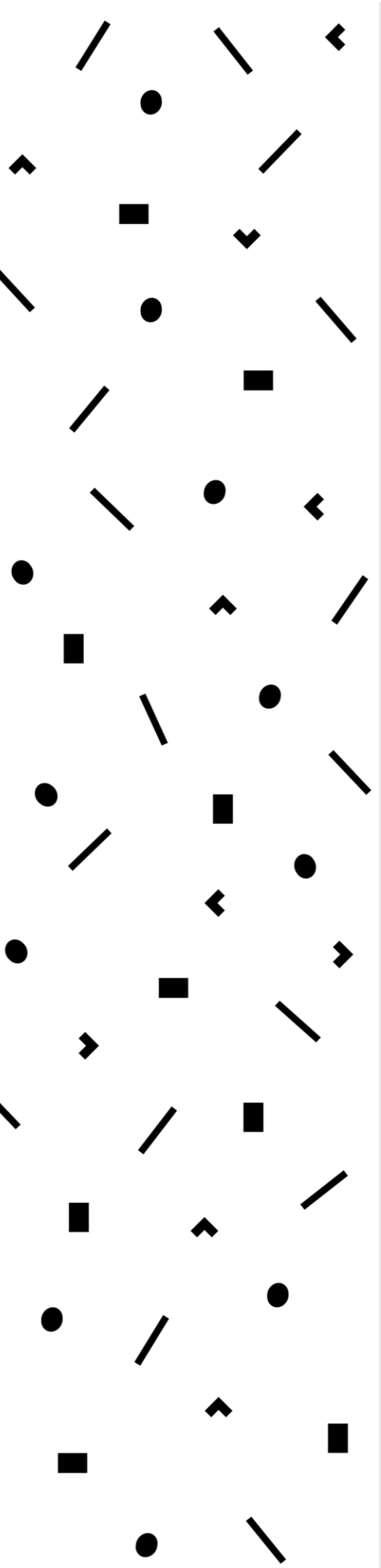


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Tuotannonohjausjärjestelmä

.NET-sovellus osana tuotantolinjaa

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2018
Niko Torvinen

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

TORVINEN, NIKO:

Tuotannonohjausjärjestelmä
.NET-sovellus osana tuotantolinjaa

Ohjelmistotekniikan opinnäytetyö, 42 sivua

Kevät 2018

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli toteuttaa tuotantoa tehostava Windows-sovellus käytettäväksi Nipema Oy:n rakentamalla pintakäsittelylinjastoilla, joilla on ohjelmoitava logiikkaohjainyksikkö. Työn aikana luotiin koekäyttöön tarkoitettu sovellus, jonka ensimmäinen versio asennettiin valmiiseen linjastoon jo toiminnassa olevan ohjausyksikön kautta. Koekäytön perusteella on määrä jatkaa järjestelmän kehitystä.

Nipema Oy:n kehittelemän järjestelmän tarkoituksena on tehostaa linjan tuotantoa. Se vähentää virhetapahtumien lukumäärää ja auttaa työntekijöitä näyttämällä ohjeita. Ohjelma havaitsee ongelmia tuotannossa tarkkailemalla työn seisauksia ja hylättyjen tuotteiden lukumäärää. Järjestelmä tallentaa tunnuslukuja, joiden arvioidaan vaikuttavan työn tehokkuuteen sekä tuotteen laatuun.

Järjestelmään luotiin Windows-työpöytäsovellus, Windows-palvelu sekä SQL Server -tietokanta. Työssä tutkittiin Microsoftin kehittämää Windows Presentation Foundation -käyttöliittymäjärjestelmää, jota käytetään työpöytäsovelluksen käyttöliittymän luomiseen. Erityisesti työssä tutkittiin käyttöliittymäjärjestelmään liittyvän Model-View-Viewmodel -ohjelmistoarkkitehtuurin sekä niin ikään Microsoftin kehittämän, tietokantamallintamiseen käytettävän Entity Framework -liitännäisen ominaisuuksia.

Työssä arvioidaan myös valittujen teknologioiden soveltuvuutta järjestelmän toteutukseen. Valitut teknologiat havaittiin testauksen kautta sopiviksi järjestelmän tarpeisiin. Windows-ympäristöön kehitetyn sovelluksen vaatimukset täyttyivät hyvin. Käyttöliittymäsovelluksen suhteen ongelmaa syntyi vain, kun suurta datamäärää pyrittiin näyttämään yhtäaikaisesti.

Avainsanat: Windows Presentation Foundation, Business Intelligence, tuotantolinjat

Lahti University of Applied Sciences
Degree Programme in Information Technology

TORVINEN, NIKO:

Production control system
.NET application in manufacturing environment

Bachelor's Thesis in software engineering, 42 pages

Spring 2018

ABSTRACT

The objective of this thesis was to design and implement a Windows-native data collection and work instruction system in a surface treatment line that has a programmable logic controller (PLC). During this work a beta application was created for testing purposes and installed in an operating facility. The development of the system is to be continued after the completion of the thesis. The work was commissioned by Nipema Oy, a company which builds various production lines.

The aim of this system is to improve the performance of the production line, to help detect problems and to reduce malfunctions. It improves the effectiveness of work done in the facility by instructing its personnel and tracking interruptions in manufacturing. The application is connected to the manufacturing equipment via a pre-existing Programmable Logic Controller (PLC), which is also provided by Nipema.

A Windows Presentation Foundation (WPF) application was created for the user interface part of the system, while the data is collected from a PLC server using a Windows Service application. Although any user interface framework would work, WPF was chosen because of its wide range of available tools. A secondary goal of this thesis was to study the user interface framework WPF.

The thesis also aims to assess the suitability of the selected software solutions for this project. The selected solutions were found to be well-suited to match the software project's requirements. In this application targeted for the Windows platform, the only major concern was that a large data set was a heavy load on the user interface's memory.

Key words: production line, data collection, Windows Service, Windows Presentation Foundation

SISÄLLYS

1 JOHDANTO

Ihmiskunta on kautta historian kehittänyt yhä tehokkaampia tapoja tehdä työtä. Teollisen vallankumouksen jälkeisenä aikana kehitys on keskittynyt vahvasti automaatiotekniikkaan perustuvaan työn tehostamiseen, jota modernissa liiketoimintaympäristössä edesautetaan hyödyntäen tietotekniikan suomia mahdollisuuksia. Digitalisaatioksikin kutsuttu kehitys kohti tietoyhteiskuntaa painostaa yrityksiä tutkimaan uusia toimintamalleja. (Valtioneuvosto 2014, sivu 4.)

Suomessa ja muissa Pohjoismaissa suurten tuotantolaitosten osalta on osallistuminen moderniin tietotekniseen kehitykseen edelleen vähäistä. Elinkeinoelämän tutkimuslaitoksen (ETLA) vuonna 2014 julkaiseman Digibarometri-tutkimuksen mukaan Suomessa toimivilla yrityksillä on maailman parhaat edellytykset käyttää hyväkseen digitalisaation tuomia etuja, mutta silti Suomi sijoitettiin digitalisaation hyödyntämisen mittauksessa 22 maan joukossa vasta sijalle 7 (Elinkeinoelämän tutkimuslaitos 2014). Informaatiotekniikan sulautuminen osaksi tuotantoteollisuutta on suurelta osin vielä alkuvaiheillaan. Vaikka tuotantolinjoja on runsaasti eri puolella Suomea, eivät ne useinkaan kerää tuotannollisia tunnuslukuja analysoitavaksi.

Opinnäytetyön toimeksiantaja, tuotantolinjoja kokoava ja maahantuova Nipema Oy, on suunnitellut tiedonkeruujärjestelmän, jonka tarkoituksena on kerätä tietoa linjan toiminnasta, helpottaa työtä linjalla, ja täten parantaa tuotannon tehokkuutta. Savitaipaleella toimivalla Nipemalla on kokemusta automaatiotekniikasta jo kymmenien vuosien ajalta, ja osa yrityksen pitkäaikaisista yhteistyökumppaneista on ilmaissut halukkuutensa osallistua tiedonkeruujärjestelmän kehittämiseen. Nipeman liikevaihto vuonna 2016 oli noin 1 200 000 euroa (Kauppalehti 2017). Liikekumppani Tume Agri Oy salli kehityksen aikaisen asennuksen tuotantolaitokselleen Hämeenlinnan Turengissa, jonne ensimmäinen julkinen ohjelmaversio asennettiin.

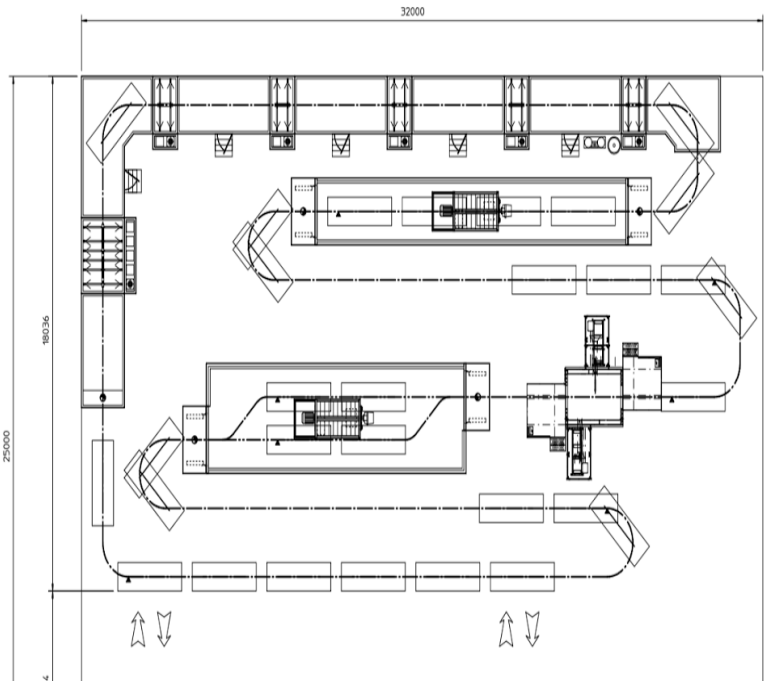
Tämän työn päätavoitteena on luoda toimiva sovellus liitettäväksi osaksi nykyistä, jo olemassa olevaa tuotantolinjaa ja kerätä analyysikelpoista tietoa linjan toiminnasta. Sovellukseen kuuluu datankeräyspalvelu, käyttöliittymäohjelma sekä niihin liittyvät tietovarastot. Ohjelma kehitetään Windows Presentation Foundation -ympäristöön, joka on Microsoftin luoma graafisten sovellusten alusta. Ohjelmointityössä käytetään Microsoftin Visual Studio -ohjelmistoa ja C#-ohjelmointikieltä. Järjestelmän kehityksessä on otettava erityisesti huomioon sen soveltuvuus myös muille tuotantolinjoille.

Lisätavoitteena on selvittää ohjelman luomiseen käytettävän viitekehyksen Windows Presentation Foundationin toimintaperiaatteita sekä siihen vahvasti liitetyn sovellusarkkitehtuurimallin model-view-viewmodel (MVVM) erityispiirteitä. Lisäksi tavoitteena on luoda luotettava ohjelma, joka kerää jatkuvasti tietoa analyyttisiin tarkoitusperiin.

Työn tarkoitus on helpottaa erityisesti uuden tai väliaikaisen työntekijän työskentelyä ohjeistamalla työntekijää tuotteen käsittelyssä. Lisäksi tarkoituksena on laajentaa mahdollisuuksia valvoa linjan toiminnan tehokkuutta analysoimalla tuotannon tunnuslukuja, kuten käyttö- ja täyttöasteita, seisausten kestoa, nopeuskerrointa sekä valmistuneiden tuotteiden laatukerrointa. Työstä kerättyjen tunnusmerkkien perusteella pyritään linjaan tekemään sen toimintaa tehostavia muutoksia.

2 JÄRJESTELMÄN KUVAUS

Käsittelylinjojen (kuvio 1) toiminnan päämäärä on yksinkertainen: tuotteen on määrä kulkea linjan läpi, minkä jälkeen sen tulee olla valmis seuraavaa työvaihetta varten. Jo pitkään on tuotannossa ollut käytössä malli, jossa linjalla työskentelevä henkilökunta on koulutettu varmistamaan linjan toimintaa ja täten tukee muutoin automaattista työtä.



KUVIO 1. Maalauslinjaston pohjapiirros

Nykyisen mallin heikkoutena on työntekijöiden sitominen yhteen työpisteeseen, mikä heikentää tuotannon joustavuutta sekä skaalautuuvuutta. Lisäksi työntekijöiden vaihtuvuus muodostuu ongelmaksi, sillä uudet työntekijät on aina koulutettava toimimaan osana tuotantoa. Ohjeistus parantaisi myös työturvallisuutta ympäristössä, jossa liikutellaan usein myös suuria ja erittäin kuumia metalliesineitä, kuten kuvassa 1.

KUVA 1. Valmis tuote tulossa käsittelyuunista ulos

Tuotantolinjat koostuvat sekä automatisoidusta että työntekijöiden tekemistä työvaiheista. Ei-automatisoituja vaiheita – joita ovat lastaus, käsinmaalaus ja purku – varten tehtiin ohjeistamisjärjestelmä, joka helpottaa linjan työntekijöiden työtä näyttämällä monitorilla ohjeen siitä, miten tuotteita tulee käsitellä. Ohjeiden näyttäminen on oleellista, sillä tuotannossa ilmenneet puutteelliset käsinmaalaukset johtavat tuotteen hylkäämiseen. Myös tuotteiden linjalle ripustamisessa on havaittu vajaatäyttöä, joka vaikuttaa suorasti linjan kokonaiskäyttöasteesen.

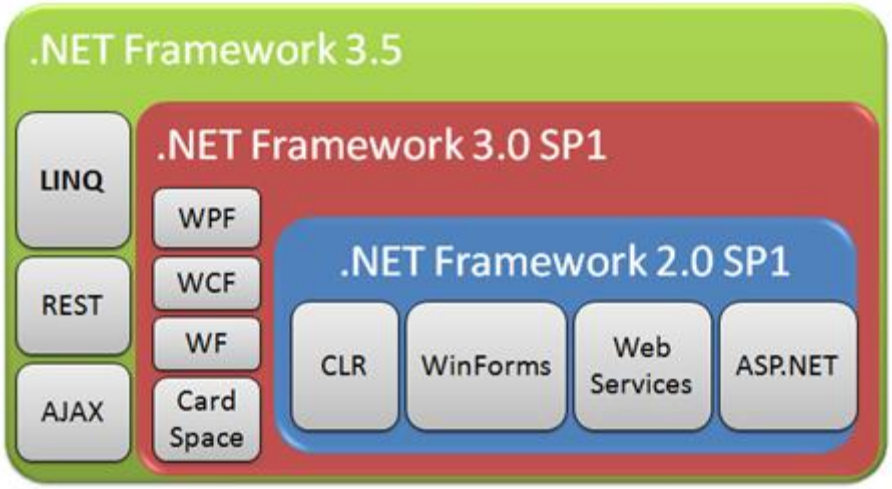
Koko tuotannon tehostamista varten luotiin tiedonkeräysjärjestelmä, joka tallentaa dataa linjan toiminnasta ympäri vuorokauden. Kerätyn datan pohjalta tehdyillä analyyseillä pyritään tunnistamaan sekä tuotantoa hidastavia että tuotteen laatuun vaikuttavia tekijöitä, joiden pohjalta voidaan tehdä muutoksia linjan toimintaan. Koska suuret linjat ja niiden osat poikkeavat usein toisistaan, on ohjelmiston arkkitehtuuri suunniteltava modulaariseksi, jotta toiminnallisuutta voi muokata asennuskohteen mukaan. Yhteinen vaatimus määrittelyissä oli kuitenkin se, että ohjelma tulee asentaa sellaiselle linjalle, jossa on toimintaa ohjaava PLC-yksikkö.

Merkittävimmin tuotteen laatuun vaikuttaviksi tekijöiksi arvioitiin muun muassa pesuaineen happamuusaste ja metalliuunin lämpötila. Ohjelman jatkokehityksen helpottamiseksi luotiin kerättyjä tietoja varten ohjelmaan asetustiedosto, jonka kautta voidaan määrittää, mitä tietoa linjasta palvelu tallentaa.

3 JÄRJESTELMÄN KEHITYS

3.1 .NET-kehitys

.NET on Microsoftin kehittämä, Windowsilla toimiva ohjelmistokomponenttikirjasto. .NETin ensimmäinen versio julkaistiin tammikuussa 2002, ja ensimmäinen kuluttajaversio on tullut saataville Windows XP-käyttöjärjestelmään. .NET-kehityksessä voi käyttää kaikkia Windowsin tukemia ohjelmointikieliä, joista käytetyimmät ovat C# ja VB.NET. .NET mahdollistaa DLL-ohjelmakirjastojen käytön, jolloin samoja ohjelman osia voidaan käyttää uudelleen myös muissa sovelluksissa.



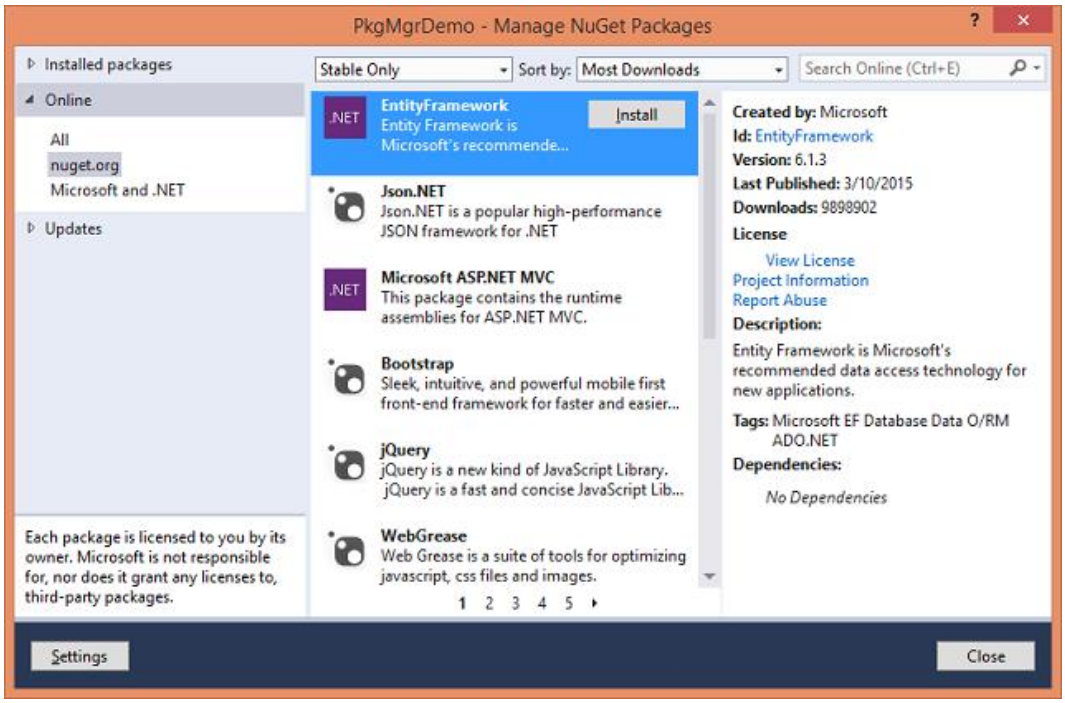
KUVIO 2. .NET-pinon kaavio (Hanselman, S. 2018)

Alkuperäisen vaatimusmäärittelyn mukaan ohjelmiston asennuskohteen alustana käytettiin Windows-tietokonetta, josta syystä .NET valittiin. Kuviossa 2 on kuvattu kaikki .NET-alustan teknologiat, jotka kytkeytyvät vahvasti Windowsin eri versioiden yhteyteen. .NET 3.0. on julkaistu alun perin Windows XP:n kanssa samaan aikaan, jolloin julkaistiin myös frontend-kehys Windows Presentation Foundation. Alustan sovellusten kehitykseen on helposti saatavilla monia laajennuksia ja niiden asennustyökalu, natiivi kehitysympäristö Microsoft Visual Studio sekä laaja tukiverkosto. (Microsoft 2017d.)

3.1.1 Entity Framework

Ohjelman määrittelyn mukaan ohjelman oli määrä tallentaa keräämänsä tiedot tietokantaan. Tietokannan datan mallintaminen ohjelman käyttöön vaatii sovittamisen tietokannan tauluista ohjelman määrittelemään tietomuotoon. Tietokannan käsittelyn helpottamiseksi päätettiin valita sovituskirjasto eli Object-Relational-Mapper (ORM). Näin tietokannassa säilytettävä staattinen data saadaan sovitettua ohjelman suorituksenaikaiseen ympäristöön (CLR, Common Language Runtime). Vaihtoehtoina punnittiin sekä hyvin pienikokoista PetaPoco-kirjastoa, joka sisällytetään ohjelmaan yksittäisenä tiedostona, että raskaampaa mutta monipuolisempaa Microsoftin kehittämää Entity Frameworkia. PetaPocon nimi on johdettu termistä POCO, joka suomennettuna tarkoittaa yksinkertaista CLR-objektia (Plain Old CLR-object). (Microsoft Developer Network 2016.)

Vaikka tietokannan ORM-kirjastoksi harkittiin myös kevyttä PetaPocoa, valittiin Entity Framework sen monipuolisten ominaisuuksien takia, joista tärkeimmäksi osoittautui tietokantamallin automaattinen päivittäminen ohjelman lähdekoodin perusteella. Entity Frameworkin voi asentaa Visual Studiosta NuGet-paketinhallintatyökalun avulla (kuvio 3).



KUVIO 3. Entity Frameworkin asennus NuGet-paketinhallintatyökalun kautta

Entity Framework on tietokannan olioiden relaatioita kartoittava viitekehys, joka vähentää tiedonhallinnallisten toimintojen tarvetta (Microsoft Developer Network 2016). Sen käyttö mahdollistaa lähdekoodipohjaisen Code First -tietokantatoteutuksen, jolloin tietokanta luodaan tietotyyppien olioiden lähdekoodin perusteella. Etuna on se, että jos tietorakenne muuttuu, tietokantaa ei tarvitse muuttaa erikseen, vaan sen taulujen rakenne päivittyy lähdekoodin muuttamisen mukana. Toisaalta ohjelman kehitysvaiheessa havaittiin, ettei Entity Frameworkin sisältämä tietokantamallin päivitys ole täysin luotettava, sillä tietokantamallia päivittäessä jouduttiin useasti tyhjentämään koko tietokanta päivitystä varten.

```
8   using System.ComponentModel.DataAnnotations;
9   using System.ComponentModel.DataAnnotations.Schema;
10  using System.Data.Entity.Infrastructure;
11  ...
15      [Table("Ripustetut")]
16      public class Ripustettu : NotifyPropertyChangedImplementor
17      {
18          [Key]
19          [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
20          public long Id { get; set; }
21
22          [Required]
23          [ForeignKey("Tilaus")]
24          public long TilausId { get; set; }
25          public virtual Tilaus Tilaus { get; set; }
26
27          [Required]
28          [ForeignKey("Vaunu")]
29          public long VaunuId { get; set; }
30          public virtual Vaunu Vaunu { get; private set; }
31
32          [Required(ErrorMessage="Ripustaessa tuotetta on asetettava lukum'a'a'r'a'.")]
33          public int RipustettuMaara { get; set; }
34
35          public DateTime RipustusAika { get; set; }
36
```

KUVIO 4. Lähdekoodipohjainen tietokannan taulun toteutus Entity Frameworkilla

Tarkemmat relaatiot tietokannan olioiden välillä sekä indeksiavaimet määritellään hakasulkujen sisällä olevilla määreillä, kuten kuviossa 4 riveillä 15 ja 18. Lisäksi Model Builderin ominaisuuksia voi muokata omiin tarpeisiinsa, esimerkkinä omien tapahtumien liittäminen tietokannan muutoksiin. Tietokannan rakenteen muuttuessa Entity Framework osaa itse luoda migraation eli päivitystiedoston, jolla päivitetään tietokannan todellinen rakenne vastaamaan mallia. Migraatiot kuitenkin epäonnistuvat usein, jolloin on riskinä tietokannan rakenteen sotkeutuminen. Tietokannan sotkeentumisen voi kuitenkin kiertää luomalla tietokannasta etukäteen kehitys- ja tuotantoversiot, jolloin sotkeentumisvaiheessa voidaan palauttaa edellinen versio ja tutkia mikä päivityksessä meni pieleen.

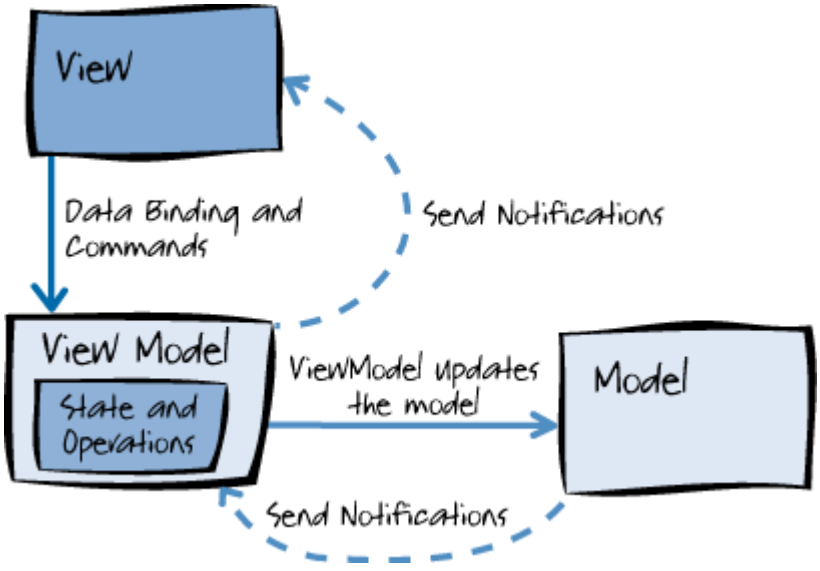
Tietokanta mallinnettiin lähdekoodin pohjalta. Näin vältyttiin SQL-skriptien kirjoittamiselta, kun tietokannan taulujen luontilauseet muodosti Entity Framework. Lisäksi varsinaiset tietokantakyselyt onnistuvat Entity Frameworkin kanssa vahvasti yhteen liitetyn LINQ-kirjaston kautta (Language Integrated Query). LINQ julkaistiin .NET version 3.5

yhteydessä vuonna 2007 (Microsoft Developer Network 2007). Pääosin Entity Frameworkin ja LINQ:n ansiosta ei projektin aikana kirjoitettu riviäkään SQL:ää.

Tietokannan selaamiseen ja ylläpitoon käytettiin ohjelmaa HeidiSQL, joka on itsenäisesti toimiva ohjelma eikä täten vaadi asennusta. HeidiSQL valittiin kehityksen työkaluksi, jotta sillä voitaisiin tarkistaa tietokannan sisältö ilman ohjelman ajamista.

3.1.2 MVVM-arkkitehtuuri

Model-View-Viewmodel on sovellusarkkitehtuurimalli, jonka tarkoitus on erotella toisistaan ohjelman graafinen toteutus (View), toiminnallisuus (View Model) ja varastoitu tieto (Model). Kuviossa 5 on kaavio arkkitehtuurin eri ohjelmistokomponenttien välisestä kommunikaatiosta. (Microsoft Developer Network 2012.) Kommunikaatio tapahtuu kuplintamallin mukaisesti siten, että ohjelman näkyvässä osiossa (View) tapahtuvat muutokset välittyvät View Modelin kautta Modelille, ja Modelin muutos vuorovaikutteisesti näkymälle.



KUVIO 5. MVVM-malli (Microsoft Developer Network 2012)

MVVM-mallissa View tarjoaa graafisen liittymän, joka välittää käyttäjän syötteet View Modelille tapahtumien kautta. View Model päivittää tietovaraston ja ilmoittaa käyttöliittymälle, mikäli sen tulee reagoida muutoksiin. View Model on toimija, joka ei suoraan muuta tai tarkastele Viewin tilaa, vaan ilmoittaa Viewille muutoksista siihen liitetyssä datassa. View itse voi olla koko ohjelman tai vain sen osan malli ja voi käyttää joko omaa tai hierarkiassa ylemmän elementin View Modelia.

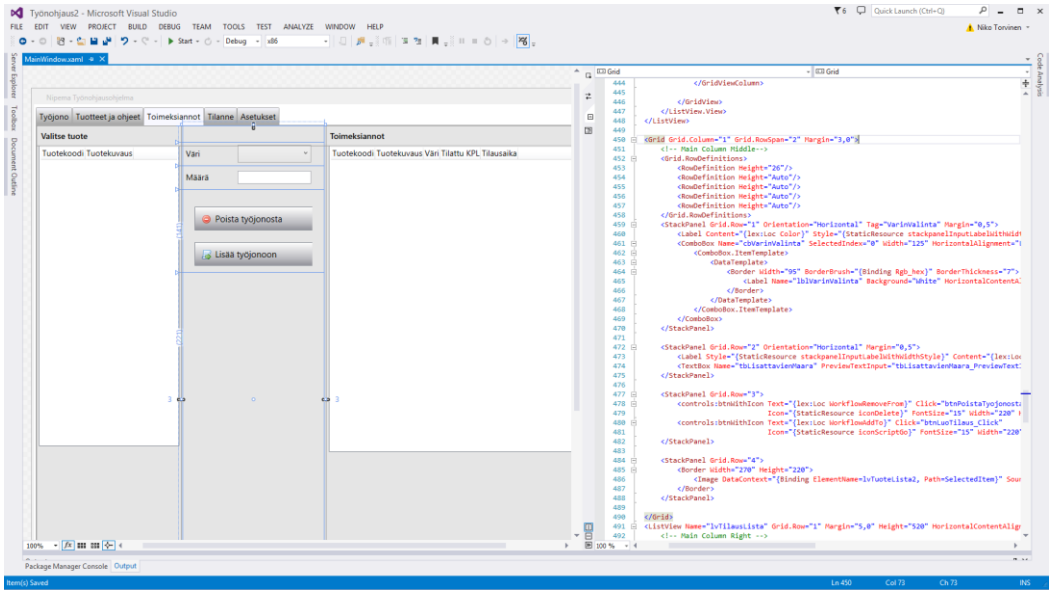
View Model -luokat toteuttavat useimmiten INotifyPropertyChanged -rajapinnan, joka sisältää tapahtuman joka ilmoittaa muutoksista olion ominaisuuksissa. Tällöin View päivittyy automaattisesti, mikäli sen Data Binding -ominaisuus on asetettu muuttuneeseen ominaisuuteen. Vaihtoehto itse toteutetuille PropertyChanged-tapahtumakutsuille WPF:ssä on luoda View Modelin ominaisuudet DependencyPropertyinä. Tällöin ohjelman sisältämä resurssienhallinta huolehtii tiedonvälityksestä View Modelilta Viewille. Lisää DependencyPropertyistä luvussa 3.2.3.

3.2 Kehitysympäristö

Työssä kehitettävä sovellus on suunniteltu asennettavaksi Windows-ympäristöön. Sen kehitys tapahtuu Microsoftin Visual Studio -kehitysympäristössä, jonka käyttö WPF:n kehityksessä on suositeltavaa. Visual Studion Express- ja Community-versiot ovat ladattavissa ilmaiseksi. Visual Studio sisältää visuaalisen XAML-suunnittelutyökalun eli Designerin sekä useita mallipohjia sekä työkaluja WPF-sovelluskehitystä varten. Mallipohjat sisältävät alkeellisen mutta toimivan tiedostorakenteen projektille sekä yksinkertaisen ohjelman tuottamiseen vaadittavat dokumentit.

Designer tarjoaa reaaliaikaisesti visuaalisen esityksen siitä, miltä ohjelman näkymä ulkoasu tai yksittäinen käyttöliittymäelementti tulee näyttämään ohjelman ajon aikana (kuvio 6). Tämän lisäksi Designerin kautta voidaan hyödyntää IntelliSense-ominaisuutta, joka tarjoaa kehittäjälle ehdotuksia esimerkiksi ominaisuuksien arvoille tai tapahtumien kytkennälle.

IntelliSensen saa käyttöön asettamalla merkintätiedoston elementille datakontekstin.



KUVIO 6. Visual Studio Designer

Designerin sisältämät elementit voi myös täyttää mallinnetulla datalla. Mallidatan käyttö tapahtuu liittämällä kehityksen aikaiset tietomallit JSON- tai XML-tiedoston muodossa projektiin. Tämän jälkeen asetetaan dataliitos mallidatan objektiin lähdekoodi- tai merkintätiedoston kautta kuten kuviossa 7.

```
<Grid d:DataContext="{d:DesignData Source=/SampleData/CustomerSampleData.xaml}">
```

KUVIO 7. Kehitysvaiheen aikaisen tietueen asettaminen XAML:n kautta

Kehitysvaiheen aikaista tietuetta apuna käyttäen voidaan kehittää käyttöliittymän asettelua ja ulkoasua, vaikka sovelluksesta ei olisi valmiina toimivaa versiota. Tätä erottelua asettelun ja toiminnallisuuden välillä voidaan käyttää hyödyksi erityisesti ohjelmiston suunnitteluvaiheessa tai tilanteessa, jossa projektissa on samanaikaisesti eri osa-alueilla työskenteleviä henkilöitä.

3.3 Datankeräyspalvelu

Koko työssä kehitettävän järjestelmän osapäämäärä on varastoida tieto kaikista linjan logiikkaohjaimen havaitsemista tapahtumista. Tiedonkeräykseen kuuluu linjaston ohjain, logiikkapalvelin, sekä palvelimelta toistuvasti dataa tietokantaan kirjoittava palvelu. Palvelin toimii siltana logiikan ja työaseman välillä, ja Windows Service -palvelu kirjoittaa arvot tietokantaan. Vaikka useat logiikkapalvelimet sisältävät tietokantaan kirjoittamisen toiminnon, todettiin itse tehdyn ratkaisun olevan turvallisempi ja edullisempi. Syynä tähän on lisenssimaksut, jotka nousevat vaadittujen pro-versioiden myötä, sekä mahdolliset yhteensopivuusongelmat.

Datankeräyspalvelu on Windows-palvelu, joka lukee logiikalta tulevan datan ja kirjoittaa sen tietokantaan. Dataa luetaan palvelinohjelman asetustiedoston mukaisesti, jolloin täytyy konfiguroida sekä PLC-ohjain että palvelin keskustelemaan samojen sääntöjen mukaan. Tiedonkeräyspalvelu alustaa komponentin, jolla luodaan yhteys palvelimeen. Komponentti on luotu avoimessa levityksessä olevalla ohjelmakirjastolla Interop.OPCAutomation.

```
152         private MyOPCClient CreateOPCClient()
153         {
154             var client = new MyOPCClient();
155             try
156             {
157                 client.server.Connect(client.progID, null);
158                 client.handshakeGroup = client.server.OPCGroups.Add("HandshakeGroup");
159                 client.trolleyLocationGroup = client.server.OPCGroups.Add("TrolleyLocationGroup");
160                 client.sensorDataGroup = client.server.OPCGroups.Add("KerattavatTiedotGroup");
161 
162                 handshake = client.handshakeGroup.OPCItems.AddItem
163                     ("Channel1.OmronCP1L.Handshakes", client.CLIENT_HANDLE);
164                 trolleyLocations = client.trolleyLocationGroup.OPCItems.AddItem
165                     ("Channel1.OmronCP1L.TrolleyLocations", client.CLIENT_HANDLE);
166                 sensorData = client.sensorDataGroup.OPCItems.AddItem
167                     ("Channel1.OmronCP1L.SensorData", client.CLIENT_HANDLE);
168 
169                 //These values are duplicates to avoid problems if index starts from 1.
170                 handle_handshake = (Array)new int[2] { handshake.ServerHandle, handshake.ServerHandle };
171                 handle_trolleyLocations = (Array)new int[2] {
172                     trolleyLocations.ServerHandle, trolleyLocations.ServerHandle
173                 };
174                 handle_sensorData = (Array)new int[2] { sensorData.ServerHandle, sensorData.ServerHandle };
175 
176                 isServerConnected = true;
```

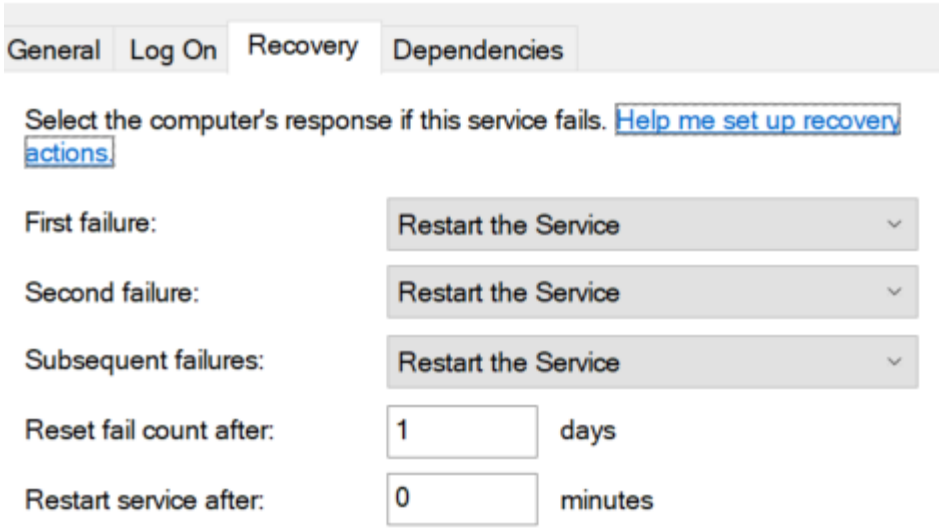
KUVIO 8. OPCAutomation-komponentin alustus

Luotuaan logiikkaohjainpalvelin KepServeriin yhteyden kiinnittyy palvelu kuuntelemaan logiikkaohjaimessa olevaa ennalta sovittua kättelybittiä (kuvion 9 rivi 158 ja 162) taajuudella 1 Hz. Kättelybitin käyttö valittiin, koska Nipema Oy:llä oli ennalta logiikkaohjaimen ohjelma, joka ilmoittaa kättelybitin kautta sensoritietojen muuttuneen. Muutokset asetetaan luettavaksi OPC-käsittelijään kuvion 8 rivillä 164 ja 174.

Havaittuaan muutoksia tiedoissa Datankeräyspalvelu käynnistää taustaprosessin, joka tallentaa tietokantaan uudet arvot luettuaan ne ensin logiikkapalvelimelta. Kun Datankeräyspalvelu on saanut tallennettua tiedot, se palauttaa kättelybitin arvon, jolloin ohjaimen muistipaikat vapautetaan seuraavaa muutosta varten. Tätä pollaukseksikin kutsuttua metodia käytetään useimmin silloin kun tietokantalähtöiset suoritteet eli Triggerit eivät sovi toteutukseen.

Vaikka palvelun alustaksi valittiin Windows Service, olisi palvelun toteutus voitu tehdä myös WCF- eli Windows Communication Foundation -palvelulla ja SOAP-rajapintaa hyödyntäen. Tällöin Datankeräyspalvelun metodeja olisi voitu kutsua suoraan Tiedonkeräysjärjestelmän asiakasohjelman kautta. Tälle ei kuitenkaan nähty tarvetta, sillä tiedot oli joka tapauksessa määrä tallentaa tietokantaan, jota kautta ne voitiin lukea asiakasohjelmasta suoraan.

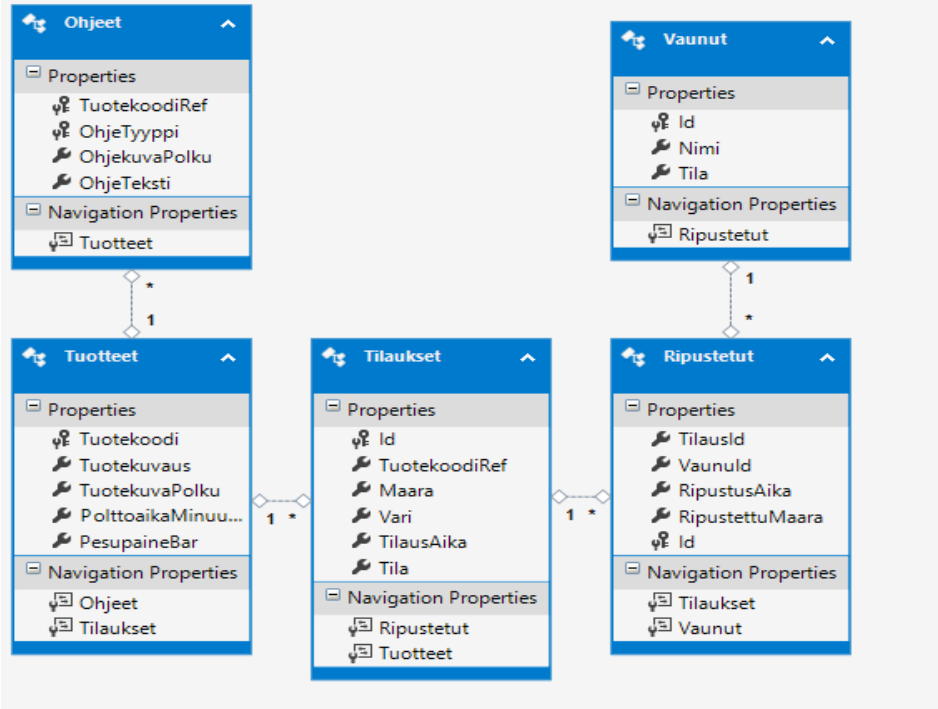
Tiedonkeräyspalvelun vaatimuksena oli kyky tallentaa ympäri vuorokauden dataa linjan tilanteesta. Jotta palvelu pysyy päällä kaikkina toiminta-aikoina, varmistetaan sen pysyvyys aloittamalla palvelun varsinainen toiminnallinen osa erillisessä säikeessä, jonka tilaa monitoroidaan. Palvelulle asetetaan Windowsin palvelunhallintatyökalusta asetus, joka käynnistää virhetilanteen sattuessa palvelun uudelleen (kuvio 9). Tällä menetelmällä havaittiin olevan hyviä tuloksia palvelun pysyvyyden suhteen.



KUVIO 9. Windowsin palvelun palautumisasetukset

3.4 Tietokanta

Tietokantamoottori on toteutettu SQL Server 2012 Express -ohjelmistolla, sillä se tukee Entity Framework-kehystä, jota ohjelmassa käytetään**.** SQL Server Express -version käyttö on maksutonta, ja sen kehitykseen on mittavasti tiedonlähteitä. SQL Serverin käytön varjopuolena on, että sen asennustiedosto on kooltaan mittavat 800 megatavua, joka moninkertaistaa järjestelmän asennustiedostojen koon levyllä ja täten hankaloittaa jakelua.



KUVIO 10. Sovelluksen tietokantamallin kaavio

Kuviossa 10 on kaaviossa kuvattuna tietokantamalli, jonka Entity Frameworkin Model Builder on luonut. Tietokantamallin kaavion generoimiseen EF sisältää valmiin työkalun, joka on käytettävissä riippumatta siitä onko EF tietokanta- vai lähdekoodipohjainen toteutus.

Muita vaihtoehtoja vastaavan tietokannan toteutukseen olivat MySQL, Oracle ja SQLite. Vaikka ADO.net -pohjaiset tietokannat mahdollistavat Entity Fameworkin käytön, valittiin MS SQL Server sen tunnettuuden ja laajamittaisen tuen vuoksi. Entity Frameworkia käsitellään tarkemmin luvussa Entity Framework.

3.5 Käyttöliittymäkehitys

Windows Presentation Foundation eli WPF on Microsoftin luoma .NET-ympäristössä toimiva graafisen käyttöliittymän viitekehys. Se pohjautuu Microsoftin aikaisempaan Windows Forms -käyttöliittymäkehykseen, mutta laajentaa sen toiminnallisuutta muun muassa resurssien, komentojen,

tapahtumien ja dataliitosten avulla. Ohjelman käyttöliittymän näkymä kehitetään Extensible Application Markup Language eli XAML-merkintäkielellä. WPF-ohjelman toiminnallisuuden eli code behindin kehitys tapahtuu millä tahansa .NET-ohjelmointikielellä, tässä työssä C#:lla. (Microsoft Developer Network 2010.)

Näkyvät elementit eli ikkunat ja kontrollit koostuvat .XAML-merkintätiedostoista (kuvio 11) sekä kooditiedostosta, jonka pääte C# kielellä kehittäessä on .CS. Ikkunoiden graafista kehittämistä varten Visual Studiossa on designer-kehitystila.

```
17      <StackPanel.DataContext>
18          <ViewModels:SingleOhjeVM />
19      </StackPanel.DataContext>
20      <Grid Height="920" VerticalAlignment="Top">
21          <Image VerticalAlignment="Top" x:Name="Ohjekuva" Height="Auto" MaxWidth="580"
22                 MaxHeight="700" Margin="10,5" Stretch="Uniform" ></Image>
23          <TextBlock x:Name="Kuvaus" FontSize="28" HorizontalAlignment="Center"
24                     VerticalAlignment="Top">
25              <TextBlock.Effect>
26                  <DropShadowEffect ShadowDepth="0" Color="White" BlurRadius="6" />
27              </TextBlock.Effect>
28          </TextBlock>
```

KUVIO 11. XAML-tiedosto

Kuten muutkin työpöytäsovellukset, WPF-ohjelmat koostuvat ikkunoista ja käyttöliittymän toiminnallisista osista eli kontrolleista. Kehitysympäristönä WPF-ohjelmille suositellaan vahvasti Visual Studiota, sillä se sisältää luontaisesti ominaisuuksia, jotka helpottavat WPF-kehitystä.

Windowsin kehittäjäyritys Microsoft käyttää WPF:ää pohjana monille sovelluksilleen, mainittavimpina Windows 10 -käyttöjärjestelmän Settings- eli asetussovellus sekä Windowsin sovelluskauppa Windows Store. Lisäksi monet suuret laitevalmistajat käyttävät sitä Windows-sovelluksissaan, kuten Lenovo sovelluksessaan Support Center.

3.5.1 WPF:n ja Windows Formsin erot

WPF muistuttaa hyvin paljon Windows Formsia, mutta laajentaa Formsin pohjalta, luoden uusia kehittämistä helpottavia työkaluja kuten DependencyPropertyt. Alusta painottaa visuaalisessa toiminnassaan nykyaikaisten kuvanohjauslaitteistojen tehokasta käyttöä. Käytännössä tämä tarkoittaa sitä, että ikkunoiden piirto tapahtuu laitteistokiihdytyksen mahdollistavalla DirectX-pohjaisella piirtoteknologialla, kun taas Forms käyttää vanhempaa GDI- eli kuvanohjauslaitteen piirtorajapintaan pohjautuvaa grafiikkateknologiaa. Formsin eduksi taas voidaan lukea sen kevyempi kehitysympäristö, sillä Visual Studion Designer-ohjelman muistinkäyttö WPF:n kehityksessä on huomattavasti Formsia raskaampaa. (Microsoft Developer Network 2010.)

3.5.2 Käyttäjän omat kontrollit

WPF:n käyttöliittymäelementeistä voi myös kehittää omia versioita, jotka saa myös näkymään suunnittelutyökalussa (kuvio 6). Jotta elementti näkyy designerissa, on sen oltava kokoonpantu tuotos, joka vastaa ominaisuuksiltaan lähdekoodia. Visual Studiossa kokoonpano tapahtuu valikosta Build. Projekti on kokoonpantava uudelleen, mikäli muutoksia luokkiin on tehty, jotta designer pystyy piirtämään ne virheettä.

Itse kehitetyt kontrollit voi luoda periytettynä mistä tahansa WPF:n oletuskontrollista, jolloin kantaluokan ominaisuudet ovat kehittäjän käytettävissä. Visual Studion uusimmat versiot sisältävät mallipohjan, joka on periytetty luokasta UserControl. UserControl on yksinkertaisin kantaluokka, josta elementin voi periyttää.

Kehittäjän luomia kontrolleja käytetään muuten samalla tavalla kuin WPF:n oletuskontrolleja, mutta koska ne sijaitsevat eri nimiavaruuksissa, on muihin kuin oletuskontrolleihin viitattava määrittäen nimiavaruuden. Nimiavaruus on ensin määriteltävä ikkunan kuvaustiedostossa, jonka jälkeen voidaan viitata.

Omia kontrolleja luodessa on tärkeää ymmärtää erot DependencyPropertyn ja luokan ominaisuuden välillä. Lisää DependencyPropertyistä luvussa 2.4.5.

3.5.3 Dataliitokset

Data Binding eli dataliitos sitoo kontrollin DependencyPropertyn liitoksen kohteeseen. Jokaisella kontrollilla on Data Context -ominaisuus, joka määrittää minkä olion ominaisuuksia tarkastellaan, kun valitaan arvo liitetylle datalle. Oletusarvoa Data Contextille ei ole, mutta sen arvo peritään hierarkiassa ylemmältä kontrollilta, ellei toisin ole määritelty. (Microsoft 2017a.)

```
<TextBox Grid.Column="1" Grid.Row="1" Text="{Binding UserName,Mode=TwoWay}" />
```

KUVIO 12. Dataliitoksen kohteen ja moodin määrittely (C-Sharp Corner 2015)

Kuviossa 12 TextBox-tyypin kontrollille asetetaan dataliitos muuttujaan UserName. Liitoksen voi määritellä joko XAML:ssa tai lähdekoodissa kuten kuviossa 12, ja sillä on oltava jokin Mode, jolla valitaan liitoksen käyttäytymiseen vaikuttava moodi. Moodityyppejä on OneWay eli datalähteeltä kontrollille, TwoWay eli kahdensuuntainen, OneWayToSource eli kontrollilta lähdekoodille ja kertaluontoinen OneTime. Käyttämällä kahdensuuntaista tai kontrollilta lähdekoodille kulkevaa liitosta voidaan päivittää ViewModelin ominaisuuksia päivitystapahtuman kutsusta. Yhdellä riippuvuusominaisuudella voi olla vain yksi dataliitos, josta se saa arvonsa, mutta yhtä datalähdettä voi käyttää monessa liitoksessa.

```
private void DataBindingMode()
{
    Binding binding = new Binding();
    binding.ElementName = "Slider1";
    binding.Path = new PropertyPath("Value");
    binding.UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged;
    txtMyTextBox.SetBinding(TextBox.TextProperty, binding);
    Binding_string.Text = "Text ={Binding ElementName=" + binding.Elem

}
```

KUVIO 13. Dataliitoksen ominaisuudet lähdekoodissa

Kuviosta 13 nähdään lähdekoodin kautta tehdyn dataliitoksen luokan Binding ominaisuuksia, joista tärkeimpiä ovat aiemmin mainittu liitosmoodi Mode, päivityksen aiheuttava UpdateSourceTrigger sekä muuttujan sijainnin asettava Path. Sijaintiominaisuus reitittää dataliitoksen tietueen sen lähteen juuresta aloittaen osoittamansa kohteen ominaisuuteen. Jokaisella kontrollilla on metodi SetBinding, jolla liitos asetetaan lähdekoodissa.

3.5.4 Tapahtumat

Kommunikaatio Viewin ja View Modelin välillä tapahtuu WPF:ssä pääosin tapahtumien aloittamana. Merkintätiedostoissa olevilla kontrolleilla on kontrollin luokasta riippuen tapahtumia, joita ne jo valmiiksi kuuntelevat, kuten Click-tapahtuma napilla tai TextChanged tekstilaatikolla. Myös omia kontrolliluokkia on mahdollista luoda ja periyttää valmiista luokista, jolloin ne voi asettaa kuuntelemaan muitakin tapahtumia. View Modelin saa ilmoittamaan tilastaan näkymälle asettamalla se toteuttamaan rajapinta INotifyPropertyChanged. (Microsoft 2017c.)

```
37      public class TilausVM : INotifyPropertyChanged
38      {
39          /* Toteutetaan INotifyPropertyChanged-rajapinta ilmoituksia varten */
40          public event PropertyChangedEventHandler PropertyChanged;
41          public void NotifyPropertyChanged(string name)
42          {
43              if (PropertyChanged != null) {
44                  PropertyChanged(this, new PropertyChangedEventArgs(name));
45              }
46          }
47          /* Kun ripustusmäärä muuttuu, lasketaan jäljelle jäävien tilausten määrä */
48          private int _ripustettavaMaara;
49          public int JaljellaOlevaMaara {
50              get { return (this.LaskeJaljellaOlevaMaara()); }
51          }
52          public int RipustettavaMaara
53          {
54              get { return _ripustettavaMaara; }
55              set
56              {
57                  _ripustettavaMaara = value;
58                  /* Ilmoitetaan Viewille muutoksesta */
59                  NotifyPropertyChanged("RipustettavaMaara");
60                  NotifyPropertyChanged("JaljellaOlevaMaara"); }
61          }
62
```

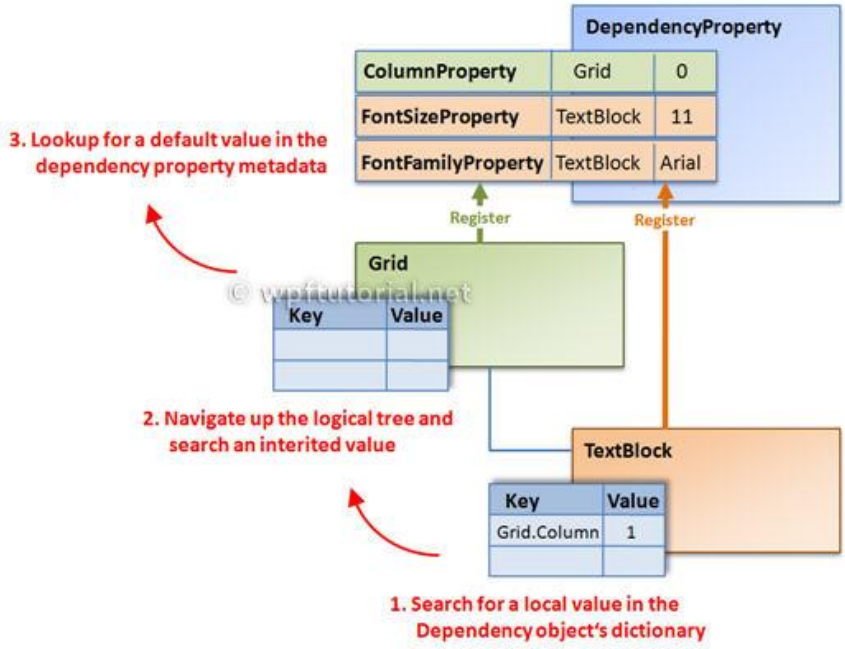
KUVIO 14. INotifyPropertyChanged-rajapinnan toteutus ja käyttö

INotifyPropertyChanged toteutetaan lisäämällä luokalle tapahtuman PropertyChanged käsittelijä. Käsittelijää käytetään kutsumalla sitä ominaisuuden muuttuessa. Parametrinä on muuttujan nimi kuten kuviossa 14. On luotava kaksi muuttujaa, joista ainakin tapahtumaa kutsuvan muuttujan näkyvyysalueen on oltava julkinen, jotta WPF huomaa muutokset. Yleinen C#:n kielikonstruktio auto-property ei ole käytettävissä, mikäli ominaisuuden tahdotaan ilmoittavan muutoksestaan, sillä auto-propertylle ei voi luoda taustamuuttujaa jonka ilmoituksen toteutus vaatii.

WPF sisältää myös rajapinnan INotifyCollectionChanged, joka ilmoittaa kokoelmamuotoisen tiedon muutoksista. Mielestäni tämä on kuitenkin vajaatoimintainen toteutus, sillä rajapinta ei ilmoita kokoelman sisällä olevan datan muutoksista, vaikka kokoelman alkiot kutsuisivatkin PropertyChanged-tapahtumaa. Sen sijaan muutoksista itse kokoelman ominaisuuksiin rajapinta ilmoittaa kyllä. Tässä projektissa vajaatoiminta on korjattu käyttämällä rajapinnasta periytettyä laajennettua versiota.

3.5.5 DependencyPropertyt

DependencyPropertyt sijaitsevat dictionary-tyyppisessä resurssikirjastossa avain-arvopareina ohjelman varaamassa välimuistissa. Poikkeuksellisesti WPF kutsuu niitä GetValue-metodilla suoraan kirjastosta, kun tavallisesti ajonaikaisia ominaisuuksia kutsutaan oliolta itseltään. WPF-ohjelma itse käsittelee oletuksena kaiken DependencyPropertyihin liittyvän toiminnallisuuden kuten arvon päivittämisen ja mahdollisen piirron. Alusta on optimoitu siten, etteivät muuttumattoman arvon säilyttävät tai käyttämättömät ominaisuudet tallennu välimuistiin, ellei toisin määrätä. Kuitenkin esimerkiksi kuvien tallentumiseen välimuistiin on valittavissa monta asetusvaihtoehtoa, joilla on vaikutusta ohjelman lopulliseen suorituskykyyn. (Microsoft 2017 b.)



KUVIO 15. DependencyPropertyn tapahtumat ja ominaisuudet kaaviona (WPF Tutorial 2010)

Mikäli jollakin kontrollin DependencyPropertyllä ei ole arvoa, ohjelma matkaa elementtien hierarkiassa ylöspäin, kunnes löytää kontrollin jolla sellainen on, tai käyttää oletusarvoa. Hierarkiaan pohjautuvan arvon

valinnan hyöty on siinä, että ominaisuudet joiden arvoa muutetaan useasta paikasta, voidaan yhtenäistää selkeillä säännöillä.

Kuviossa 15 on kaavio DependencyPropertyjen tapahtumista ja ominaisuuksista. Esimerkki DependencyPropertystä on kuviossa 14 riveillä 52-61, jossa olion kentän muutos laukaisee PropertyChanged-tapahtuman, joka ilmoittaa sitä kuunteleville käyttöliittymäelementeille muutoksesta.

3.5.6 Resurssit

Resurssit ovat objekteja jotka ohjelma tallentaa omaan pitkäaikaiseen muistiinsa, ja joita voi uudelleenkäyttää ohjelman eri osissa (Microsoft Developer Network 5). Kehittäjä voi sisällyttää ohjelmaansa resurssit joko avain-arvo-pareina XML-muodossa levyllä tai ohjelman muistiin luotuina olioina. Kummallakin tavalla luotuja resursseja ohjelmaan voi lisätä joko suoraan elementteihin, näkymiin, tai ohjelman globaaleihin resursseihin. Hierarkisesti lähinnä elementtiä oleva resurssi valitaan käyttöön. Mikäli resurssia ei löydy, asettaa viitekehys poikkeustapahtuman.

WPF laajentaa Windows Formsin käsitystä resursseista hakemalla niitä nimiavaimien perusteella valitusta resurssikirjastosta, joka on yksittäinen ilmentymä luokasta ResourceDictionary. Tällöin viitekehys voi optimoida saataville mahdollisimman nopeasti näkymän tarvitsemat resurssit, ja vapauttaa tarpeettomat.

WPF sisältää sekä dynaamisia että staattisia resursseja. Staattiset resurssit arvoineen määritellään ohjelman käynnistyessä, dynaamisia taas kutsutaan vasta silloin, kun jokin ohjelman osa haluaa sen käyttöönsä. Käyttämällä staattisia resursseja säästetään tietokoneen suorituskyvyn kuormitusta. Esimerkki staattisen resurssin käytöstä on kuviossa 16 rivillä 59, jossa asetetaan tekstikentän tyyliksi AnnotationStyle-niminen staattinen resurssi. Tämä resurssi on määritelty ohjelman resurssikirjastoon Resources.xml.

```
52          <TextBlock
53                  x:Name="tbOhjeTeksti"
54                  Width="Auto"
55                  Text="{Binding Path=Ohjeteksti}"
56                  Height="Auto"
57                  TextWrapping="Wrap"
58                  FontSize="30"
59                  Style="{StaticResource AnnotationStyle}"
60                  Margin="1,1,0,0"></TextBlock>
```

KUVIO 16. Staattinen resurssi voi olla esimerkiksi tyyli tai kuva

Tyylit merkitään WPF:ssä DependencyPropertyinä. Ohjelman käynnistyessä tyylit tallennetaan resurssikirjastoon, josta niitä kutsutaan niillä kontrolleilla, joita ne koskevat. WPF:n tyyliluokan pakolliset asetettavat ominaisuudet ovat tyylin kohde TargetType ja tyylin tunniste ResourceKey. Tyylin kohde määrittää, mitä tyyliominaisuuksia on käytettävissä. WPF-pohjaisten sovellusten käyttöliittymän tyyli mukailee usein varsinkin oletuksena Windowsin käyttöjärjestelmän ulkonäköä, mutta tyylejä voi muokata lähes rajattomasti.

3.5.7 Luokkien väliset riippuvuudet

Dependency Injection (DI) on ohjelmistokomponenttien suunnittelumalli, jossa luokkien keskinäiset riippuvuudet ratkaisee ulkoinen kirjasto. Kirjastoja löytyy useita, esimerkiksi Ninject, Autofac ja Microsoftin luoma Prism, joista kaikkien perustoiminnot toimivat samalla tapaa. Ensin riippuvuuden tyydyttävä luokka rekisteröidään yleiseen säiliöön (Container), jonka jälkeen yleensä kirjasto automaattisesti syöttää riippuvuudesta ilmoittavalle oliolle sen tarvitsemat viitteet. Sen sijaan, että jokainen tiettyä luokkaa hyödyntävä olio loisi uuden ilmentymän tarvitsemastaan luokasta, antaa luokkasäiliö omasta varastostaan viitteen itse luomaansa ilmentymään. (Microsoft Developer Network Blogs 2012.)

Dependency Injection -malli lisää sovelluksen pysyvyyttä, sillä uusia olioita luodaan harvemmin, jolloin tietyn olion tila säilyy ohjelman suorituksen

aikana vakaana eri luokkien välillä. Sen lisäksi ohjelman kehitys helpottuu, sillä kehittäjän ei itse tarvitse luoda rekisteröimiensä luokkien olioiden ilmentymiä. Käyttääkseen tätä mallia on luotava rajapintakuvaukset rekisteröitävien olioiden luokista.

WPF-sovelluksen kehityksessä on suositeltavaa noudattaa Dependency Injection -sovellussuunnittelumallia. Siinä määritellään luokan ulkoiset riippuvuudet rajapintoina, jotka ulkoinen luokkasäiliökomponentti välittää niitä kutsuvalle oliolle. Tällöin riippuvuuden kohdeluokka voidaan toteuttaa ainokais-suunnittelumallilla ilman luokan sisäisiä muutoksia, kun estetään pääsy kohdeluokan konstruktoriin muilta kuin luokkasäiliöltä. Luokkasäiliökirjasto Autofac sisältää ainokaisominaisuuden, jonka saa käyttöön kutsumalla Autofacin metodia AsSingleInstance rekisteröidessä luokan rajapintaa säiliöön.

```
8   public class AutofacWebApiDependencyResolver : IDependencyResolver
9   {
10      readonly ILifetimeScope _container;
11      readonly IDependencyScope _rootDependencyScope;
12      internal static readonly string ApiRequestTag = "AutofacWebRequest
13
14      public AutofacWebApiDependencyResolver(ILifetimeScope container)
15      {
16          if (container == null)
17              throw new ArgumentNullException("container");
18          _container = container;
19          _rootDependencyScope = new AutofacWebApiDependencyScope(contai
20      }
21
22      public ILifetimeScope Container { get { return _container; } }
23
24      public object GetService(Type serviceType)
25      {
26          return _rootDependencyScope.GetService(serviceType);
27      }
```

KUVIO 17. Autofac-luokkasäiliökirjaston käyttö

Kuviossa 17 rivillä 13 luokka saa tarvitsemansa luokan rajapinnan ILifetimeScope ilmentymän injektiona konstruktorin parametrissä. Rivillä 19 esitellään julkisen yhteyden riippuvuusluokkaan mahdollistava ominaisuus Container. Mikäli jokin toinen luokka tarvitsee ILifetimeScope-rajapintaa, voi se käyttää joko omaa injektoitua riippuvuutta tai tätä julkista

viittausta. Tästä on se etu, ettei ylimääräisiä ilmentymiä luokista luoda, vaan resursseja käytetään järkevästi optimoiden ne luokkasäiliön avulla.

3.5.8 Arvomuuntimet

Arvomuuntimet eli Value Converterit ovat WPF:n sisäänrakennettu tapa muuntaa dataliitoksen lähteen ja kohteen välillä data haluttuun muotoon. IValueConverter-rajapinta sijaitsee nimiavaruudessa System.Windows.Data, ja sen toteuttavan luokan tulee sisältää metodit Convert ja ConvertBack, jotka muuntavat datan määriteltyyn muotoon. Viitekehys itse hoitaa metodien kutsumisen tarvittaessa. Ensimmäinen kutsu tapahtuu vasta kun elementti piirretään, seuraavilla kerroilla kutsu lähetetään, jos ominaisuuden arvo muuttuu. Arvomuuntimia käyttämällä säästetään resursseja ohjelman piirtämisen osuudelta. Arvomuuntimet, kuten monet muutkin WPF:n ominaisuudet, perustuvat riippuvuusominaisuuksiin ja niiden muutosten tarkkailuun INotifyPropertyChanged-rajapinnan toteutuksen kautta.

Yleisimmin arvomuuntimia käytetään view-osiossa eli käyttöliittymän näkymissä, mutta niitä voidaan kutsua myös lähdekoodissa. Arvomuunninten hyöty on siinä, että niitä hyödyntämällä voidaan dataliitokseen lisätä logiikkaa. Arvomuuntimen avulla voi esimerkiksi muuttaa tekstikentän väriä sen perusteella, onko teksti hyväksyttävässä muodossa vai ei. Virheellisen syötteen tilanteessa voidaan tällöin käyttäjälle antaa palautetta graafisesti.

```
<Grid.Resources>
    <local:ToUpperConverter
        x:Name="ToUpperConverter" />
</Grid.Resources>

<Button Content="{Binding ButtonText,
    Converter={StaticResource ToUpperConverter}}" />
```

KUVIO 18. Arvomuuntimen lisääminen dataliitokseen.

Muunnin lisätään ohjelman resurssikirjastoon staattiseksi resurssiksi, jonka jälkeen dataliitoksen Converter-ominaisuuden arvoksi asetetaan luodun resurssin avain. Kuviossa 18 lisätään Button-elementin Content-ominaisuuden arvon liitokseen muunnin ToUpperConverter. Tuloksena kuvion tilanteessa on painike, jonka tekstisisällön kirjaimet on muunnettu suuraakkosiksi.

3.5.9 Listat ja kokoelmat

Listamuotoisen datan esitykseen WPF:ssä käytetään luokkia jotka toteuttavat System.Windows.Forms -nimiavaruuden rajapinnan ICollectionView. Se sisältää toimintoja listan tietojen lisäämiseen, rivien järjestämiseen, ryhmittämiseen ja suodattamiseen sekä valitun rivin paikan muuttamiseen (Sur 2010). Rajapinnan oletustoteutus sisältää kokoelman Items, jonka tyyppi oletusarvoisesti on ObservableCollection.

Rivejä lisätään lisäämällä kokoelmaan alkioita, jotka sisältävät ominaisuudet, joihin listan sarakkeet on liitetty. Koska kokoelman tyyppi mahdollistaa muutostapahtumat toteuttamalla rajapinnan INotifyCollectionChanged, listan näkymä päivittyy, kun sen Items-kokoelmaan lisätään tai siitä poistetaan alkioita, mutta ei jos alkion ominaisuudet muuttuvat.

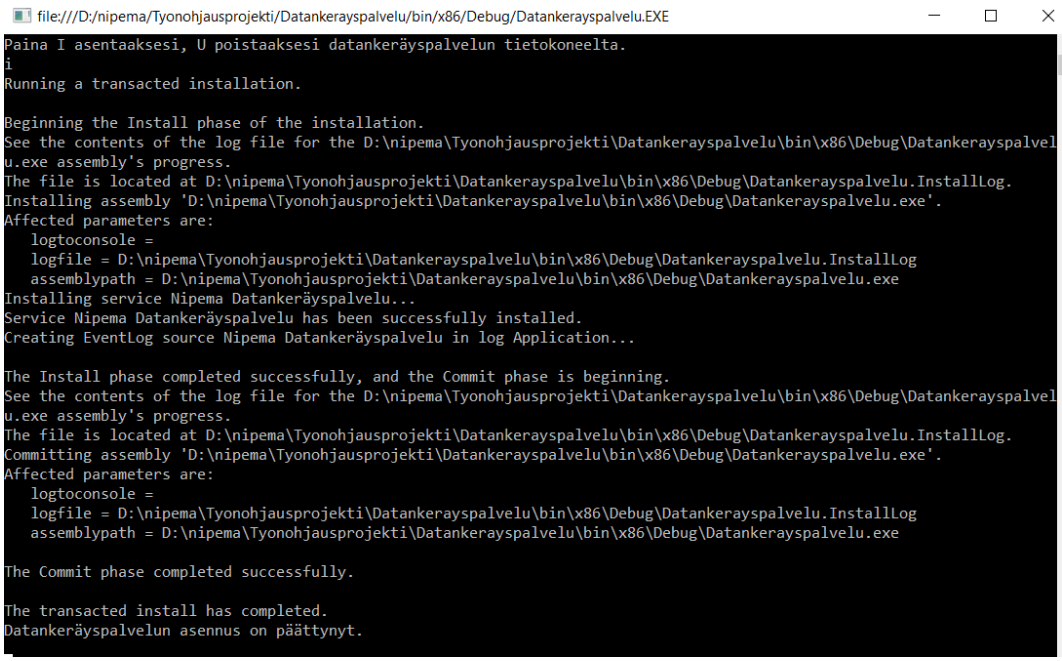
3.5.10 DelegateCommands

Microsoft itse suosittelee WPF:n kehitykseen mallia, jossa käyttöliittymän toiminnot jaotellaan yleisiin delegaattimuotoisiin komentoihin. Delegaattimuotoinen komento on ennalta kuvailtu abstrakti metodi, jonka View Model toteuttaa. Esimerkiksi näkymät, joilla on tarkoitus tallentaa dataa muodossa tai toisessa, voivat käyttää yhtenäistä komentodelegaattia, joka toteuttaa tallentamisfunktion. Mikäli tämä funktio toteutetaan delegaattina, voidaan sovelluksen käyttöliittymän toiminnallinen osuus ulkoistaa täysin ViewModel-tasolle pakottamalla ViewModel toteuttamaan abstraktin metodin tallentamiseen. Tällöin sovelluksen käyttöliittymän kehittäjän ei tarvitse tietää, mitä metodia tallennukseen varsinaisesti käytetään, vaan hän voi merkitä elementin komennoksi viitteen delegaattimetodiin, ja sovelluskehittäjä suorittaa metodin delegaation.

Tässä projektissa ei käytetty delegaattikomentojen mallia, sillä käyttöliittymän ja ViewModelin toteutuksesta vastasi sama kehittäjä - tällöin delegaateille ei ole tarvetta. Suurin hyöty delegaattikomentojen käytöstä on käyttöliittymäelementtien generalisointi.

3.6 Asennusohjelma

Jotta ohjelman levityksessä ei tarvittaisi erillistä avustusta, luotiin ohjelmistosta komentorivipohjainen automaattinen asennusohjelma (kuvio 19). Asennusohjelma asentaa tietokantamoottorin ja luo tietokannan ilman tauluja. Tietokannan taulut luo Entity Framework ohjelman käynnistyessä. Lisäksi se asentaa KepServer-logiikkapalvelimen sekä muut sen käyttämät asennettavat komponentit, joita ovat GhostScript-ohjelma PDF-tiedoston ohjelmallista lukemista varten sekä Datankeräyspalvelun.



KUVIO 19. Datankeräyspalvelun asennusohjelma

Ohjelman asennuksen jälkeen Datankeräyspalvelu käynnistyy tietokoneen käynnistyessä ja alkaa etsimään yhteyttä logiikkaohjaimeen KepServerin kautta.

Entity Frameworkin ominaisuuksiin kuuluu tietokannan rakenteen automaattinen päivitys ohjelman luokkiin tapahtuvien muutosten perusteella. Tämä mahdollistaa sen, että ohjelman jatkokehityksessä voidaan lisätä tai muuttaa kerättäviä tietoja. Muuttamalla logiikkapalvelimen asetustiedostoa voidaan linjaan lisätä mittalaitteita ja tietoa kerätä enemmän.

4 TYÖNOHJAUSOHJELMAN TOTEUTUS

Tämän opinnäytetyön käytännön osion päätavoite oli toteuttaa työnohjausohjelma maalauslinjastoille Nipema Oy:lle, joka on pintakäsittelylinjastoja toimittava yritys. Ohjelmisto asennettiin ensimmäiseksi Tume Agri Oy:lle, joka on Turengissa toimiva maatalouskoneita valmistava yritys. Maalauslinjasto oli asennettu etukäteen. Nipema Oy toimitti linjastolle ohjelmiston lisäksi kaikki tarvittavat laitteet, kuten näytöt ohjeiden näyttämistä varten sekä PC:n ohjelmiston ajamiseen.

Ohjelmistoa testattiin Tume Agrilla ja sen kehitystä jatkettiin käyttäjiltä eli tuotantolinjan työntekijöiltä saadun palautteen mukaan. Jatkossa ohjelmistoa on tarkoitus myydä myös muille maalauslinjastoille, jolloin täytyy mukauttaa ohjelman toimintaa soveltuvaksi myös eri linjalle. Ohjelmistoa kehitetään jatkossa tulevien asiakkaiden toiveiden mukaisesti. Esimerkiksi tietojen keräystä ja analysointia on määrä laajentaa, jolloin voidaan myydä eri osia ohjelmasta eri asiakkaan tarpeiden mukaan.

Asennusohjelma asentaa PC:lle tietokantaohjelman käyttämällä Microsoft SQL Serverin asennusohjelmaa omalla konfiguraatiotiedostolla (kuvio 20).

```
; Microsoft SQL Server Configuration file
[OPTIONS]
; Specifies a Setup work flow, like INSTALL, UNINSTALL, or UPGRADE.
; This is a required parameter.
ACTION="Install"
; Specifies features to install, uninstall, or upgrade.
; The list of top-level features include SQL, AS, RS, IS, and Tools.
; The SQL feature will install the database engine, replication, and full-text.
; The Tools feature will install Management Tools, Books online,
; SQL Server Data Tools, and other shared components.
FEATURES=SQL,Tools
```

KUVIO 20. MS SQL Serverin asennuksen konfiguraatio, esimerkki

Konfiguraatiotiedoston kautta asetettu Silent Mode takaa, ettei SQL Serverin asennusohjelma kysy käyttäjältä asetuksia, mutta indikoi silti

käyttäjälle asennuksen edistymisen. Asennuksen aikana ruudulla näkyy ikkuna, joka kertoo asennusvaiheen.

4.1 Laitteisto

Laitteistona käytettiin Windows-PC:n ja logiikkaohjaimen lisäksi kolmea näyttöruutua, joista kaksi yhdistettiin tietokoneeseen kytkettyyn HDMI-lähettimeen Ethernet-verkkokaapelilla. Alkuperäisen suunnitelman mukaan käyttöön oli kaavailtu VGA-kaapelia, mutta kaapelin suurin kantoetäisyys on noin 35 metriä, joka jää pienemmäksi kuin etäisyys ohjemonitoreilta PC:n keskusyksikölle. Käytettävät monitorit ovat 23-tuumaisia LCD-näyttöjä, jotka kiinnitettiin linjaa lähimpänä sijaitsevaan seinään. Monitorien pienestä koosta johtuen ohjelman käyttöliittymän suunnittelussa korostetaan suurta teksti- ja kuvakokoa. Yhdelle monitorille mahtuu yhtäaikaisesti kolme ohjetta.

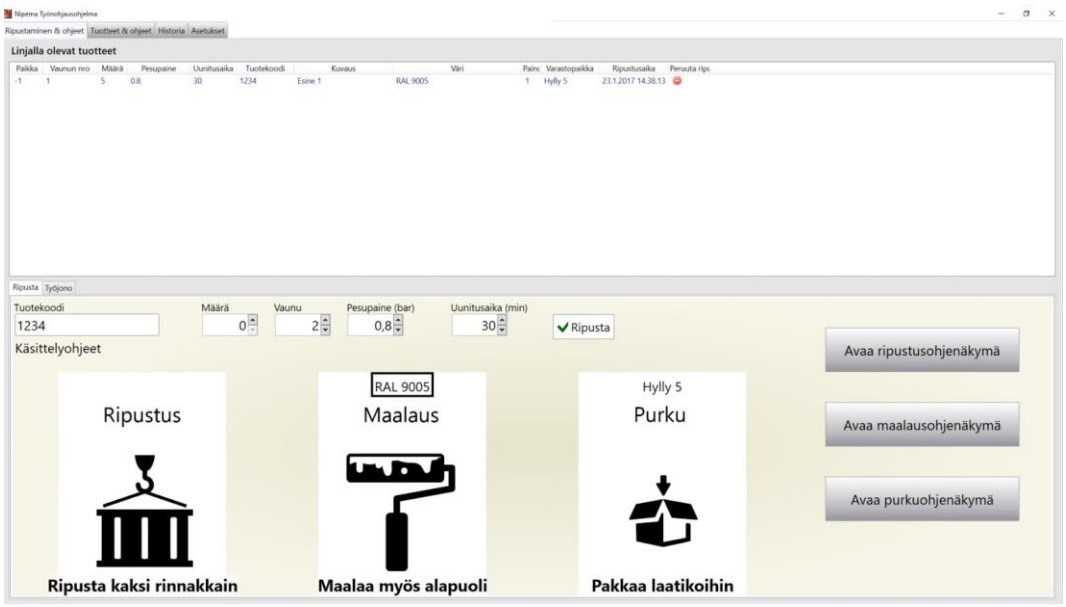
Ohjelmaa suorittava tietokone kytkettiin verkkoon, jolloin ohjelman päivitys sekä virheenkorjaus voidaan jatkossa suorittaa etäyhteyden avulla. Lisäksi linjalle asennettiin kamera, joka tunnistaa vaunujen numerot sekä mahdollistaa vaunun täyttöasteen arvioimisen tunnistamalla siihen ripustettujen esineiden pinta-alan ja välittämällä sen logiikkapalvelimelle. Kameraa ei ole kytketty tietokoneeseen, vaan sen sisältämä piiri laskee valmiiksi täyttöasteen ja välittää tiedon PLC-ohjaimelle.

4.2 Logiikkaohjain

Ethernet-kaapelilla kytkettävä ohjelmoitava logiikkaohjain, malli Omron CP1L, vastaa siitä, että jokainen linjan tapahtuma välittyy logiikkapalvelimelle. Tämä tapahtuu siten, että linjalle asennetut lukijalaitteet havaitsevat vaunuun kiinnitetyn tunnistelaatan, jolle on ohjelman tietokantaan luotu sitä vastaava alkio. Logiikkaohjain kirjoittaa lukijan asemaa linjalla vastaavaan muistipaikkaan havaitun tunnistelaatan numeron, sekä kertoo tiedonkeräyspalvelulle uutta tietoa olevan tarjolla. Lisäksi ohjain lukee antureista uunien sekä pesualtaan lämpötilat,

pesualtaan pH-arvon sekä linjan nopeuden, ja tallentaa tiedot kullekin vaunulle niiden kulkiessa pisteen ohi.

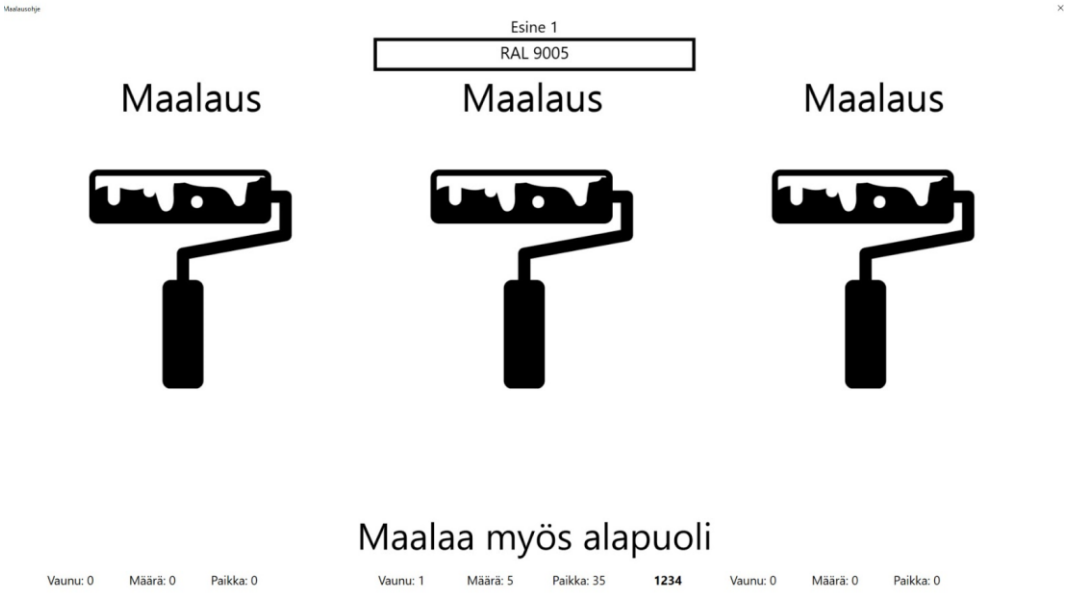
4.3 Työnohjausohjelma



KUVIO 21. Ripustaminen ja ohjeet -välilehti

Ohjelmisto avautuu kuviossa 21 näkyvään Tuotteet ja ohjeet -nimiseen välilehteen. Linjalla olevat tuotteet-listassa näkyvät tällä hetkellä linjaa kiertävät vaunut. Vaunun kierrettyä linjan, se siirtyy valmiit-listaan, jota voidaan tarkastella Historia-välilehdeltä. Ripustuksen voi peruuttaa listasta.

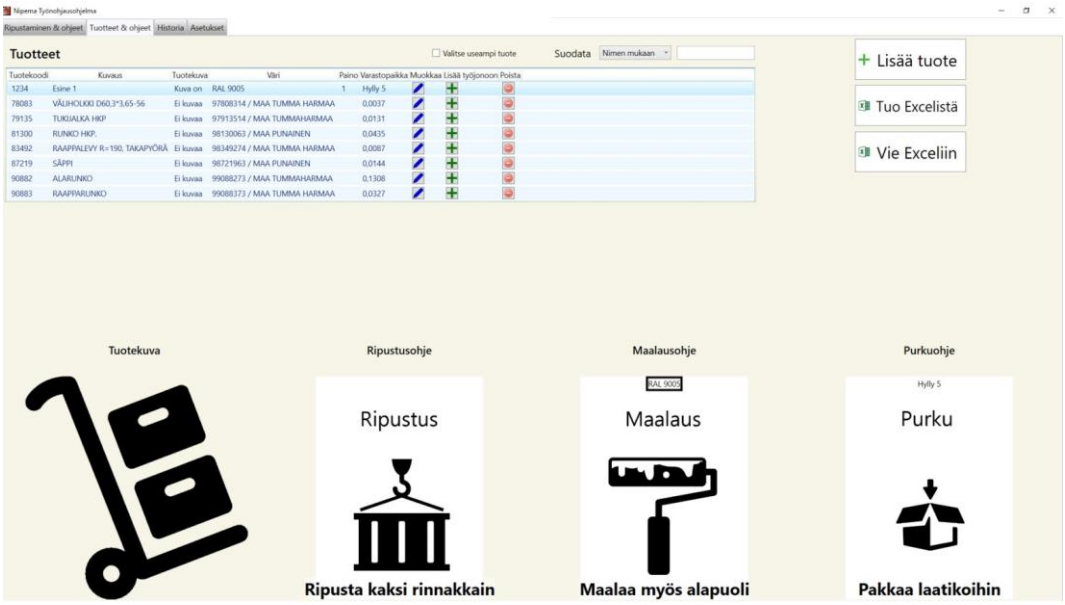
Ripusta-kohdasta tuote voidaan ripustaa linjalle. Tuotteesta tulee täyttää vaaditut tiedot: tuotekoodi, ripustettava määrä (samaan vaunuun), vaunun numero sekä pesupaine ja uunitusaika. Pesupaineen ja uunitusajan voi piilottaa asetuksista, mikäli tietoa ei linjalla tarvita, sekä määrittää oletusajan ja -paineen. Oletusarvoiksi kannattaa määrittää eniten käytetyt arvot; näitä pystyy kuitenkin myös ripustettaessa muokkaamaan.



KUVIO 22. Maalausohjenäkymä

Ripustamisen alapuolella näkyy tuotteiden ohjeet, ja niitä pääsee myös tästä muokkaamaan. Muokkaaminen onnistuu myös Tuotteet & ohjeet-välilehdeltä. Ohjeiden vieressä on napit ohjenäkymien (kuvio 22) avaamiseen. Napeista avautuu ruudun valinta-näkymä, josta voidaan valita ruutu, johon valittu näkymä avataan. Ohjenäkymä avataan sitä vastaavaan kohtaan linjalla. Ohjenäkymässä on vierekkäin kolme ohjetta: ohjekuva, ohjeteksti sekä lisätietoja tuotteesta ruudun alareunassa (vaunu, määrä, paikka, tuotekoodi). Ohjeet siirtyvät ruudussa sitä mukaa, kun tuote siirtyy linjastolla.

Ripusta-välilehden vieressä on työjono-välilehti, josta päästään työjononäkymään. Työjonolistassa näkyy kaikki työjonoon lisätyt tuotteet, kuinka monta niitä on tilattu ja kuinka monta jäljellä. Ripustaminen tapahtuu samaan tapaan kuin Ripusta-välilehdeltä. Tuotteen voi myös poistaa työjonosta. Tuotteiden lisäys työjonoon tapahtuu Tuotteet & ohjeet-välilehden tuotelistasta.



KUVIO 23. Tuotteet ja ohjeet -välilehti

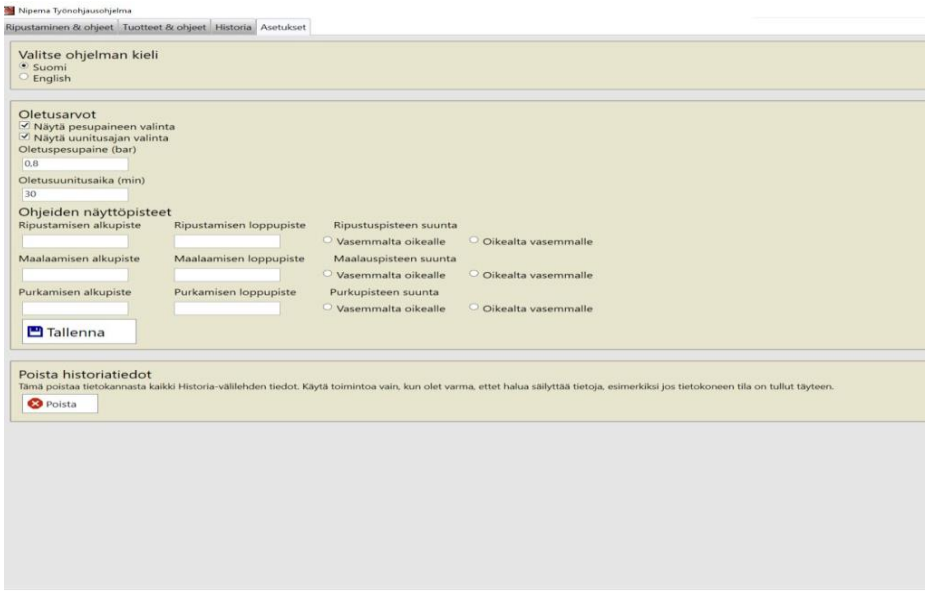
Kuviossa 23 näkyvällä tuotteet ja ohjeet -välilehdellä tuotteet-kohdassa on lista kaikista tietokannassa olevista tuotteista. Tuotetta klikkaamalla tulee alas näkymään kyseisen tuotteen ohjeet sekä tuotekuva. Ohjeita voidaan muokata myös tätä kautta ohjekuvaa klikkaamalla. Tuotteella on myös kolme nappia: muokkaa, lisää työjonoon ja poista. Tuotteita voidaan hakea yläreunasta Suodata-laatikosta.

Tuotteita voidaan lisätä tuotelistan oikealta puolelta. Lisää tuote-nappi avaa ikkunan, josta voidaan lisätä suoraan yksittäinen tuote. Tuotteita voidaan myös tuoda Excelistä. Tuo Excelistä-nappi avaa näkymän, josta valitaan Excel-tiedosto, jossa tuotavat tuotetiedot ovat. Ruutuun laitetaan rasti, mikäli Excel-tiedoston ensimmäinen rivi on otsikko. Tämän jälkeen valitaan, mikä tiedoston sarake sisältää minkäkin tiedon. Mikäli Excel-tiedoston sarake A on tyhjä ja vasta sarake B sisältää tuotekoodin, jätetään sarake A tyhjäksi ja valitaan Sarake B:n kohtaan tuotekoodi. Kaikkia tietoja ei tarvitse täyttää, esimerkiksi tuotekuvan voi jättää valitsematta. Nämä voidaan täyttää jälkikäteen ohjelman muokkaa-toiminnolla. Excel-tiedosto ei saa olla auki samanaikaisesti Excel-tuonnin kanssa.

Tietokanta voidaan myös viedä Excel-tiedostoon. Tällä toiminnolla tietokannasta voidaan ottaa varmuuskopio, tai suurta määrää tuotteita voidaan muokata ja lisätä nopeammin. Vie Excel -napista avautuu näkymä, josta valitaan tiedoston haluttu kohdekansio. Tiedostoon tallennetaan kaikki tietokannassa olevat tuotetiedot.

KUVIO 24. Historia-välilehti

Historia-välilehdeltä (kuvio 24) voidaan tarkastella linjalla aiemmin kulkeneita tuotteita. Aikajakso voidaan valita Näytä tiedot ajalta-kohdasta. Listalla näytetään määrän, tuotekoodin, ripustus- ja valmistumisajan lisäksi valitut mittaustiedot linjalta. Mittaustietoja voidaan muokata asetukset-välilehdeltä. Listan tiedot voidaan tallentaa myös Excel-tiedostoon, josta tietoja päästään tarkastelemaan eri tavoilla. Historiatiedot voidaan tarvittaessa poistaa asetukset-välilehdeltä.



KUVIO 25. Asetukset

Asetukset-välilehdeltä (kuvio 25) voidaan valita ohjelman kieli. Lisäksi voidaan määrittää pesupaineen ja uunitusajan oletusarvot sekä se, näytetäänkö niitä ohjelmassa ollenkaan; mihin suuntaan linja kulkee ohjeiden näyttöpisteissä sekä missä pisteissä näytöt sijaitsevat. Asetuksissa on myös kerättävien tietojen valinnat. Myös historiatiedot voidaan poistaa asetukset-välilehdeltä.

5 YHTEENVETO

Tämän opinnäytetyön tarkoituksena oli tutkia, miten maalauslinjaston työntekijöiden kouluttamista voisi nopeuttaa ja työntekijöiden vaihtuvuutta parantaa työnohjausohjelmiston avulla. Työnohjausohjelman on tarkoitus näyttää työntekijälle oikeat työohjeet kullakin työskentelypisteellä, esimerkiksi linjalla lähestyvän tuotteen maalausohje maalauspisteessä. Lisäksi ohjelmisto kerää tietoa linjalta, esimerkiksi uunien lämpötiloja ja vaunujen täyttöastetta.

Työnohjausohjelma toteutettiin graafisella Windows Presentation Foundation -ympäristöllä. Ohjelmointiin käytettiin C#-ohjelmointikieltä ja ohjelmointityöhön Microsoftin Visual Studio -ohjelmistoa. Tietokannan hallinnointiin käytettiin Entity Framework -viitekehystä. Sovellusarkkitehtuurin mallina käytettiin Model-View-Viewmodel-mallia, joka erottelee ohjelmiston näkymän, toiminnallisuuden sekä varastoidun tiedon.

Työn aikana havaittiin, että Windows-ympäristön sovelluskehitys onnistuu hyvin joustavalla kirjolla teknologiaratkaisuja, ja tähän projektiin valitut ratkaisut toimivat hyvin. Ongelmaksi muodostui lähinnä suuren datamäärän näyttäminen ohjelman käyttöliittymässä muistiresurssien vuoksi, sekä tietokannan rakenteen päivityksen yhteydessä ilmenneet Entity Frameworkin puutteet. Edellä mainitusta syystä ei voida suositella Code First -toteutusta, parempi vaihtoehto voisi olla Database First -malli. Siinä tietokanta luodaan ensin, minkä jälkeen generoidaan POCO-luokat luotujen taulujen pohjalta. Varsinkin, jos projektissa on mukana SQL-osaajia tai tietokantamoottorina on joku muu kuin Microsoft SQL Server, voi ORMina käyttää Entity Frameworkin sijaan kevyttä PetaPocoa.

WPF ei myöskään osoittautunut niin vahvaksi frontend-alustaksi kuin oli ennakoitu. Designer kulutti suuren määrän tietokoneen resursseista, eikä ohjelmaa voi asentaa muille alustoille kuin Windowsille. Tähän projektiin se soveltui silti hyvin, sillä kehitys WPF:llä onnistui nopeasti. WPF:n vahvuudet ovat voimakkaassa grafiikkamoottorissa ja laajassa

käyttöliittymäkomponenttien kirjastossa. Suurin osa niistä jäi kuitenkin käyttämättä, sillä ohjelmiston käyttötarkoitus ei edellyttänyt vaikuttavaa visuaalista ilmettä.

Yksinkertaisen Windows-palvelun sijasta olisi voitu käyttää joko REST- tai SOAP-pohjaista rajapintaratkaisua, jolloin asiakasohjelman kehityksen olisi voinut ulkoistaa. Tällöin ohjelman palvelinosan olisi voinut asentaa eri tietokoneelle kuin käyttöliittymäsovelluksen. Alkuperäisessä asennuskohteessa oli kuitenkin vain yksi tietokone, jossa ei ollut verkkoyhteyttä. Jatkokehitystä ajatellen olisi suotavaa erottaa toisistaan käyttö- ja palvelinlaitteet.

KUVA 2. Maalauspisteen ohjenäyttö Tume Agrin maalauslinjastolla

Työn käytännön osuuden lopputuloksena ohjelmisto asennettiin olemassa olevalle maalauslinjastolle, ja sen testaaminen aloitettiin. Kuvassa 2 näkyy linjan maalauspisteen ohjenäyttö. Testauksen kuluessa ohjelmistosta korjattiin virheitä sekä lisättiin puuttuvia ominaisuuksia. Käytännön testeissä parannusehdotuksia havaitsi paremmin kuin ilman aitoa testiympäristöä. Myös kyseisen asiakkaan omat tarpeet otettiin huomioon ja ohjelmistoa räätälöitiin niiden mukaan. Lopputuloksena asiakas oli tyytyväinen toimitettuun ohjelmistoon, ja aloittivat sen käyttöönoton ja

testauksen. Ohjelmiston asentamisesta myös muille linjastoille on jo alustavasti keskusteltu, ja ohjelmiston kehitys jatkuu niiden mukaan.

6 LÄHTEET

C-Sharp Corner 2015. WPF Interview Questions And Answers [viitattu 20.4.2017]. Saatavissa: http://www.c-sharpcorner.com/UploadFile/8ef97c/most-asked-wpf-interview-questions-and-answer

Elinkeinoelämän tutkimuslaitos 2014. Digibarometri 2014: Huipputason edellytyksillä keskinkertaisia tuloksia [viitattu 27.5.2017]. Saatavissa: https://www.etla.fi/uutiset/digibarometri-2014-huipputason-edellytyksilla-keskinkertaisia-tuloksia/

Hanselman, S. 2018. How to set an IIS Application or AppPool to use ASP.NET 3.5 rather than 2.0 [viitattu 18.4.2018]. Saatavissa: https://www.hanselman.com/blog/HowToSetAnIISApplicationOrAppPoolToUseASPNET35RatherThan20.aspx

Kauppalehti 2017. Yrityshaku: Nipema Oy [viitattu 9.3.2018]. Saatavissa: https://www.kauppalehti.fi/yritykset/yritys/nipema+oy/06164532

Microsoft 2017a. Data Binding Overview [viitattu 6.2.2018]. Saatavissa: https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview

Microsoft 2017b. Dependency Property Overview [viitattu 6.2.2018]. Saatavissa: https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/dependency-properties-overview

Microsoft 2017c. How to Implement Property Change Notification [viitattu 11.1.2018]. Saatavissa: https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/how-to-implement-property-change-notification

Microsoft 2017d. .NET Framework Versions and Dependencies [viitattu 6.2.2018]. Saatavissa: https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies

Microsoft Developer Network 2007. LINQ: .NET Language-Integrated Query [viitattu 27.1.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/bb308959.aspx

Microsoft Developer Network 2010. Introduction to WPF [viitattu 27.1.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/aa970268(v=vs.100).aspx

Microsoft Developer Network 2012. The MVVM Pattern [viitattu 15.1.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/hh848246

Microsoft Developer Network 2016. Introduction to Entity Framework [viitattu 9.3.2017]. Saatavissa: https://msdn.microsoft.com/en-us/data/ef.aspx

Microsoft Developer Network 2017. XAML Resources [viitattu 27.5.2017]. Saatavissa: https://msdn.microsoft.com/en-us/library/ms750613(v=vs.110).aspx

Microsoft Developer Network Blogs 2012. Dependency Injection with ASP.NET Web API and Autofac [viitattu 6.3.2018]. Saatavissa: https://blogs.msdn.microsoft.com/roncain/2012/07/16/dependency-injection-with-asp-net-web-api-and-autofac/

Stephens, R. 2010. WPF Programmer’s Reference. Wiley Publishing, Inc. Indianapolis: Wiley Publishing, Inc.

Sur, A. 2010. Working with CollectionView in WPF [viitattu 20.4.2017]. Saatavissa: http://www.abhisheksur.com/2010/08/woring-with-icollectionviewsource-in.html

Valtioneuvosto 2014. Digitalisaatio keskisuurissa yrityksissä [viitattu 27.5.2017]. Saatavissa: https://julkaisut.valtioneuvosto.fi/bitstream/handle/10024/77886/Julkaisuja_14-2014.pdf

WPF Tutorial 2010. Dependency Properties [viitattu 30.3.2018].
Saatavissa: https://www.wpftutorial.net/DependencyProperties.html