lisää tunnisteen ”spBl” perään parametreissa applicationName ja applicationMethodName annetut merkkijonot. Tietokantaproseduuria ”spBlPackingFinnishPacking” voidaan kutsua myös antamalla esimerkiksi parametrin applicationName arvoksi ”PackingFinnishPacking” ja applicationMethodName arvoksi tyhjän merkkijonon. Kuvassa 15 on esitelty luokkakaavio e1Remoting-palvelun ExecuteBusinessLogicProcedure-metodin suorituksesta.

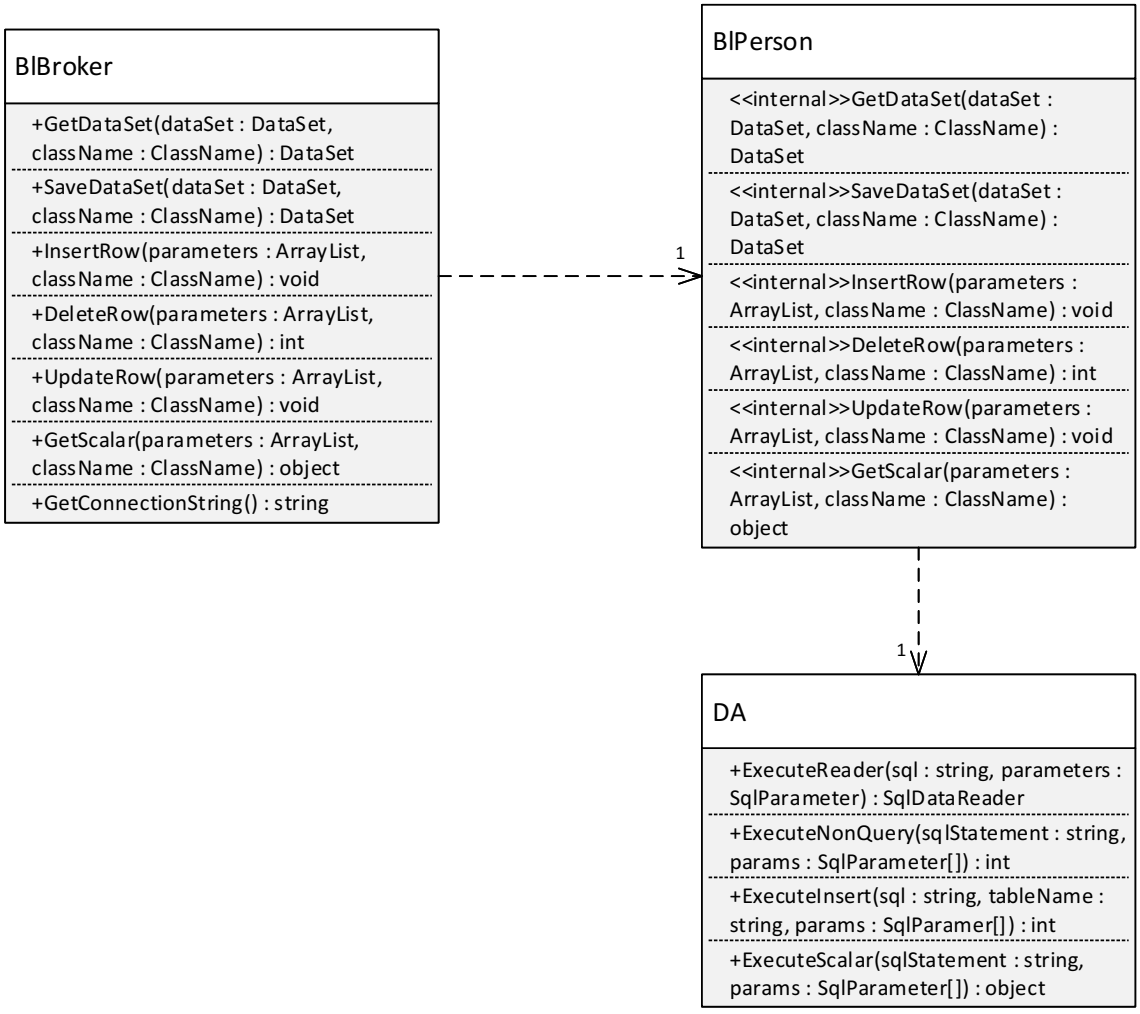


Kuva 15. *ExecuteBusinessLogicProcedure-metodin suoritus*

ExecuteBusinessLogicProcedure-metodi etsii tietokantaproseduurin nimen rakentamisen jälkeen ensimmäisenä parametrikokoelmasta DataSet-tyypin, jonka jälkeen tämä muutetaan XML-dokumentiksi BlBroker-luokan sisäisellä prepareDatasetXmlDocument-metodilla. Tämän jälkeen DataSet-tyyppinen parametri poistetaan kokoelmasta, jonka jälkeen parametrikokoelma muutetaan XML-dokumentiksi sisäisellä prepareProcedureParameterXmlDocument-metodilla. Metodit prepareDataSetXmlDocument- ja prepareProcedureParameterXmlDocument muokkaavat nimiavaruuksia ja aikaleimojen formaattia XML-dokumentin rakentamisen yhteydessä.

Lopulta nämä parametrit välitetään tietokantaproseduurille, joka suoritetaan e1Remoting-palvelun DA-luokan ExecuteDataSet-metodilla. ExecuteBusinessLogicProcedure-metodi antaa paluuarvona DataSet-tietoaineiston, joka muodostuu dynaamisesti tietokantaproseduurin paluuarvosta.

e1Remoting-palvelu sisältää osalle näkymistä sovelluslogiikan toteutuksen, joihin BlBroker-välittäjäluokka ohjaa suorituksen className-parametrilla. FrmPerson- ja FrmPersons-näkymien sovelluslogiikan toteutus on BlPerson-luokassa. Kuvassa 16 on esitelty BlBroker-välittäjäluokan henkilöiden käsittelyyn tarkoitettujen metodien ohjautuminen e1Remoting-palvelussa.



Kuva 16. *Henkilöiden käsittelyyn tarkoitettujen metodien suoritus*

BlPerson-luokan metodeja ei voida kutsua ilman BlBroker-välittäjäluokkaa, koska metodit ovat näkyvyysalueeltaan projektin sisäisiä. BlPerson-luokan metodit ovat täysin identtiset BlBroker-luokan vastaavien metodien kanssa, koska metodien vaihtoehtoinen toiminnallisuus ohjataan vasta BlPerson-luokan metodien sisällä. Myös muiden Bl-luokkien metodit ovat täysin identtiset BlBroker-luokan metodeihin nähden. e1Remoting-palvelussa Bl-luokkia on 50 kappaletta ja className-parametrin arvoja on 300.

4.1.4 e1WebService

e1WebService-palvelu tarjoaa kolme Web-metodia, joilla e1Mobile-asiakasohjelma kommunikoi e1-alijärjestelmän tietokannan kanssa. e1WebService on erillinen palvelu e1Remoting-palvelusta. ExecuteBusinessLogicMethod-metodi rakentaa suoritettavan tietokantaproseduurin nimen e1Remoting-palvelun ExecuteBusinessLogicProcedure-metodin tavoin. DataSet-tietoaineisto muutetaan XML-dokumentiksi, joka annetaan tietokansaproseduurille dataSetXml-parametrin arvoksi.

Siirtonäkymässä muutetaan lavojen varastopaikkoja, jolloin tietokantaan välitetään lavan tunniste ja uusi varastopaikka. Tämä tieto välitetään tietokantaan muutaman kilotavun kokoisena XML-dokumenttina.

4.1.5 e1Database

e1Database-tietokannan vastuualueina on paikallisen e1-alijärjestelmän resurssisuunnittelun data. e1Database-tietokannassa on myös suurin osa sovelluslogiikasta, joka on toteutettu tietokantaproseduureilla, joita on yhteensä 670 kappaletta. Tietokantafunktioita on 80 kappaletta. e1Client-asikasohjelmasta suoritukseen pyydetyt tietokantaproseduurit e1Remoting-palvelun kautta eivät ole aina ainoa suoritettava ohjelma, koska usein nämä kutsuvat tietokantafunktioita suorituksensa aikana. Tietokantafunktioita kutsutaan vain e1Database-tietokannan sisäisesti tietokantaproseduurien ohjelmakoodista.

e1Client-asiakasohjelmasta suoritettavissa tietokantaproseduureissa on kaksi parametria. parametersXml-parametri ottaa vastaan XML-muodossa parametrikokoelman ja se on suurimmassa osassa tietokantaproseduureja 4000 merkkiä pitkä varchar-tyyppi. Joissakin tietokantaproseduureissa tämän pituus on varchar-tyypin maksimi. dataSetXml-parametri ottaa vastaan käyttöliittymän DataSet-tietoaineiston XML-muodossa. Ohjelmassa 19 on esitelty spBlSalesOrderSave-tietokantaproseduurin parametrit.

```
CREATE PROCEDURE spBlSalesOrderSave
(
  @parametersXml VARCHAR(4000),
  @dataSetXml TEXT
)
```

Ohjelma 19. *spBlSalesOrderSave-tietokantaproseduurin parametrit*

Parametrikokoelman parametreja varten tehdään väliaikainen parametritaulu, johon määritellään tietokantaproseduurissa maksimipituudet parametrien nimille ja arvoille. Suurimmassa osassa proseduureja maksimipituus parametrin nimelle ja arvolle on 255 merkkiä. Ohjelmassa 21 on esitelty väliaikaisen parametritaulun määrittely.

```
DECLARE @parameters TABLE
(
  parameterName VARCHAR(255),
  parameterValue VARCHAR(255)
)
```

Ohjelma 20. *Väliaikainen parametritaulu*

Tämän jälkeen parametrit luetaan parametersXml-parametrista SELECT-lauseella ja syötetään väliaikaiseen parametritauluun INSERT-lauseella. Ohjelmassa 21 on esitelty parametrien lukeminen väliaikaiseen tauluun.

```
INSERT INTO @parameters
SELECT
  ParameterName,
  ParameterValue
FROM OPENXML (@rdoc, 'ArrayOfProcedureParameter/ProcedureParameter', 1)
WITH
(
  ParameterName VARCHAR(255) 'ParameterName',
  ParameterValue VARCHAR(255) 'ParameterValue'
)
```

Ohjelma 21. *Parametrien lukeminen väliaikaiseen tauluun*

Parametreja varten tietokantaproseduuriin määritellään paikalliset muuttujat, johon ne sijoitetaan tietokantaproseduurin sovelluslogiikan laskentaa varten. Sijoituksen yhteydessä tehdään tarvittavat tyyppimuunnokset. Ohjelmassa 22 on esitelty parametrin sijoittaminen paikalliseen muuttujaan.

```
DECLARE @salesOrderID INT
SELECT
  @salesOrderID = CAST(parameterValue AS INT)
FROM @parameters
WHERE parameterName = 'SalesOrderID'
```

Ohjelma 22. *Parametrin tallentaminen paikalliseen muuttujaan*

Parametrikokoelmassa voi olla useita kymmeniä eri nimisiä parametreja mukana. dataSetXml-parametria käytetään usean rivin käsittelyyn samalta entiteetiltä. Esimerkiksi spBlSalesOrderSave-proseduuri poistaa myyntitilaukselta yhden tai useamman tuotantoerän, jos nämä löytyvät dataSetXml-parametrista. Ohjelmassa 23 on esitelty poistettavien tuotantoerien hakeminen dataSetXml-parametrista.

```
INSERT INTO @tableDeleteBatchRow
(
  batchRowID,
  orderID
)
SELECT
  batchRowID,
  orderID
FROM
(
  SELECT
    [a:BatchRowID] batchRowID,
    [a:OrderID] orderID
  FROM OPENXML (@rDoc, '//a:TableRemovedBatchRows', 2)
  WITH
  (
    [a:BatchRowID] INT,
    [a:OrderID] INT
  )
)
```

Ohjelma 23. *Arvojen hakeminen dataSetXml-parametrista*

dataSetXml-parametrin rakenteesta ei ole mitään mallia, koska XML-dokumentti generoidaan e1RemotinService-palvelussa ajoaikana näkymän tietoaineistosta. Tietokantaproseduurissa virheellistä tai puutteellista rakennetta käsitellään arvona NULL, joka ei päätä tietokantaproseduuria virheeseen. Ohjelmassa 24 on esitelty tietoaineiston liittäminen tietokantatauluun.

```
  [a:SalesOrderRowID],
  ISNULL([a:RearrangeBatch], 0),
  iv.itemVersionID,
  i.askQuantity
FROM OPENXML (@rDoc, '//a:TableBatchRows', 2)
WITH
(
  HasChanges varchar(20) 'diffgr:hasChanges'
  [a:SalesOrderRowID] INT,
  [a:ItemVersionID] INT,
  [a:RearrangeBatch] BIT
)
INNER JOIN itemVersion iv ON [a:ItemVersionID] = iv.itemVersionID
INNER JOIN item i ON iv.itemID = i.itemID
WHERE HasChanges = 'Inserted'
```

Ohjelma 24. *Tietoaineiston liittäminen tietokantatauluun*

Tietokantaproseduureissa käytetään hyvin paljon tietoaineistoja XML-muodossa ja usein niitä liitetään tietokantatauluihin SQL-kyselyissä. Tietokantaproseduureissa käsiteltävät tietoaineistot XML-muodossa voivat olla kooltaan 10 megatavua.

4.2 Kerroksien riippuvuudet

Edellisessä luvussa esiteltiin e1-alijärjestelmän osien toiminnallisuutta ja vastuualueita, jonka pohjalta tehdään riippuvuusmatriisit väliltä e1Client-asiakasohjelma ja e1Database-tietokanta sekä e1Mobile-asiakasohjelma ja e1Database-tietokanta. Taulukossa 7 on esitelty riippuvuusmatriisi e1Client-asiakasohjelman ja e1Database-tietokannan väliltä.

Taulukko 7. *e1Client - e1Database riippuvuusmatriisi*

Rivikerros riippuu sarakekerroksesta	**e1Client**	**e1Remoting**	**e1Database**
e1Client		X	X
e1Remoting	X		X
e1Database	X	X	

Tietokantaproseduurit ovat riippuvaisia käyttöliittymän tyypitetyistä tietoaineistoista, koska sarakkeen nimen muuttaminen käyttöliittymässä vaatii myös tietokantaproseduurin toiminnan muuttamista. e1Client-asiakasohjelman käyttöliittymä on riippuvainen e1Database-tietokannasta, koska proseduurikutsujen yhteydessä täytyy tietää suoritettavan tie-

tokantaproseduurin nimi. Lisäksi osa e1Client-asiakasohjelman näkymistä viittaa tietokantaproseduurin tai muun sovelluslogiikan dynaamisesti generoimaan tietoaineistoon. Näkymät myös käsittelevät merkkijonoihin tallennetuilla SQL-kyselyillä suoraan tietokantaa. Käyttöliittymässä täytyy myös tuntea tietokantaproseduurin sisältämä toiminnallisuus, jotta voidaan tietää, että esimerkiksi SalesOrderSave-proseduuri tallentamisen lisäksi poistaa myös tuotantoeriä.

e1Remoting-palvelun Bl-luokat ovat myös riippuvaisia näkymien tyypitetyistä tietoaineistoista, koska DataTable-taulujen sarakkeiden nimen muuttaminen käyttöliittymässä vaatii myös Bl-luokan toteutuksen muuttamista. e1Database-tietokanta on riippuvainen e1Remoting-palvelun parametrikokoelmasta generoiduista XML-dokumenteista.

e1Mobile-asiakasohjelma on täysin erillinen e1Client-asiakasohjelmasta ja e1WebService-palvelu on täysin erillinen e1Remoting-palvelusta, joten riippuvuudet esitellään erikseen. Taulukossa 8 on esitelty riippuvuusmatriisi e1Mobile-asiakasohjelman ja e1Database-tietokannan välillä.

Taulukko 8. *e1Mobile – e1Database riippuvuusmatriisi*

Rivikerros riippuu sarakekerroksesta	**e1Mobile**	**e1WebService**	**e1Database**
e1Mobile		X	X
e1WebService	X		X
e1Database	X	X	

e1Mobile-asiakasohjelma on riippuvainen e1WebService-palvelusta. e1Mobile on myös riippuvainen e1Database-tietokannasta, koska e1Mobile viittaa tietokantaproseduurin paluuarvosta dynaamisesti generoituun tietoaineistoon.

e1WebService on riippuvainen e1Database-tietokannasta. e1WebService on myös riippuvainen e1Mobile-asiakasohjelmasta, koska tämä viittaa suotaan FrmLogin-näkymän tyypitetyn DataSetLogin-tietoaineiston TableLogin-tauluun. DataSetLogin-tietoaineiston muuttaminen vaatii myös e1WebService-palvelun muuttamista.

e1Mobile-asiakasohjelman tietokantaproseduureissa käytetään vain dataSetXml-parametria, jolloin kaikki toiminnallisuus tietokantaproseduureissa riippuu näkymän tyypitetyistä tietoaineistoista. Tietokantaproseduurit ovat myös riippuvaisia e1WebService-palvelun XML-dokumentin muodosta.

E1-alijärjestelmä voidaan jakaa 3-kerrosarkkitehtuurin mukaisiin kerroksiin, niiden näkymien osalta joiden sovelluslogiikka on toteutettu e1Remoting-palvelun Bl-luokissa. Tällöin e1Client-asiakasohjelma on käyttöliittymäkerroksessa, e1Remoting-palvelu on sovelluslogiikkakerroksessa ja e1Database-tietokanta on tietokantakerroksessa. Näkymät

käyttävät myös asiakas-palvelin-arkkitehtuurin mukaista rakennetta suorien tietokantakyselyiden osalta.

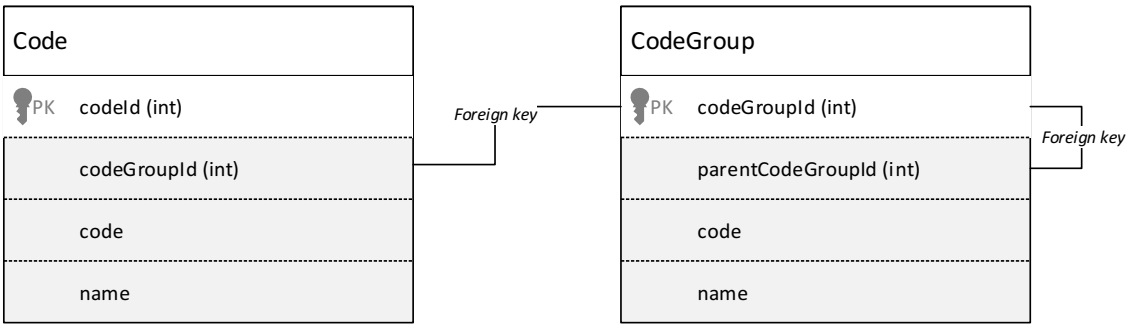
E1-alijärjestelmää ei voida jakaa ExecuteBusinessLogicProcedure-metodia käyttävien näkymien osalta 3-kerrosarkkitehtuurin mukaisiin kerroksiin, koska e1Remoting-palvelun vastuualueena on tällöin XML-dokumentin rakentaminen ja tietokantaproseduurin suorittaminen. Sovelluslogiikka on näiden näkymien osalta hajautettu e1Client-asiakasohjelman näkymiin ja tietokantaproseduureihin, joka vastaa aliluvussa 3.1 esiteltyjen asiakas-palvelin-arkkitehtuurin ongelmien yhdistelmää.

4.3 E1-tietokannan rakenteelliset ratkaisut

E1-tietokannassa suurimpaan osaan tietokantatauluista on määritelty, jokin kokonaislukutyyppinen surrogaattiavain pääavaimeksi. Vierasavaimia on käytetty joidenkin taulujen liitoksissa, mutta tietokantaproseduureissa tehdään hyvin paljon taulujen välisiä liitoksia ilman vierasavainta. Entiteetit käyttävät hyvin paljon tyyppitunnisteita, jotka viittaavat Code-taulun pääavaimeen, koska e1Database-tietokannan tauluissa käytetään aliluvussa 3.6 esiteltyä STI-periytymistä.

4.3.1 E1 SLT-taulu

Code-taulu on e1Database-tietokannan SLT-taulu, joka pitää tallessa arvoja kaikkien entiteettien tyypeistä ja tiloista. CodeGroup-taulu erottelee koodit entiteettikohtaisesti codesarakkeen arvoilla, joiden suomennokset esitetään name-sarakkeen arvoilla. Code-taululla on vierasavainviittaus CodeGroup-tauluun, jolla on myös vierasavainviittaus itseensä parentCodeGroupId-sarakkeessa. Muilla tauluilla ei ole viittauksia CodeGroup-tauluun. Kuvassa 17 on esitelty tietokantakaavio Code-taulun ja CodeGroup-taulun muodostamasta tyyppijärjestelmästä.



Kuva 17. *e1 tyyppijärjestelmä*

E1-järjestelmässä entiteetillä on usein jokin tyyppi, jolla se erotellaan muista saman taulun käsitteistä. Ostotilaukset ja tuotantotilaukset ovat Order-taulussa ja ne erotellaan toisistaan tyyppitunnisteella, joka on liitetty codeGroup-taulussa entiteetin tyypin ryhmään.

CodeGroup-taulussa käytetään aliluvun 3.5 mukaista puurakennetta, jossa juurisolmulla parenCodetGroupId-sarakkeen arvo on NULL.

Juurisolmun lapsilla parentCodeGroupId-sarakkeen arvo on juurisolmun pääavain. CodeGroup-taulun juurisolmujen code-sarakkeen arvot ovat suurimmaksi osaksi nimetty taulun nimen mukaan. Ostotilauksen juurisolmun arvo on ”Order” ja siihen kuuluu ryh- OrderStatus” se ”OrderType”.

Tuotantotilauksen juurisolmun arvo on ”OrderBatch” siihen kuuluu ry ”OrderBatch-Status”. Ostotilauksen ja tuotantotilauksen tyyppikoodit kuuluvat OrderType-ryhmään, jonka suomennos name-sarakkeessa on ostotilaustyyppi. Ohjelmassa 25 on esitelty SQL-kysely, jolla haetaan ostotilauksen ja tuotantotilauksen tyyppikoodit.

```
SELECT
  c.codeID,
  c.code,
  c.name
FROM Code c
INNER JOIN CodeGroup cg1 ON c.codeGroupID = cg1.codegroupID
INNER JOIN CodeGroup cg2 ON cg1.parentCodeGroupID = cg2.codeGroupID
WHERE cg2.code = ‘Order’ AND cg1.code = ‘OrderType’
```

Ohjelma 25. *Ostotilauksen ja tuotantotilauksen tyyppikoodien hakeminen*

Code-taulun code-sarakkeen tyyppitunnisteet ovat ostotilauksella “PURCHASEORDER” ja tuotantotilauksella ”PRODUCTIONORDER”. Ostotilauksen tilaa kuvaavat koodit haetaan muuttamalla ohjelman 25 kyselyyn WHERE-lauseessa cg1.code-sarakkeen arvoksi ”OrderStatus”.

Tuotantotilauksen tilat haetaan muuttamalla cg2.code-sarakkeen arvoksi ”OrderBatch” ja cg1.code-sarakkeen arvoksi ”OrderBatchStatus”. Tuotantotilauksen tyyppi on siis ostotilaukselle tarkoitetussa ryhmässä ja tuotantotilauksen tilaa kuvaavat koodit ovat eri juurisolmun alla sijaitsevassa ryhmässä.

BatchRow-taululla on viittaus Order-tauluun ja sillä on myös tyyppitunniste Code-taulussa, joka on eroteltu erillisellä ryhmällä CodeGroup-taulussa. Juurisolmun code-sarakkeen arvo codeGroup-taulussa on ”BatchRow” ja sen suomennos name-sarakkeessa on ostotilausrivi. Tämän tyyppitunnisteet kuuluvat ryhmään ”BatchRowType”, jonka suomennos name-sarakkeessa on ostotilausrivityyppi. Ostotilauksiin viittaavilla BatchRow-tauluilla ja tuotantotilauksiin viittaavilla BatchRow-tauluilla on omat code-sarakkeen arvot Code-taulussa. Ohjelmassa 26 esitetään vertailun vuoksi BatchRow-taulun tyyppikoodien hakeminen koodiryhmän suomennoksilla.

```
SELECT
  c.codeID,
  c.code,
  c.name
FROM Code c
INNER JOIN CodeGroup cg1 ON c.codeGroupID = cg1.codegroupID
INNER JOIN CodeGroup cg2 ON cg1.parentCodeGroupID = cg2.codeGroupID
WHERE cg2.name = ‘Ostotilausrivi’ AND cg1.code = ‘Ostotilausrivityyppi’
```

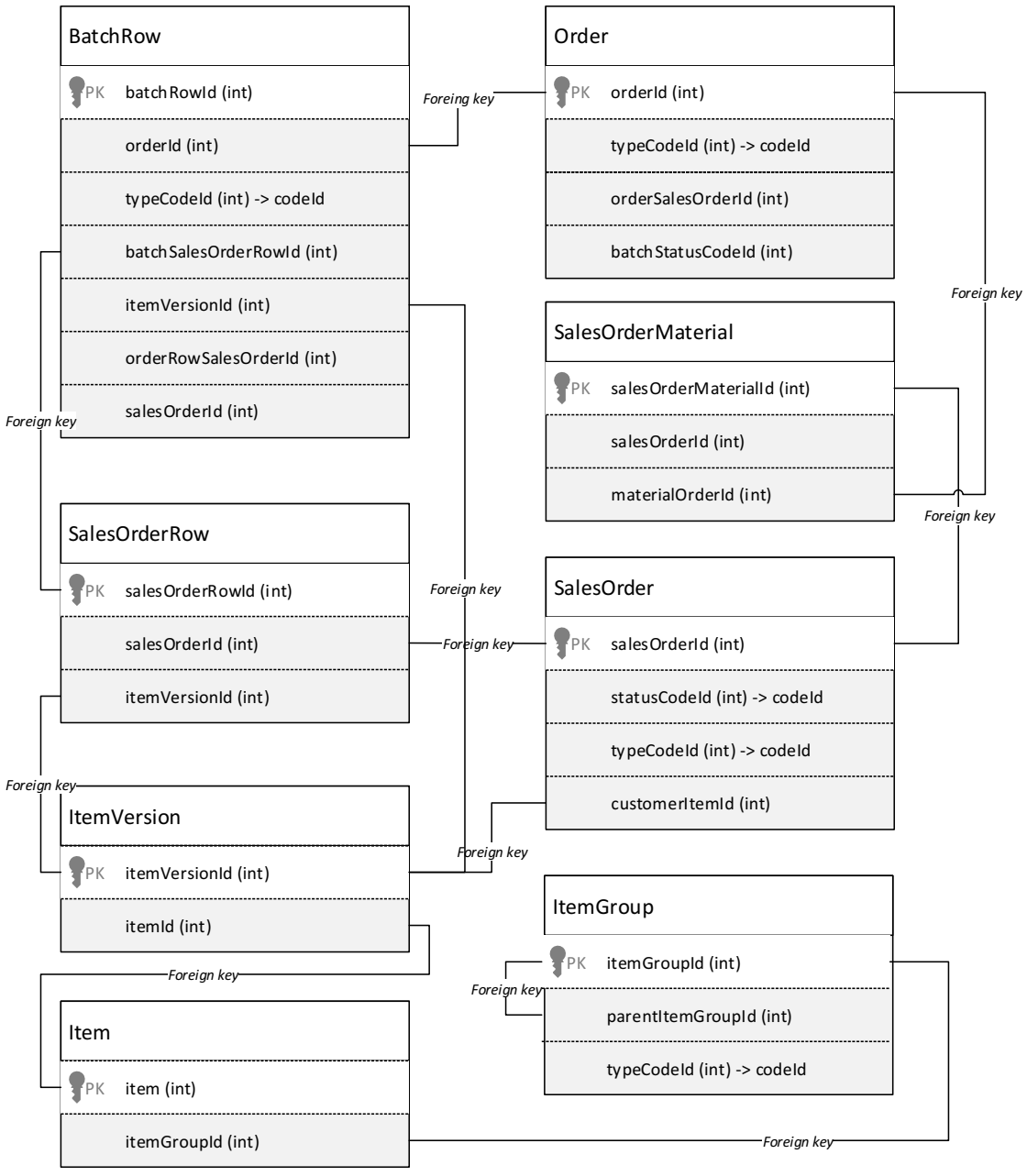
Ohjelma 26. *BatchRow-tyyppikoodien hakeminen suomennoksen kautta*

Ostotilauksilla on ”PURCHASEORDER” ja tuotantotilauksilla on ”PRODUCTIONORDER”. Tyyppitunnisteiden koodit ovat ysin samat kuin Order-taululla, mutta pääavaimien codeId-arvot eroavat.

Tuotantotilaukset ja ostotilaukset tallennetaan samaan tilauksille tarkoitettuun Order-tietokantatauluun. Tuotantotilauksella tarkoitetaan tuotantoer ”Batch”. Ostotilauksella tarkoitetaan materiaalitilausta ”Material Order”. CodeGroup-taulun suomennokset ovat harhaanjohtavia ja koodit, kuten esimerkiksi ”BatchRowType” on myös harhaanjohtava, koska kyseiseen ryhmään kuuluu myös muita käsitteitä, jotka eivät liity ollenkaan ryhmän koodiin tai nimeen.

4.3.2 E1-tietokantataulujen periytyminen

E1-tietokannassa on hyvin paljon STI-periytymistä, jossa tyyppien tunnistamisessa on käytetty Code-taulua. Kuvassa 18 esitellään e1-tietokannassa käytettyä STI-periytymistä eri tyyppisille tilauksille ja nimikkeille, jossa tietokantakaavion yksinkertaistamisen vuoksi Code- ja CodeGroup-taulut on jätetty pois.



Kuva 18. *e1-tietokannan taulujen STI-periytyminen*

Kuvassa 18 viittaus Code-tauluun on esitetty ”->”-nuolimerkillä sarakkeen tietotyypin jälkeen. Kuvasta 18 voidaan nähdä, että Order-taulun batchStatusCodeId-attribuutilta

puuttuu vierasavain viittaus Code-taulun pääavaimeen. Lisäksi BatchRow-taulun salesOrderId-attribuutilta puuttuu vierasavainviittaus SalesOrder-taulun salesOrderId-sarakkeeseen.

Tuotantoerät ja materiaalitilaukset tallennetaan Order-tilaustauluun. Myyntitilaukset ovat näistä erillisessä SalesOrder-myyntitilaustaulussa. Tilaustaulun ohella on myös BatchRow-tilausrivitaulu, joka on codeGroup-taulun rivien suomennoksilta ja koodeilta hyvin harhaanjohtava. Tilausrivitaululla on viittaus tilaustauluun, jolloin niitä voi olla useita yhtä tilausta kohden. Tuotantoerät ovat myyntitilauksen varsinaisia toteutuksia ja tuotantoerän tilausrivillä käytetään ilman vierasavainta viittausta myyntitilaukseen. Tuotantoerän tilausrivillä on myös vierasavainviittaus SalesOrderRow-myyntitilausrivitauluun, jolla on vierasavainviittaus myyntitilaustauluun.

Myyntitilausrivejä on aina yksi myyntitilausta kohden, koska myyntitilaus on todellisuudessa myyntitilausrivi. salesOrderId-sarake kuvaa myyntitilausrivin tunnusta, eikä myyntitilaukselle ole uniikkia tunnusta e1Database-tietokannassa. SalesOrderRow-taulu pitää tallessa osaa SalesOrder-taulun sarakkeista. Käytetään jatkossa selvyyden vuoksi nimitystä myyntitilaus SalesOrder-taulusta.

Materiaalitilauksella on myyntitilauksen materiaalit linkittävän SalesOrderMaterial-taulun kautta viittaus myyntitilaukseen. Materiaalitilausta ja tuotantoerää kohden pitää olla aina vain yksi tilausrivi. Materiaalitilaukselle ja tuotantoerälle tilausrivi pitää tallessa osaa sen attribuuteista. Myyntitilauksella voi olla myös usea materiaalitilaus. Ohjelmassa 27 on esitelty myyntitilauksen materiaalitilauksien hakeminen myyntitilaukselle, jonka salesOrderId on 1.

```
SELECT
  o.orderId
FROM SalesOrder so
INNER JOIN SalesOrderMaterial som ON so.salesOrderId = som.salesOrderId
INNER JOIN Order o ON som.materialOrderId = o.orderId
INNER JOIN Code c ON o.typeCodeId = c.codeId
WHERE so.salesOrderId = 1 AND c.code = 'PURCHASEORDER'
```

Ohjelma 27. *Myyntitilauksen materiaalitilauksien hakeminen*

Tuotantoerät ja materiaalitilaukset erotellaan toisistaan tyyppitunnisteilla ”PRODUCTIONORDER” ja ”PURCHASEORDER”. Tuotantoe ja materiaalitilauksilla on muutama yhteinen attribuutti, kuten tieto pituudesta ja leveydestä. Tietokantaan on kuitenkin tehty tyyppiä kohden uusia attribuutteja eri nimillä, jotka ovat tyypistä riippuen arvossa NULL. Yhteisiä attribuutteja näiden välillä on hyvin vähän, joten nämä ovat kaksi hyvin erilaista käsitettä samassa taulussa.

E1-järjestelmään on myöhemmin tullut uusi ostotilaus eli lavatilaus, joka poikkeaa ominaisuuksiltaan hyvin paljon materiaalitilauksesta. Lavatilaukselle käytetään kuitenkin samaa tyyppitunnistetta kuin materiaalitilaukselle. Lavatilauksella voi olla yksi tai useampi

tilausrivi. Lavatilaukselle on tehty myyntitilaukseen liittämistä varten erilliset attribuutit tilaustauluun ja tilausrivitauluun. Liitoksessa käytetään tilaustaulun orderSalesOrderId-saraketta tai tilausrivitaulun orderRowSalesOrderId-saraketta. Ohjelmassa 28 on esitelty myyntitilaukseen liitetyn lavatilauksen hakeminen, jossa myyntitilauksen salesOrderId on 1.

```
SELECT
  o.orderId
FROM SalesOrder so
INNER JOIN BatchRow br ON so.salesOrderId = br.OrderRowSalesOrderId
INNER JOIN Order o ON br.orderId = o.orderId
INNER JOIN Code c ON o.typeCodeId = c.codeId
WHERE so.salesOrderId = 1 AND c.code = 'PURCHASEORDER'
UNION ALL
SELECT
  o.orderId
FROM SalesOrder so
INNER JOIN Order o ON so.salesOrderId = o.OrderSalesOrderId
INNER JOIN Code c ON o.typeCodeId = c.codeId
INNER JOIN BatchRow br ON o.orderId = br.orderId
WHERE so.salesOrderId = 1 AND c.code = 'PURCHASEORDER'
```

Ohjelma 28. *Myyntitilauksen lavatilauksen hakeminen*

Samaa attribuuttia kuin tuotantoerällä myyntitilaukseen liittämistä varten ei voida käyttää, koska tietokantaproseduureissa tuotantoeriä ei ole eroteltu tyyppitunnisteen avulla. Tuotantoeriä käsiteltäessä tyypin tunnistaminen on tapahtunut tuotantoerärivitaulun salesOrderId-sarakkeen liitoksesta myyntitilaustauluun tai batchSalesOrderRow-sarakkeen liitoksesta SalesOrdeRow-tauluun, joita materiaalitilauksessa ei käytetä. Ohjelmassa 29 on esitelty myyntitilauksen tuotantoerien hakeminen myyntitilaukselta, jonka salesOrderId on 1.

```
SELECT
  o.orderId
FROM SalesOrder so
INNER JOIN BatchRow br ON so.salesOrderId = br.SalesOrderId
INNER JOIN Order o ON br.orderId = o.orderId
WHERE so.salesOrderId = 1
```

Ohjelma 29. *Myyntitilauksen tuotantoerien hakeminen*

Nimikejärjestelmässä nimike kuuluu johonkin nimikeryhmään, jolla voi olla edeltäjänä jokin toinen nimikeryhmä. Koodiryhmien tavoin nimiryhmät muodostavat aliluvussa 3.5 esiteltyjä puurakenteita, jossa juurisolmut ovat tyypitetty kooditaulun arvoilla. Nimikkeet ovat tilausten tavoin tallennettu kahteen tauluun. Nimikkeellä on Item-taulussa kolme saraketta nimikkeen nimeämistä varten. Nimikkeen nimi on 255 merkkiä pitkässä Name-sarakkeessa ja koodi on 50 merkkiä pitkässä Code-sarakkeessa. Nimikkeellä on myös 50 merkkiä pitkä shortCode-sarake. Nimikkeen varsinaiset arvot ovat ItemVersion-taulussa, joita on aina yksi rivi Item-taulun riviä kohden.

5. KEHITTÄMINEN

Ohjelman ylläpidettävyydellä tarkoitetaan kuinka helposti ja riskittömästi järjestelmää voidaan muokata. Ylläpidettävyys on ISO 25010 -standardin mukaan yksi tietojärjestelmän laatuominaisuuksista (Visser 2016).

Refaktoroidessa parannetaan järjestelmän sisäistä rakennetta eli olemassa olevaa ohjelmakoodia, niin että ulkoinen toiminta pysyy ennallaan. Refaktoroinnissa kiinnitetään huomiota rakenteen helpompaan ymmärrettävyyteen, jonka yhteydessä usein löydetään olemassa olevia virheitä järjestelmästä. Ohjelman sisäisen rakenteen korjaamiseen kuuluu huonon ohjelmakoodin korjaaminen, kuten toistuvien rakenteiden muuttaminen yhdeksi komponentiksi. Metodeista tehdään lyhyempiä ja pilkotaan osiin, koska usein liian pitkät metodit vaikeuttavat ohjelman ymmärtämistä. Metodien kutsuja tehdään helpommaksi poistamalla tai lisäämällä parametreja. Parametrit metodien vaihtoehtoista toiminnallisuutta varten poistetaan, jolloin jokaista vaihtoehtoa kohden tehdään uusi metodi. Metodit nimetään uudelleen, niin että sen nimi ilmaisee sen toiminallisuuden. Rajapintoja varten tehdään yksikkötestejä ja luokkien vastuualueet korjataan niiden nimen mukaisiksi (Fowler 2002b).

Tässä luvussa arvioidaan e1-järjestelmän ylläpidettävyyttä tunnettujen periaatteiden pohjalta. Lisäksi arvioidaan työnmäärä, jolla järjestelmän ylläpidettävyyttä voidaan kehittää.

5.1 Merkitykselliset nimet

Muuttujien, metodien ja luokkien nimistä ylläpidettävissä ohjelmissa nähdään niiden merkitys ja toiminta. Luokkien nimet tulisi olla substantiiveja eikä sisältää verbiä. Metodien nimissä tulisi olla niiden toimintaa kuvaava verbi. Nimet eivät saa olla harhaan johtavia. Nimien tulisi myös olla helposti haettavia ohjelmakoodista (Martin 2009).

e1Client-asiakasohjelman attribuuttien nimet eivät suurimmaksi osaksi kuvaa ollenkaan niiden toiminnallisuutta. Attribuutit ovat pääasiassa nimetty attribuuttien tyypin perusteella ja lisäämällä nimessä tyypin perään numero erottamaan samat tyypit toisistaan. Esimerkiksi FrmProfiles-näkymässä on kolme DataSet-tietoaineistoa, joiden nimet ovat _dsRows, _dsRows2 ja _dsRows3.

Nämä voidaan nimetä uudelleen Visual Studion refaktorointityökalua apuna käyttäen esimerkiksi muotoon _dataSetProfiles, _dataSetPriviledges ja _dataSetAlarms. Nimeämällä esimerkiksi button1 uudelleen muotoon buttonCancel ja button1_OnClick-tapahtumankäsittelijä muotoon buttonCancel_OnClick voidaan tehdä käyttöliittymäsuunnittelusta huomattavasti helpompaa, koska e1Client-asiakasohjelman käyttöliittymässä on käytetty teknologioita, jotka ovat nykyään vanhentuneita eikä uudempien kehitysympäristöjen

graafisilla käyttöliittymäeditoreilla ole tukea näille. Uudelleen nimeäminen on turvallinen toiminto refaktorointityökalun avulla, koska mitään toiminnallisuutta ei muuteta ja kaikki viittaukset voidaan uudelleen nimetä samaan aikaan. Attribuuttien uudelleen nimeäminen helpottaa ja nopeuttaa huomattavasti ohjelmakoodin lukemista. Refaktorointi olisi melko työlästä, koska e1Client-asiakasohjelmassa on 140 kappaletta näkymiä.

E1-järjestelmässä asiakasohjelmat kommunikoivat pääasiassa metodien ExecuteBusinessLogicProcedure, GetDataSet, SaveDataSet, InsertRow, DeleteRow, UpdateRow, GetScalar ja ExecuteReader kautta. Tämä on hyvin ongelmallista, koska metodien vaihtoehtoisen toiminnallisuuden vuoksi tiettyyn toiminnallisuuteen ei voida etsiä suoraan viittauksia ohjelmakoodista, jolloin toimintoja täytyy etsiä hakusanoilla. Esimerkiksi pakkaamiseen liittyvien tietokantaproseduurien kutsuja joudutaan etsimään hakutermillä ”ExecuteBusinessLogicProcedure(”Packing””. Ylläpidettävyyden kannalta applicationName- ja applicationMethodName-parametrien poistaminen, ja niiden korvaaminen erillisillä metodeilla e1Remoting-palvelussa tekisi tietokantaproseduurien hakemisesta helpompaa ohjelmakoodissa, koska tällöin tietyn tietokantaproseduurin suorittavaan metodiin olisi suora viittaus. Samoin className-parametrien poistaminen, ja niiden korvaaminen erillisillä metodeilla tekisi Bl-luokkien ylläpidettävyydestä helpompaa. Refaktorointi olisi työlästä, koska e1Database-tietokannassa on 670 kappaletta tietokantaproseduureja ja numeerisella className-tyypillä on 300 kappaletta arvoja. Tämä myös vaatii ohjelmakoodin muuttamista hyvin paljon, joka on riskialtista. Muutoksen päivittäminen olisi myös työlästä kaikkien yksiköiden e1Remoting-palveluihin samaan aikaan.

E1-järjestelmässä on myös käytetty harhaanjohtavaa nimeämistä toiminnallisuuksien kuvaamisessa. Esimerkiksi tietokantaproseduuri ”spBlSalesOrderSave” tallentaa myyntitilaukseen tehdyt muutokset sekä poistaa tuotantoeriä. CodeGroup-taulun ryhmien koodit ja nimet ovat hyvin harhaanjohtavia. Vääristä ryhmistä koodien pois siirtäminen ja uusien kuvaavien ryhmien luominen olisi hyvin työlästä ja riskialtista, koska tietokantaproseduurien ohjelmakoodissa oletetaan koodien kuuluvan alkuperäisiin ryhmiin. Tämä vaatisi muutoksen jäljittämisen kaikkiin tietokantaproseduureihin, joissa on yhteensä 270 000 riviä ohjelmakoodia.

e1Client-asiakasohjelmassa nimeämiskäytännöt eivät myöskään ole aina helposti haettavia. Esimerkiksi järjestelmästä on hyvin vaikeata hakea kaikki toiminnot, joissa käsitellään pakkaamisen viimeistelyä. Joissakin näkymissä tämä on oikein kirjoitettu muotoon ”PackingFinished”, mutta suurimmaksi osaksi esit eksi on ytetty muotoa ”PackingFinnished”, jolloin on hankalaa löytää kaikki pakkaamisen viimeistelyä käsittelevät koodiyksiköt. Nimien muuttaminen haettaviksi on ongelmallista, koska virheellinen nimeäminen on tehty tietokantataulujen sarakkeisiin ja tietokantaproseduureihin asti.

5.2 Virheidenkäsittely

Ylläpidettävissä järjestelmissä virheiden käsittely tulisi tehdä poikkeuksilla paluuarvoina annettujen koodien sijaan. Poikkeuksien tulisi antaa mukanaan tarpeeksi tietoa virheen lähteestä ja sijainnista. NULL-arvojen välittämistä tulisi myös välttää, koska tämä johtaa tilanteeseen, jossa suurin osa toteutuksesta koostuu NULL-arvojen tarkastelusta. Yksikin puuttuva NULL-arvon tarkastelu voi johtaa ohjelman päättymisen virheeseen (Martin 2009).

E1-järjestelmän virheilmoitukset eivät kuvaa selkeästi virheen syytä ja mistä koodiyksiköstä se on lähtöisin. Järjestelmässä tehdään jossain määrin poikkeuksien käsittelyä ja lähes kaikki lokitauluun kirjoitetut poikkeukset lähtevät ExecuteBusinessLogicProcedure-metodista. Ylläpidettävyyden kannalta applicationName- ja applicationMethodName-parametrien esittäminen virheilmoituksissa tekisi poikkeuksien jäljittämisestä helpompaa. Esimerkiksi virheen syy voi olla liian suuren merkkijonon sijoittaminen jonkin tietokantaproseduurin parametersXml-parametriin tai tietokantaproseduurin paikalliseen muuttujaan.

Tietokantaproseduurien sisäisessä ohjelmakoodissa käsitellään hyvin vähän poikkeuksia. Joillekin e1-järjestelmän väärille toiminnoille määritellään virhekoodeja paluuarvona annettavassa TableError-taulussa, jota ei käsitellä poikkeuksilla. Virheelliset viittaukset tietokantaproseduurin dataSetXml-parametriin eivät aiheuta mitään virheilmoitusta, joten tietokantaproseduurien ylläpitäminen on hyvin riskialtista. dataSetXml-parametrien poistaminen helpottaisi huomattavasti järjestelmän ylläpitoa, mutta tämä vaatisi lähes kaikkien tietokantaproseduurien uudelleen kirjoittamista.

Tietoaineistojen käyttäminen asiakasohjelmissa aiheuttaa hyvin paljon NULL-arvojen käsittelyä joissakin näkymissä. Esimerkiksi tilanteet joissa tiettyä tietoaineistoa ei alusteta ennen kuin näkymän tietyn attribuutin arvo on syötetty. Näkymän avaamisessa kuitenkin kutsutaan refresh-metodia, joka hakee esimerkiksi tietokannan parametritaulusta tiedon painikkeen näkyvyydestä. Refresh-metodi myös alustaa näkymän kentät tietoaineiston perusteella, joka on näkymän avaamisen yhteydessä NULL, jolloin ohjelmakoodissa on hyvin paljon NULL-arvojen tarkastelua.

5.3 Pienet ja yksinkertaiset metodit

Ylläpidettävissä ohjelmissa metodien tulisi olla mahdollisimman lyhyitä. Lyhyet metodit ovat helpompi ymmärtää, testata ja uudelleen käyttää. Metodeilla tulisi olla myös selkeät vastuualueet. Metodeja voidaan tehdä pienemmäksi jakamalla niiden vastuualueita erillisiin metodeihin. Metodien pituudella tarkoitetaan koodirivien määrää. Metodin pituus ei saisi olla yli 100 koodiriviä ja hyvän metodin pituus on alle 20 koodiriviä (Martin 2009).

E1-järjestelmän tietokantaproseduurit voivat olla jopa 8000 koodiriviä pitkiä, mikä hidastaa hyvin paljon niiden toiminnallisuuden ymmärtämistä. Tietokantaproseduureilla ei ole myöskään selkeitä vastuualueita. Esimerkiksi raporttien tulostamista varten on erilliset tietokantaproseduurit, jotka hakevat raportin datan. Joihinkin raporttien tietokantaproseduureihin on tehty käsitteen tilan muuttava toiminnallisuus samalla, jolloin raporttia ei voida esikatsella ilman käsitteen tilan muuttamista.

e1Remoting-palvelun rajapinnassa on hyvin vähän metodeja. Vastuualueita metodeilla voi olla satoja ja parametrit voivat olla mitä tahansa, jolloin metodeja ei voida kutsua tuntematta niiden sisäistä toteutusta täysin. Metodit eivät myöskään ole yleiskäyttöisiä.

5.4 Pienet ja yksinkertaiset luokat

Ylläpidettävissä ohjelmissa luokkien tulisi olla mahdollisimman lyhyitä. Luokkien pituudella tarkoitetaan luokan vastuualueiden määrää (Martin 2009).

Windows Forms -näkymissä on oletuksena osittaisluokkarakenne, jossa käyttöliittymän elementit ovat erillisessä Designer-tiedostossa, johon Visual Studion käyttöliittymäeditori automaattisesti kirjoittaa elementteihin liittyvän ohjelmakoodin. e1Client-asiakasohjelmassa osittaisluokkarakenne on purettu ja näkymän kaikki koodi on samassa luokassa. Näkymien luokat voivat sisältää 12000 riviä ohjelmakoodia, joista suurin osa on näkymän elementtien esittämistä varten. Palauttamalla näkymät osittaisluokkarakenteeseen voidaan näkymien ohjelmakoodista tehdä helpommin ymmärrettävää, koska näkymän toiminnallisuus olisi erillään käyttöliittymän elementeistä.

5.5 Toiminnallisuus kirjoitetaan vain kerran

Ylläpidettävyyteen kuuluu periaate, jossa toiminnallisuus kirjoitetaan vain kerran. Tämä tehdään kirjoittamalla uudelleen käytettäviä metodeja. Koodin kopioiminen hidastaa hyvin paljon ylläpitoa, koska muutokset täytyy tehdä kaikkialle, jossa kopioitua koodia on käytetty (Visser 2016).

Tietokantaproseduureissa toiminnallisuutta on laajennettu tekemällä eri nimiselle käsitteelle uuden ehtolauseen, jolloin edellisen ehtolauseen sisältä ohjelmakoodi on kopioitu ja uuden ehtolauseen sisälle on tehty hyvin pieniä muutoksia. Tietokantaproseduureissa ehtolauseiden sisältö voi olla 500 riviä pitkä, mikä on kopioitu useita kertoja peräkkäin. Tämä vaikeuttaa ylläpitoa, koska muutokset voivat koskea kaikkia ehtolauseita. Ehtolauseiden refaktorointi helpottaa ylläpitoa, mutta testaaminen on hidasta, koska e1-järjestelmässä ei ole ainuttakaan yksikkötestiä.

E1-järjestelmässä ei ole uudelleen käytettävää toiminnallisuutta, koska viestintä perustuu näkymien tietoaineistoihin, jolla käsitteiden oliomalli on korvattu. Esimerkiksi tuotan-

toerän poistamiselle on erilliset toteutukset näkymiä varten, jotka käyttävät eri tietoaineistoja. Tietoaineistojen välillä samojen taulujen ja sarakkeiden nimet ovat harvoin kirjoitettu yhtenevästi. Henkilön etunimen ja sukunimen yhdistelmää varten on tehty erillinen metodi, eikä samaa metodia kuin henkilön tietojen hakemiseen FrmPerson-näkymässä käytetä, koska tämä hakee tietoaineiston, joka sisältää henkilön tietojen lisäksi muita henkilöön liittyviä käsitteitä. Tietoaineistojen poistaminen ja oliomalliin siirtyminen vaatisi koko järjestelmän uudelleen rakentamista.

5.6 Koodin kommentointi

Yläpidettävyyden kannalta ohjelmakoodin kommentoinnin tulee olla informatiivista ja perustella ohjelmakoodissa käytettyjä ratkaisuja. Koodin kommentoinnilla ei kuitenkaan tule korvata merkityksellisiä nimeämiskäytäntöjä. Vanhan ohjelmakoodin pois kommentointi on ylläpitoa vaikeuttavaa, koska ohjelmakoodista ei voida tietää miksi koodiyksikkö on poistettu käytöstä (Martin 2009).

E1-järjestelmässä on hyvin paljon pois kommentoitua ohjelmakoodia. Tietokantaproseduurien ohjelmakoodin välistä voi olla 1000 koodiriviä kommentoitu pois ilman mitään selitystä. Yksittäisiä attribuutteja on myös kommentoitu pois ilman selitystä. Tämä tekee ohjelmakoodin rakenteesta sekavaa ja vaikeuttaa ylläpitoa. Informatiivisia kommentteja e1-järjestelmän ohjelmakoodissa on hyvin vähän. Kommentoidun koodin poistaminen parantaa ylläpidettävyyttä, mutta muutoksen tekeminen kaikkiin tietokantaproseduureihin on työlästä, johtuen tietokantaproseduurien määrästä ja eri versioista yksiköiden välillä.

5.7 Yksikkötestit

Ylläpidettävissä järjestelmissä on automaattisia yksikkötestejä, jotka voidaan suorittaa toistuvasti, kun muutoksia tehdään. Yksikkötesteillä voidaan nopeasti tarkistaa esimerkiksi, että metodin lopputulos on validi. Yksikkötestien käyttäminen ohjaa kirjoittamaan testattavaa ohjelmakoodia, joka samalla on helposti ymmärrettävää ja ylläpidettävää. Lisäksi ohjelman toiminnasta saadaan ennustettava (Visser 2016).

E1-järjestelmässä ei ole ainuttakaan yksikkötestiä eikä siihen ole mahdollista tehdä yksikkötestejä, koska metodit ovat hyvin pitkiä, eikä niillä ole selkeitä vastuualueita. e1Client-asiakasohjelman metodit sisältävät tietokantaproseduurikutsuja, jolloin metodien toiminnot ovat riippuvaisia myös tietokantaproseduurien validista toiminnasta. Tietokantaproseduureille ei voida tehdä yksikkötestejä, koska parametrit sisältävät näkymän tietoaineiston XML-muodossa, joka voi olla kooltaan hyvin suuri.

5.8 Kerroksien riippuvuuksien minimointi

Arkkitehtuurissa kerroksien välisien riippuvuuksien tulee olla mahdollisimman vähäiset. Moduulien tulee olla mahdollisimman pieniä ja käyttää uudelleen käytettävää koodia muista moduuleista. Itsenäiset komponentit ovat helpommin ylläpidettäviä, koska ylläpidossa tehtävät toimenpiteet voidaan eristää vain tiettyyn osaan ohjelmasta (Visser 2016).

E1-järjestelmässä tietoaineistojen käyttämisen vuoksi kaikki kerrokset ovat riippuvaisia toisistaan. Tämä vaikeuttaa erittäin paljon järjestelmän ylläpitoa. Väärän suuntaisten riippuvuuksien poistaminen vaatii DataSet-tyypin parametrien poistamisen, joka vaatii koko järjestelmän rakentamista uudelleen.

6. JATKOKEHITYS

Ylläpidettävyys ei ole jälkeen päin tehtävä ominaisuus järjestelmässä. Ylläpidettävyyteen vaikuttaa heti suunnitteluvaiheessa tehdyt ratkaisut (Visser 2016). Huono ohjelmakoodi jättää järjestelmään bugeja ja uusien ominaisuuksien toteuttaminen hidastuu, koska järjestelmässä kaikkien huonosti tehtyjen ohjelmayksiköiden toiminta täytyy opetella ja selvittää niiden vaikutukset järjestelmään. Robert C. Martinin mukaan huono ohjelmakoodi ajan kuluessa laskee tuottavuutta tilanteeseen, jossa tuottavuus lähestyy asymptoottisesti nollaa (Martin 2009).

E1-järjestelmän ylläpidettävyyttä voidaan parantaa turvallisesti refaktoroimalla attribuuttien ja metodien nimeäminen merkityksellisiksi sekä kehittämällä poikkeuksia informatiivisemmiksi. Ylläpitoa parantaisi myös kaikkien yksiköiden tietokantaproseduurien versioiden yhdistäminen yhdeksi versioksi. Kommentoidun koodin poistaminen kaikkien yksiköiden tietokantaproseduureista auttaisi myös järjestelmän ylläpitoa. Näkymien osittaisluokkarakenteen palauttaminen parantaisi myös ylläpidettävyyttä. e1-järjestelmän arkkitehtuurin toteutus on kuitenkin helposti ylläpidettävän järjestelmän vastainen, joten järjestelmään ei ole mielestäni suositeltavaa tehdä suurempaa refaktoroitia.

Nykyisessä e1-järjestelmässä yksiköiden välillä on erilliset tietokannat, jolloin e1-järjestelmä ei täytä ERP-järjestelmän keskitetyn tietokannan vaatimusta. Tämän vaatimuksen täyttämiseksi yksiköiden paikallisten e1-alijärjestelmien e1-tietokannat täytyisi yhdistää. Nykyisen e1-järjestelmän kehittäminen ERP-järjestelmän mukaiseksi ei ole kuitenkaan suositeltavaa, koska tietokanta on rakennettu antisuunnittelumalleja käyttäen, joten tietokantaa ei ole suositeltavaa käyttää uudelleen. Tietokannan vaihtaminen nykyisessä e1-järjestelmässä ei ole myöskään mahdollista, koska e1-järjestelmän kaikki kerrokset ovat riippuvaisia toisistaan. e1-järjestelmässä parametrit kerroksien välissä voivat olla hyvin suuria, joten nykyinen arkkitehtuurinen ratkaisu aiheuttaa myös suorituskykyyn liittyviä ongelmia.

7. YHTEENVETO

Työssä käsiteltiin Efigen Oy:n kehittämää toiminnanohjausjärjestelmää, joka on tavoitellut yleiskäyttöisen ERP-järjestelmän ominaisuuksia, joita ovat modulaarisuus ja moduulien vaiheittainen käyttöönotto yrityksen tarpeiden mukaan. E1-järjestelmän tavoitteena on myös ollut helppo käyttöönotto. Työssä ei tarkasteltu e1-järjestelmän käyttöönoton helppoutta, mutta järjestelmästä paremman kokonaiskuvan saamiseksi esiteltiin käyttöönoton vaiheet. E1-järjestelmän kehityshistorian kautta esiteltiin järjestelmän elinkaaren pituutta mahdollisten arkkitehtuuristen muutosten vuoksi.

Työssä esiteltiin rakenteellinen suunnitelma e1-järjestelmästä, jonka yhteydessä esiteltiin nykyisen e1-järjestelmän laitteisto ja komponenttien hajautus. Komponenttien hajautuksen yhteydessä havaittiin, että nykyinen e1-järjestelmä ei täytä ERP-järjestelmän vaatimuksia, koska yksiköiden välillä on erilliset tietokannat.

Ensimmäisenä tavoitteena oli e1-järjestelmän arkkitehtuurin rekonstruointi, jonka yhteydessä tarkasteltiin kerrosarkkitehtuurin periaatteiden noudattamista. E1-järjestelmässä kerrokset ovat tiiviisti riippuvaisia toisistaan ja kerroksien välillä on myös väärän suuntaisia riippuvuuksia. E1-järjestelmän rajapinnassa on hyvin vähän metodeja ja yksittäisillä metodeilla hyvin paljon vastuualueita. Metodien parametreissa ei ole myöskään rajoitettu annettujen parametrien määrää eikä tyyppiä, jolloin metodeja ei voida käyttää tuntematta täysin niiden toiminnallisuutta. E1-järjestelmässä on myös asiakas-palvelin-arkkitehtuurin ominaisuuksia, koska käyttöliittymästä tehdään suoraan kyselyitä tietokantaan.

Tietokanta rekonstruoitiin yksityiskohtaisemmin, koska tietokanta on keskeinen osa ERP-järjestelmissä. E1-järjestelmän tietokanta on suunniteltu antisuunnittelumalleja käyttäen. Tietokannassa on käytetty hyvin paljon STI-periytymistä ja OSFA-antisuunnittelumallin mukaisesti lähes kaikille entiteeteille on määritelty surrogaattiavain. Periytetyt tyypit erotellaan toisistaan SLT-antisuunnittelumallin mukaisella kooditaululla, jossa samat koodit erotellaan toisistaan ryhmillä, jotka käyttävät ADOP-antisuunnittelumallin mukaista puurakennetta. Nimikejärjestelmä käyttää myös ADOP-antisuunnittelumallin mukaista puurakennetta, jolloin yksikertaisten toiminnallisuuksien suorittamisesta tulee hyvin monimutkaisia. Tietokannassa viittauksia tehdään hyvin paljon ilman vierasavainta ja tauluissa voi olla useita sarakkeita, joita käytetään viitatessa samaan pääavaimeen.

Toinen tavoite oli e1-järjestelmän kehittäminen, joten rekonstruoinnin pohjalta tarkasteltiin e1-järjestelmän ylläpidettävyyttä teoriaan perustuen, minkä yhteydessä arvioitiin refaktorointiin vaadittavaa työn määrää e1-järjestelmän ylläpidettävyyden kehittämiseksi. E1-järjestelmä on suunnittelun aloitusvaiheessa tehtyjen ratkaisujen pohjalta helposti ylläpidettävän järjestelmän vastainen. Jälkeenpäin e1-järjestelmästä ei voida tehdä helposti

ylläpidettävää, mutta järjestelmää on mahdollista kehittää, mikä parantaisi ylläpidettävyyttä.

Jatkokehityksen kannalta e1-järjestelmässä ei ole yhtään uudelleen käytettävää komponenttia, koska kerroksien eristyneisyyden periaatetta ei ole noudatettu ollenkaan ja kerrokset ovat tiiviisti riippuvaisia toisistaan. Tietokantaa ei ole myöskään suositeltavaa käyttää uudelleen, koska tietokannan rakenne on tehty antisuunnittelumallien pohjalta. Työssä ei tarkasteltu e1-järjestelmän integraatioita ulkopuolisiin järjestelmiin.

LÄHTEET

Efigen Oy. Kauppalehti. Saatavissa (viitattu 21.12.2017): https://www.kauppalehti.fi/yritykset/yritys/efigen+oy/14628907.

ERP-j rjestelm n hankinta. Logistiikan maailma. Saatavissa (viitattu 3.1.2018): http://www.logistiikanmaailma.fi/wp-content/uploads/2017/02/ERP-jarjestelman_hankinta.pdf.

Fowler, M. (2002a). Patterns of Enterprise Application Architecture. Addison Wesley. p. 1-30.

Fowler, M. (2002b). Refactoring: Improving the Design of Existing Code. Addison Wesley. p. 9, 45-80.

Iskanius, P. & Juuso, J. (2009). Arviointikriteerit toiminnanohjausjärjestelmän valintaan. Oulun yliopisto, Raahen toimintayksikkö, Raahe. s. 33. Saatavissa: https://www.researchgate.net/publication/253341758_Arviointikriteerit_toiminnanohjausjarjestelman_valintaan_TOMI_raportti_5.

Karvonen, T. (2004). Kotimaista väriä toiminnanohjaukseen. Digitoday. Saatavissa (viitattu 21.12.2017): https://www.is.fi/digitoday/art-2000001425428.html.

Karwin, B. (2014). SQL Antipatterns. The Pragmatic Programmers, p. 3, 23-52, 70.

Koskimies, K. & Mikkonen, T. (2005). Ohjelmistoarkkitehtuurit. Talentum. s. 126-130.

Martin, R.C. (2009). Clean Code A Handbook of Agile Software Craftsmanship. Pearson Education, p. 4, 17-71, 103-112, 135-150.

Microsoft .NET Remoting: A Technical Overview. Microsoft Corporation. Saatavissa (viitattu 15.1.2018): https://msdn.microsoft.com/en-us/library/ms973857.aspx.

Peterson, D. (2006). Lookup Table Madness. SQLServerCentral.com. Saatavissa (viitattu 28.12.2017): http://www.sqlservercentral.com/articles/Advanced/lookuptablemadness/1464.

Richards, M. (2015). Software Architecture Patterns. O'Reilly Media. Saatavissa (viitattu 25.12.2017): https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html.

Syverson, B. & Murach, J. (2016). Murach's SQL Server 2016 for Developers. Mike Murach & Associates. p. 4-21, 450-486.

Toiminnanohjausjärjestelmä. Logistiikan maailma. Saatavissa (viitattu 27.12.2017): http://www.logistiikanmaailma.fi/logistiikka/ohjausjarjestelmat/toiminnanohjausjarjestelma.

Troelsen, A. (2012). Pro C# 5.0 and the .NET 4.5 Framework. 6th ed. Apress, p. 3-22, 859-876, 985-990, 1030-1036.

What is ERP. Netsuite. Saatavissa (viitattu 22.12.2017): http://www.netsuite.com/portal/resource/articles/erp/what-is-erp.shtml.

Windows Forms. Microsoft Corporation. Saatavissa (viitattu 29.12.2017): https://docs.microsoft.com/en-us/dotnet/framework/winforms.

Visser, J. (2016). Building Maintainable Software, C# Edition. O'Reilly. p. 2, 43-54, 81-90, 111-124.

XML Tutorial. W3Schools. Saatavissa (viitattu 29.12.2017): https://www.w3schools.com/xml/default.asp.