MVVM-arkkitehtuurimallia noudattava WPF-sovellus asiakirjojen tulostamiseen

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeenlinnan ammattikorkeakoulukeskus,
Tietojenkäsittelyn koulutusohjelma

Kevät, 2020

Tomi Korkalainen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tradenomi, Tietojenkäsittely
Hämeenlinnan ammattikorkeakoulukeskus

Tekijä Tomi Korkalainen **Vuosi** 2020

Työn nimi MVVM-arkkitehtuurimallia noudattava WPF-sovellus asiakirjojen tulostamiseen

Työn ohjaaja Erkki Laine

TIIVISTELMÄ

Monella organisaatiolla, erityisesti julkishallinnossa on tarve hallita asioita ja asiakirjoja. Tähän tarpeeseen on luotu sähköisiä asianhallintajärjestelmiä, jotka tehostavat asioiden ja asiakirjojen käsittelyä niiden koko elinkaaren ajan.

Työn toimeksiantaja Triplan Oy on erikoistunut julkishallinnon asioiden ja asiakirjojen hallintaan. TWeb on Triplan Oy:n kehittämä asianhallintajärjestelmä, jonka lukuisiin toiminnallisuuksiin lukeutuu asiakirjojen tulostaminen työaseman oletuskirjoittimelle. Tämä tulostustoiminnallisuus on toteutettu teknologialla, jonka tuesta suurin osa selaimista on luopunut.

Työn tarkoituksena oli luoda korvaava, Windows-työasemilla toimiva työpöytäsovellus asiakirjojen tulostukseen toimeksiantajan järjestelmästä. Työssä käsiteltiin WPF-käyttöliittymäkehystä ja MVVM-arkkitehtuurimallia, joita hyödyntämällä sovellus toteutettiin. Työ käsittelee sovelluksen ennalta määritetyn käyttötapauksen, suunnitelman ja toteutuksen. Sovelluksen ulkoasu ja toiminallisuudet pyrittiin pitämään samankaltaisena kuin aiemmassa toteutuksessa.

Työn tulos on yksinkertainen ja toimiva Windows-työpöytäsovellus, joka mahdollistaa sille määritetyn käyttötapauksen. Sovellus muistuttaa aiempaa toteutusta niin ulkoisesti kuin toiminnallisuuksiltaankin.

Avainsanat MVVM, WPF, .NET, C#, XAML

Sivut 40 sivua, joista liitteitä 3 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Bachelor’s Degree programme
Business Information Technology
Hämeenlinna University Centre

Author Tomi Korkalainen **Year** 2020

Subject WPF application with MVVM pattern for file printing

Supervisor Erkki Laine

ABSTRACT

Many organizations, especially in public administration, have a need to manage affairs and documents. To this end, electronic case management systems have been set up to streamline the handling of cases and documents throughout their life cycle.

The client, Triplan Oy, specializes in the management of public administration cases and documents. TWeb is a case management system developed by Triplan Oy, whose numerous functionalities include printing documents on the workstation's default printer. This printing functionality is implemented with technology that most browsers have removed support for.

The purpose of the work was to create a replacement desktop application running on Windows workstations for printing documents from the client's system. The work dealt with the WPF user interface framework and the MVVM architecture model, which were used to implement the application. The work deals with a predefined use case, plan and implementation of the application. The layout and functionalities of the application were sought to be similar to the previous implementation.

The result of this work is a simple and functional Windows desktop application that enables the use case assigned to it. The application resembles a previous implementation both externally and in terms of functionality.

Keywords MVVM, WPF, .NET, C#, XAML

Pages 40 pages including appendices 3 pages

KÄSITELUETTELO

MVVM	Model-View-ViewModel (MVVM) on ohjelmistoarkkitehtuurimalli, joka on yleisesti käytössä WPF-sovelluksissa
WPF	Windows Presentation Framework (WPF) on käyttöliittymäkehys Windows-työpöytäsovelluksien kehittämiseen
.NET	.NET on Microsoftin kehittämä luokkakirjastoista ja ajoympäristöstä koostuva ohjelmistokomponenttikehys
XML	Extensible Markup Language (XML) – merkintäkieli, joka on suunniteltu datan säilyttämiseen ja siirtämiseen
XAML	XML-merkintäkieleen pohjautuva kieli .NET-olioiden luomiseen ja määrittämiseen
C#	C# (C-Sharp) on .NET-kehykselle luotu ohjelmointikieli
Visual Basic	Microsoftin kehittämä ohjelmointikieli
Presentation Model	Presentation Model (PM) on ohjelmistoarkkitehtuurillinen malli, johon MVVM-malli perustuu
Java	Java – luokkaperustainen, oliokeskeinen ohjelmointikieli
Java SE 8	Java Standard Edition (SE) 8 – Java-jakelu, jolla Java-kielellä luotuja sovelluksia ajetaan
MVC	Model-View-Controller (MVC) on ohjelmistoarkkitehtuurillinen malli, jonka tarkoitus on erottaa käyttöliittymä sovelluslogiikasta
MVP	Model-View-Presenter (MVP) on MVC-mallista periytyvä arkkitehtuurillinen malli, jota käytetään pääasiallisesti käyttöliittymien tekemisessä.
Separation of Concerns	Separation of Concerns (SoC) – Sovelluskehityksessä käytetty suunnitteluperiaate, sovelluksen erottamiseksi erillisiksi osioiksi siten, että kukin osa käsittelee erillistä huolenaihetta
Www-sovelluspalvelu	Www-sovelluspalvelu (Web Service) on ohjelmistojärjestelmä, joka mahdollistaa tietokoneiden välisen vuorovaikutuksen tietoverkon yli

SISÄLLYS

1 JOHDANTO 1
2 MVVM-ARKKITEHTUURIMALLI 2
2.1 MVVM-mallin rakenne ja tiedon sidonta 3
2.2 Hyödyt ja haitat 4
3 WPF-KÄYTTÖLIITTYMÄKEHYS 6
3.1 XAML-merkintäkieli 6
3.2 Kontrollit 9
3.3 Looginen ja visuaalinen puu 10
3.4 Reititetyt tapahtumat 12
3.5 Tiedon sidonta 14
3.5.1 Tiedon muuntaminen 17
3.6 Komennot 19
3.7 Tyylit 21
4 TYÖN TAVOITE 23
5 SUUNNITTELU JA KÄYTETYT KIRJASTOT 25
5.1 Asiakirjojen haku, noutaminen ja tulostaminen 25
5.2 Käytetyt kirjastot 26
6 TOTEUTUS 28
6.1 Malli 28
6.2 Asiakirjojen hakuikkuna 29
6.3 Asiakirjojen tulostuksen pääikkuna 31
6.4 Asiakirjojen tulostuksen tilanneikkuna 33
7 YHTEENVETO 35
LÄHTEET 36

Liitteet
Liite 1 TPrinter – Tulostaminen aiemmalla sovelluksella

1 JOHDANTO

Monella organisaatiolla, erityisesti julkishallinnossa, on tarve hallita asioita ja asiakirjoja. Tähän tarpeeseen on luotu sähköisiä asianhallintajärjestelmiä, jotka tehostavat asioiden ja asiakirjojen käsittelyä koko niiden elinkaaren ajan. Asianhallintajärjestelmillä pyritään tehostamaan asioiden valmistelua, käsittelyä, päätöksentekoa, julkaisemista ja arkistointia sekä asiakirjamuodossa olevien tietojen hallintaa.

Työn toimeksiantaja Triplan Oy on ohjelmistotalo, joka on erikoistunut julkishallinnon asioiden ja asiakirjojen hallintaan. TWeb on Triplan Oy:n selainkäyttöinen asianhallintajärjestelmä, jonka toiminnallisuuksiin kuuluu muun muassa haettujen asiakirjojen tulostaminen työaseman tulostimelle. Tämä tulostustoiminnallisuus on järjestelmästä irrallinen Java-sovelma, joka on toteutettu käyttäen selaimiin kehitettyä NPAPI-rajapintaa, jonka tuesta suurin osa selaimista on luopunut. Sovelma on vaatinut sitä käyttäneelle työasemalle Java asennuksen. Tarkemmin ottaen, työasemalle on asennettava Java SE 8, joka on viimeisin NPAPI-rajapintaa tukeva Java-jakelu. Kyseinen Java-jakelu tarvitsee tätä nykyä maksullisen lisenssin kaupallista uusimpia tietoturvapäivityksiä varten. Täten Triplan Oy:lle on tullut tarve luoda korvaava ratkaisu asiakirjojen tulostamiseen.

Opinnäytetyön tavoitteena on luoda toiminnallisuuden korvaava Windows-työpöytäsovellus, johon käyttäjä voi kirjautua asianhallintajärjestelmän tunnuksilla, hakea ja tulostaa halutut asiakirjat sekä kirjautua ulos. Aiemmin luotua tulostustoimintoa voidaan pitää vertauskohteena, jonka toiminnot työn on tarkoitus ensisijaisesti toteuttaa. Parhaimmillaan työkalu tarjoaisi käyttäjilleen aiempaa ratkaisua muistuttavat tulostustoiminnallisuudet, vain lisäten toiminnallisuuksia sen päälle ja parantaen sen toiminnallisuuksia entisestään.

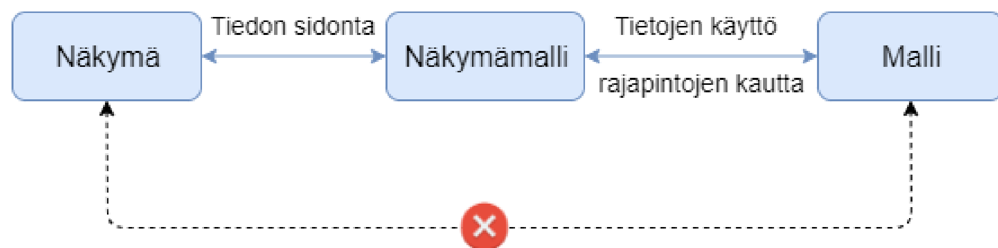
Työn tekijä sai vapaat kädet tehdä teknologioihin ja toteutustapoihin liittyviä ratkaisuja. Sovelluksen käyttämäksi alustaksi valittiin WPF-käyttöliittymäkehys. Lisäksi sovellus suunnitellaan ja kehitetään MVVM-arkkitehtuurimallia noudattaen. Sovellusalustan valintaan vaikutti sillä tuotetun sovelluksen toimivuus Windows-käyttöjärjestelmissä, tekijän aiempi osaaminen ja tarvittujen toiminnallisuuksien toteuttamisen helppous. WPF-kehyksellä luodut sovellukset tyypillisesti toteutetaan MVVM-arkkitehtuurimallin mukaisesti, mikä mahdollistaa helpomman ylläpidettävyyden ja jatkokehittämisen.

Opinnäytetyön keskeisenä tavoitteena on vastata kysymyksiin:

- Mikä on MVVM-arkkitehtuurimalli?
- Mikä on WPF-käyttöliittymäkehys?
- Miten MVVM-mallia noudattava tulostustyökalu voidaan toteuttaa WPF-kehyksellä?

2 MVVM-ARKKITEHTUURIMALLI

Model-View-ViewModel (MVVM) on sovellusarkkitehtuurimalli, jonka John Gossman, Microsoftin sovellusarkkitehti julkaisi blogissaan vuonna 2005. Sen päämääränä on jakaa sovellus Separation of Concerns -suunnitteluperiaatteen mukaisesti kolmeen osa-alueeseen: liiketoimintamalliin, käyttöliittymään ja liiketoimintalogiikkaan. Se pohjautuu Presentation Model -malliin, jonka päämääränä on erottaa näkymä eli käyttöliittymä ja sovelluksen esityslogiikasta. MVVM-mallin päämääränä oli ratkaista MVP- ja MVC-mallien ongelmia ja osin yhdistää niiden vahvuuksia. (Garofalo, 2011, s. 1; Vice & Siddiqi, 2012; Yuen, 2017)



Kuvio 1. MVVM-arkkitehtuurimallin rakenne (Perustuu: Yuen, 2017)

MVVM-arkkitehtuurimallin rakenne (kuvio 1) koostuu kolmesta pääasiallisesta komponentista: mallista (model), näkymästä (view) ja näkymämallista (viewmodel). Näkymän, näkymämallin ja mallin vastuualueiden lisäksi on tärkeätä ymmärtää kuinka ne keskustelevat toistensa kanssa (Microsoft, 2017 a). Malli edustaa sovelluksen toiminnan kannalta oleellisia entiteettejä, kun taas näkymässä määritetään miltä sovellus näyttää. Näkymämalli sisältää käyttöliittymälogiikan ja tarvittavat viittaukset malleihin. (Garofalo, 2011, s. 1-39)

MVVM-mallin edellyttämä tiedon sidonta jää Yuenin mukaan usein vähälle huomiolle aihetta käsiteltäessä. Tiedon sidonta tarjoaa helpon kommunikointitavan näkymän ja näkymämallien välille ja se on edellytys Separation of Concerns -suunnitteluperiaatteen täydelle toteutumiselle. (Yuen, 2017)

Käyttöliittymässä käytetään tiedon sidontamerkintää, jolla se yhdistetään sovelluksen tietoa ja komentoja sisältäviin osiin. Tiedon sidontarakenne tarjoaa liitoksen, mikä pitää käyttöliittymän ja siihen yhdistetyn datan synkronisoituna ja reitittää käyttäjän syötteet tarkoituksenmukaisiin komentoihin. Tiedon sidonnan tarjoamat liitokset vähentävät riippuvuuksia koodin eri osa-alueiden välillä. Tämä helpottaa koodin muokkaamista sen eri osa-alueissa aiheuttamatta tahattomia sivuvaikutuksia muissa sovelluksen osa-alueissa. (Microsoft, 2018 a)

2.1 MVVM-mallin rakenne

Mallin voidaan ajatella kuvastavan sovelluskehykseen liittyvää toimialan mallia ja se kuvastaakin sovelluksen ei visuaalisia osia, jotka koteloivat sovelluksen käyttämän datan (Microsoft, 2017 a). MVVM-arkkitehtuurimallissa malli sisällyttää useita elementtejä. Se käsittää sovelluksen käyttämän datamallin ja siihen liittyvien validointilogiikkojen lisäksi datan käyttökerroksen tai tietovaraston, joka tarjoaa sovellukselle pääsyn dataan ja tiedon pysyvyyden. (Yuen, 2017)

Datamalli kuvaa usein tietokannan tauluja, vaikka datamallin hierarkisesta muodosta johtuen on tyypillistä, että tietolähteeltä saatua dataa joudutaan liittää toisiinsa datamallin kenttien täyttämiseksi. Malli sisältää myös tietojen pysyvyystekniikkaan yhtymiseen tarvittavan ohjelmakoodin. (Yuen, 2017)

MVVM-mallin näkymät määrittelevät käyttöliittymän ulkonäön ja sommittelun. Ne käyttävät hyväksi näkymämallin tarjoamia toiminnallisuuksia liittämällä sen komentoja käyttöliittymän ohjaimiin, joihin sovelluksen käyttäjät ovat vuorovaikutuksessa. Näkymä on vastuussa tiedon näyttämisestä ja käyttäjän syötteen taltioimisesta ja välittämisestä. Näkymän syötteestä taltioitu tieto välitetään näkymämallille käyttäen tiedon sidontaa. (Vice & Siddiqi, 2012; Yuen, 2017)

Tyypillisesti jokaisella näkymällä on oma näkymämalli. Näkymään liittyvä näkymämalli usein nimetään niin, että näkymä nimeltä autonäkymä (CarView) saa näkymämallikseen autonäkymämallin (CarViewModel). Tämä ei kuitenkaan tarkoita, että näkymä ei voisi käyttää tehdä tiedonsidontaa useamman tiedonlähteen kanssa tai, että näkymämalleja ei voisi uudelleen käyttää. MVVM-mallista löytyy joitakin variaatioita ja osassa niistä näkymälle on annettu lisävelvollisuudeksi vastata näkymien kartoituksesta ja näkymäilmentymien luomisesta. (Vice & Siddiqi, 2012; Yuen, 2017)

Näkymämallia voidaan pitää välikätenä MVVM-mallissa. Se on yhteyksissä sekä näkymään että malliin. Näkymämalli tarjoaa siihen yhdistetylle näkymälle datan ja sen tarvitseman toiminnallisuuden. Näkymämalli vastaa näkymän tilan hallinnasta ja logiikasta ja käyttää tiedon sidontaa näkymän kanssa kommunikointiin. Näkymämalli kommunikoi sovelluksen käyttäjän syötteet ja eleet näkymältä mallille. (Vice & Siddiqi, 2012; Yuen, 2017)

Näkymämalleilla on kaksisuuntainen yhteys mallikomponentteihin, jotta ne voivat päästä käsiksi, ja muokata päivittää tietoa, joita näkymät tarvitsevat. Mallilta saatua tietoa joudutaan usein muuttaa, jotta siitä saadaan sopiva näkymälle. Näkymän ja näkymämallien välillä on myös kaksisuuntaisia yhteyksiä tiedon sidonnan ja ominaisuuksien muutosilmoituksien kautta. (Yuen, 2017)

2.2 Hyödyt ja haitat

MVVM jakaa Separation of Concerns -suunnitteluperiaatteen mukaisesti sovelluksen kolmeen eri osa-alueeseen, mikä voidaan nähdä yhtenä päähyötynä sen käyttöönotossa. Se vapauttaa näkymämallit malleista ja sen vastuualueeseen kuuluvista toiminnoista kuten datamallin määrityksestä ja tiedon pysyvyyteen liittyvistä teknologioista. Toiminnallisuuksien erottaminen mahdollistaa liiketoimintamallin uudelleenkäytön muissa sovelluksissa ja datan käyttökerroksen vaihdon harjoitusversioon, jotta sovelluksen toiminnallisuutta voidaan testata näkymämalleissa ilman, että tarvittaisiin todellista tietoyhteyttä. (Yuen, 2017)

Voi olla vaikeaa tai riskialtista muokata jo olemassa olevaa mallitoteutusta, joka sisältää tarvittavan liiketoimintalogiikan. Tällaisessa tilanteessa näkymämalli toimii mallin sovittimena ilman, että täytyy tehdä muutoksia mallin koodiin. (Microsoft, 2017 a) MVVM-mallin noudattaminen luo sovelluksesta helpommin testattavan. Suurin muutos testattavuudessa liittyy näkymään, sillä kaikki näkymälogiikan osat pystytään testauttamaan. (Vice & Siddiqi, 2012)

Näkymän logiikan irrottaminen näkymästä näkymämalliin mahdollistaa niiden ajamisen ja kehittämisen erillään. Osa kehittäjistä voi suunnitella näkymiä, kun toinen osa kehittää niille näkymämalleja. Rinnakkainen työskentely sovelluksen eri osa-alueissa mahdollistaa huomattavasti lyhyemmän valmistusajan sovellukselle. (Yuen, 2017)

MVVM-mallin luoma Separation of Concerns -jaottelu helpottaa myös tilanteessa, jossa näkymä vaihdetaan käyttämään eri teknologiaa. Kyseisessä tilanteessa näkymämalleihin voidaan tarvita joitakin muutoksia, mutta mallit voivat pysyä ennallaan, mikä mahdollistaa muutoksen toteuttamisen melko vähäisillä muutoksilla. (Yuen, 2017)

MVVM-mallin yksinkertaisuus tekee myös sovelluskehityksestä helpommin ymmärrettävää. Kun tietää, että jokaisella näkymällä on näkymämalli, joka tarjoaa sille sen tarvitsemat datat ja toiminnallisuudet, on helpompaa etsiä missä näkymälle määritetyt ominaisuudet on määritetty. (Yuen, 2017)

MVVM-mallin käytössä on myös huonoja puolia ja siitä ei ole hyötyä joka tilanteeseen. Sen toteuttaminen kasvattaa sovelluksen monimutkaisuutta, mitä voidaan pitää MVVM-mallin pääheikkoutena. Tiedon sidonta vaatii sovelluskehittäjiltä aikaa oppia, eikä sen käyttö ole aina vaivatonta sillä ongelmat saattavat ilmentyä vasta ajon aikana ja niiden jäljittäminen voi olla haastavaa. (Yuen, 2017)

Päteväksi kehittyminen MVVM-mallissa vaatii paljon opettelua. Näkymien ja näkymämallien väliseen tiedonsiirtoon on useita eri tapoja, jotka täytyy ymmärtää. Tapahtumien käsittely epätavallisella tavalla vie aikaa tottua.

Optimaalisen järjestelyn löytäminen sovelluksen tarvitsemille komponenteille vie myös aikaa. (Yuen, 2017)

Vaikka MVVM-mallin käyttö oltaisiinkin jo omaksuttu, voi olla tilanteita, joissa sen käyttö ei kannata. Jos sovellus esimerkiksi on erittäin pieni, on epätodennäköistä, että sille halutaan toteuttaa yksikkötestausta tai vaihtaa sen osia. Olisi epäkäytännöllistä kasvattaa sovelluksen monimutkaisuutta ottamalla MVVM-malli käyttöön, kun Separation of Concerns -suunnittelumallin tuomia hyötyjä ei tarvita. (Yuen, 2017)

3 WPF-KÄYTTÖLIITTYMÄKEHYS

Windows Presentation Foundation (WPF) on esityskehys modernien Windows-työpöytäsovelluksien tekemiseen. WPF-kehyksen ensimmäinen julkaisu oli osa .NET-kehyksen versio 3.0 julkaisua vuonna 2006 (Yuen 2017). WPF-kehyksen alkuperäinen päätehtävä oli helpottaa näyttävien ja tehokkaiden käyttöliittymien kehityksessä. Tämän päätehtävän se pyrkii toteuttamaan helpottamalla käyttöliittymäsuunnittelijoiden ja sovelluskehittäjien välistä yhteistyötä ja tekemällä käyttöliittymäsuunnittelijoista aktiivisia osallisia käyttöliittymien luomisessa. (Microsoft, 2010)

WPF-sovellukset tyypillisesti sisältävät niin XAML-merkintäkieltä kuin .NET-pohjaisen ohjelmointikielellä kirjoitettua koodia. .NET-pohjaisia koodikieliä ovat muun muassa C# tai Visual Basic. XAML-kielellä tyypillisesti määritetään sovelluksen ulkoasu, kun taas .NET-pohjaisella ohjelmointikielellä määritetään sen käyttäytyminen. (Microsoft, 2010)

Vaikka WPF-kehys ei sido käyttämään MVVM-arkkitehtuurimallia, mahdollistaa sen noudattaminen suuret, erityisesti työkulkuun liittyvät hyödyt. Sitä noudattavat WPF-sovellukset ovat helpommin ylläpidettävissä, testattavissa ja usein yksinkertaisempia ymmärtää. (Yuen, 2017)

MVVM-mallin noudattaminen WPF-sovelluksessa ei ole kuitenkaan täysin ongelmatonta. Microsoft ei ole tehnyt suoria linjauksia mallin käytöstä ja kolmannen osapuolen luomat ohjenuorat ovat paikoittain ristiriidassa toistensa kanssa. MVVM-mallin noudattaminen vaatii paljon täytekoodia, jotta yksinkertaiselta tuntuvat asiat kuten tiedon sidonta saadaan toteutettua. Tätä ongelmaa varten on kuitenkin luotu useita ohjelmistokehyksiä ja tekniikoita, jotka auttavat vähentämään täytekoodin määrää. (Vice & Siddiqi, 2012)

3.1 XAML-merkintäkieli

Extensible Application Markup Language (XAML) on XML-pohjainen merkintäkieli .NET-olioiden luontiin. Sillä luodut .NET-oliot asettuvat XML-kiellelle tyypillisesti puumalliin. XAML-kielen avulla voidaan vaivattomasti luoda WPF-käyttöliittymiä, mitä voidaankin pitää kielen pääroolina. Sillä voidaan määrittää WPF-sovelluksen käyttöliittymän käyttämät painikkeet, kontrollit sekä niiden rakenteen ja tyylin. Tyypillinen tapa kirjoittaa XAML-kieltä on käyttää sitä tukevia graafisia työkaluja kuten Microsoft Visual Studio -ohjelmointiympäristöä tai Microsoft Expression Blend -käyttöliittymäsuunnitteluohjelmaa, mutta sitä voidaan myös kirjoittaa millä tahansa tekstieditorilla. (MacDonald, 2012, s. 21; Sells & Griffiths, 2007, s. 683)

XAML ja WPF käsitetään usein olevan yhtä kokonaisuutta, mutta ne ovat silti tarkkaan ottaen erillisiä teknologioita. WPF ei ole ainoa teknologia, jossa XAML-kieltä voidaan hyödyntää ja WPF-sovelluksen voi rakentaa ilman XAML-kieltä. WPF on suunniteltu käyttämään XAML-kieltä

käyttöliittymiensä määritykseen, mutta on silti tärkeä ymmärtää, että ne ovat erillisiä kokonaisuuksia. (Sells & Griffiths, 2007, s. 683)

Esimerkki 1. Yksinkertainen XAML-tiedosto ja sen rakenne

```
<Window
  x:Class="ExampleApplication.MainWindow"
  xmlns="http://schemas.mi-
crosoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Main Window"
  Height="150"
  Width="300">
  <Grid>
    <Button
      Content="Click Me"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"/>
  </Grid>
</Window>
```

Esimerkin 1 mukainen tiedosto sisältää kolme elementtiä: Window-, Grid- ja Button-elementit. Siitä voidaan myös havainnoida XAML-kielen puumainen rakenne ja kuinka .NET-olioiden luonti toteutetaan sitä käyttäen. XAML-kielessä .NET-nimiavaruudet, -tyypit ja -ominaisuudet ja -tapahtumat on liitetty XML-nimiavaruuksiin, -elementteihin ja -attribuutteihin. Puumaisessa rakenteessa ulommaisin elementti on Window-luokan olio, joka pitää näkymän muut elementit ja kontrollit sisällään. (Sells & Griffiths, 2007, s. 683)

Window-elementissä on määritetty tiedoston käyttämät XML-nimiavaruudet (xmlns). XAML käyttää niitä elementtien tarkoituksien määrittämiseen. Jos nimiavaruuksia ei olisi määritetty Window-elementissä, ei WPF-kehys tietäisi, mihin .NET tyyppeihin tiedostossa käytetyt elementit viittaavat. (Sells & Griffiths, 2007, s. 684)

Visual Studio -ohjelmointiympäristö luo oletuksena jokaiselle XAML-tiedostolle taustakoodin (code-behind) erilliseen tiedostoon ja määrittelee XAML-tiedostoon x:Class-attribuutin viittaamaan kyseiseen taustakoodiin. Attribuutin alkuosa x: on XML-lyhenne ja ilmaisee mistä nimiavaruudesta kyseinen attribuutti on peräisin. Kaikissa XAML-tiedostoissa ei välttämättä ole esimerkin tapaan määritetty x:Class-attribuuttia. Kyseinen attribuutti ja sen määritys vaaditaan kuitenkin tilanteissa, joissa XAML-tiedostolle on määritetty taustakoodi. (Sells & Griffiths, 2007, s. 683-687)

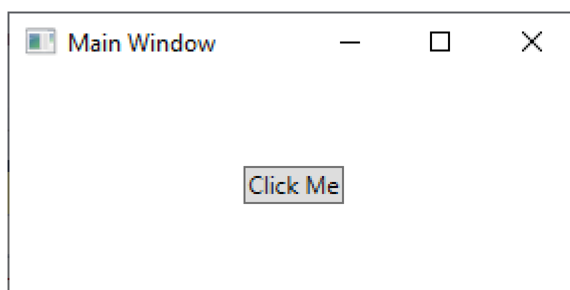
XAML-kielen viittaukset .NET olioihin on yritetty pitää niin suorina kuin mahdollista. Yleisesti ottaen XAML-elementti on jonkin .NET-luokan nimi ja sen attribuuttien nimet vastaavat kyseisen .NET-luokan ominaisuuksia tai tapahtumia. Esimerkistä 2 voidaan nähdä vaihtoehtoinen tapa luoda esimerkin 1 kaltainen sovellus C#-kieleltä käyttäen. (Sells & Griffiths, 2007, s. 9)

Esimerkki 2. C#-kielellä luotu vastine XAML-tiedoston näkymälle

```
using System;
using System.Windows;
using System.Windows.Controls;

namespace ExampleApplication {
  class MyApplication {
    [STAThread]
    static void Main(){
      Application app = new Application();
      app.Run(new MyWindow());
    }

    class MyWindow : Window {
    private Grid grid;
    private Button button;

    public MyWindow(){
      Width = 300;
      Height = 150;
      Title = "Main Window";

      grid = new Grid();

      button = new Button {
        Content = "Click Me",
        VerticalAlignment = VerticalAlignment.Center,
        HorizontalAlignment = HorizontalAlignment.Center
      };

      Content = grid;
      grid.Children.Add(button);
      }
    }
  }
}
```

MyApplication-luokan Main-metodissa luodaan Application-luokan ilmentymä, jonka Run-metodia hyödynnetään MyWindow-luokan ajamisessa. MyWindow-luokka peritytyy Window-kantaluokasta ja sen määritykset on toteutettu sen oletusrakentimen sisällä. (Sells & Griffiths, 2007, s. 6)

XML-elementin asettaminen XAML tiedostoon vastaa .NET objektin alustamista oletusrakentimella. Attribuutin määrittäminen XML-elementille taas vastaa saman nimisen ominaisuuden määrittämistä tai tapahtuman käsittelijän asettamista elementistä muodostuneelle oliolle. Molemmilla toteutustavoilla saadaan kuitenkin kuvan 1 mukainen, ulkoisesti samanlainen lopputulos. (Nathan, 2010, s. 24-25)



Kuva 1. Esimerkin mukainen WPF-applikaatio

Vaikka molemmilla toteutustavoilla saavutetaan sama lopputulos, on toteutustavoilla selkeä ero etenkin sovellusprojektin jatkojalostamiseen liittyvissä eduissa ja haitoissa. Kun XAML-kieltä käytetään käyttöliittymien määrityksessä, on ohjelmoijien huomattavasti helpompi työskennellä muiden alojen osaajien, etenkin käyttöliittymäsuunnittelijoiden kanssa. (Nathan, 2010, s. 21)

XAML mahdollistaa hyvin tiiviin tavan esittää käyttöliittymiä tai muita hierarkiallisia olioita ja se kannustaa erottamaan sovelluksen logiikan sen ulkonäöllisestä osuudesta, mikä helpottaa kehittäjiä riippumatta tiimin koosta tai heidän asiantuntemuksistaan. XAML-kielen käyttäminen mahdollistaa myös sitä tukevien suunnittelutyökalujen käytön. XAML-kielen generoimisen lisäksi Microsoft Visual Studio ja Microsoft Expression Blend tarjoavat helpon tavan tarkistaa liitettyä XAML-kieltä ja sen oikeellisuutta. (Nathan, 2010, s. 21-22)

3.2 Kontrollit

Kontrollit ovat käyttöliittymän komponentteja, jotka tuovat käyttöliittymään interaktiivisia toimintoja. Muun muassa Windows ympäristöistä tutut tekstikentät (TextBox) ja radiopainikkeet (RadioButton) ovat kontrolleja. Kaikki WPF-kehyksen elementit eivät ole kontrolleja. Elementit kuten suorakulmio (Rectangle) ja ellipsi (Ellipse) eivät omaa interaktiivisia toimintoja ja ovatkin pelkästään visuaalisia komponentteja. (Sells & Griffiths, 2007, s. 139-140)

Jokaisella kontrollilla on oletusulkoasunsa, joita pidetään olennaisena osana itse kontrollia. WPF kuitenkin tarjoaa monia keinoja muokata kontrollien ulkonäköä. Muiden muassa etu- ja taka-alan värejä sekä tekstien fontteja voidaan muokata pienillä säädöillä kontrollien ominaisuuksista. Sapluunat (template) mahdollistavat myös koko komponentin ulkonäön korvaamisen. Vaikka WPF mahdollistaa kontrollien ulkoasun laajamittaisen räätälöimisen se ei kuitenkaan anna poistaa tai muokata niille oleellisia toiminnallisuuksia. (Sells & Griffiths, 2007, s. 139)

WPF-kehyksen kontrollien toiminnallisuudet ja ominaisuudet on määritetty kontrolleihin itseensä, mutta ne tukeutuvat sapluunoihin ulkoasujensa määrityksessä. WPF-kehyksen tarjoamien laajojen

räätälöintimahdollisuuksien vuoksi ei tarvitse luoda omia mukautettuja kontrolleja, mikäli jossain valmiissa kontrollissa on tarvitut interaktiiviset toiminnot. (Sells & Griffiths, 2007, s. 139-140)

Kontrolleihin voidaan asettaa komentoja sen tukemiin toimintoihin. Esimerkiksi tekstikenttä kontrolli tukee muun muassa leikkaamis-, liittämis- ja kopioimiskomentoja. Kontrollit sisältävät ominaisuuksia niin ulkonäköön kuin toiminnallisuuksiin liittyen. Kun kontrolli havaitsee jotain merkittävää tapahtuvan, kuten käyttäjän syötteen, se herättää tapahtuman. Komennot, ominaisuudet ja tapahtumat ovat tapoja määritellä kontrollia ja sen toiminnallisuutta. (Sells & Griffiths, 2007, s. 140-141)

3.3 Looginen ja visuaalinen puu

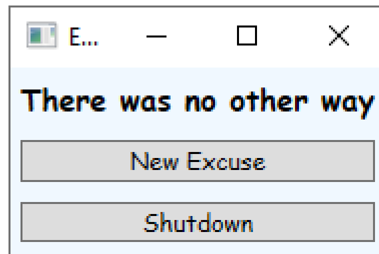
Käyttöliittymät omaavat yleensä hierarkiallisen rakenteen. WPF-kehyksen käyttöliittymä koostuu useasta .NET-oliosta, jotka asettuvat puumaiseen rakenteeseen. Tätä WPF-kehyksen muodostamaa puumaista rakennetta kutsutaan nimillä looginen ja visuaalinen puu. Käsitteet eivät kuitenkaan tarkoita samaa asiaa. Visuaalinen puu sisältää kaikki käyttöliittymän sisältämät elementit, kun taas looginen puu on sen abstraktio. (Nathan, 2010, s. 75-77)

Esimerkissä 3 on yksinkertainen XAML-merkintäkielinen WPF-näkymä. XAML-tiedostoa tutkiessa voidaan havaita, että näkymän juurielementti on Window, jonka ainoa lapsielementti on StackPanel, joka taas sisältää omat lapsielementtinsä. Näkymän Window-elementti on sen loogisen sekä visuaalisen puun juuri, josta muut elementit haarautuvat. (Nathan, 2010, s. 75-77)

Esimerkki 3. XAML yksinkertaisesta näkymästä

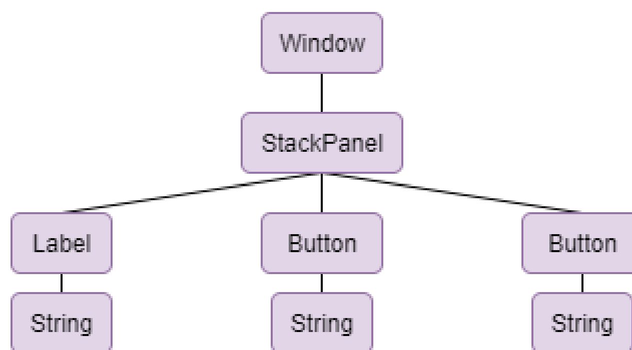
```
<Window
  xmlns="http://schemas.mi-
crosoft.com/winfx/2006/xaml/presentation"
  Title="Excuse generator"
  SizeToContent="WidthAndHeight"
  FontFamily="Comic Sans MS"
  Background="AliceBlue">
  <StackPanel>
    <Label FontWeight="Bold"
      FontSize="15">
      There was no other way
    </Label>
    <Button Margin="5">New Excuse</Button>
    <Button Margin="5">Shutdown</Button>
  </StackPanel>
</Window>
```

Esimerkin 3 XAML-tiedostosta muodostuu kuvan 2 kaltainen näkymä. StackPanelin Label- ja Button-lapsielementit asettuvat allekkain näkymässä.



Kuva 2. Esimerkin 3 pohjalta käännetty näkymä

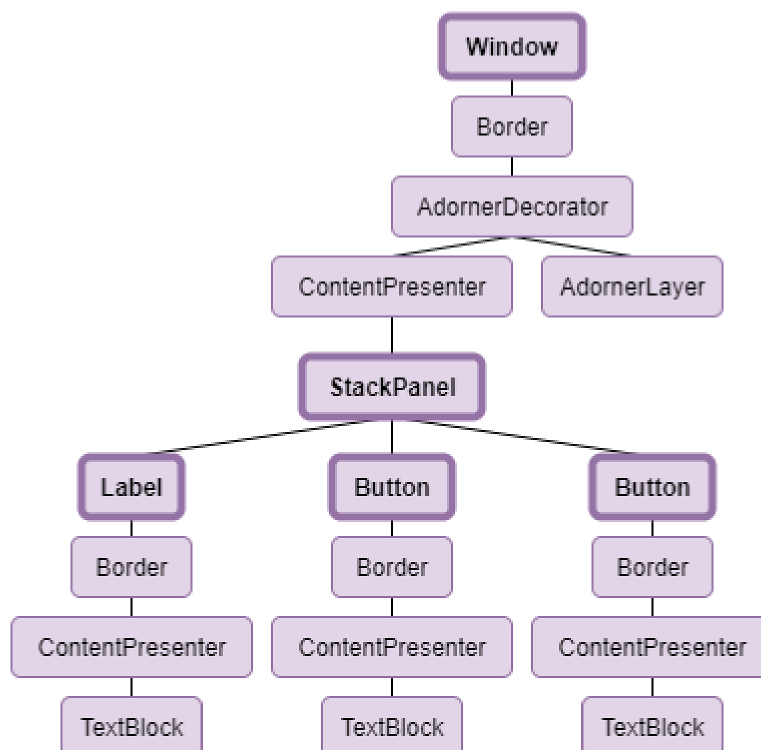
Kuvio 2 esittää millainen looginen puu esimerkin 3 XAML-tiedostosta muodostuu. Loogisen puun juuri on Window, josta muut elementit haarautuvat. Looginen puu sisältää XAML-tiedostossa näkyvät elementit ja niiden sisällöt, tässä tapauksessa String-muotoiset merkkijonot. (Nathan, 2010, s. 75-80)



Kuvio 2. Esimerkin 3 looginen puu (Perustuu: Nathan 2010, s. 76)

Loogisen puun merkitys on hyvin suuri WPF-kehyksessä. WPF-kehyksen toiminnallisuudet ovat usein tiiviisti liitetty loogiseen puuhun. Ominaisuuden arvoja levitetään usein loogista puuta alaspäin, ja joidenkin elementtien havaitsemat tapahtumat saatetaan reitittää puuta ylös- tai alaspäin. Esimerkissä 3 Window-elementille on määritetty Comic Sans MS -fontti, jonka kaikki sen lapsielementit ovat perineet, mikä voidaan nähdä kuvasta 2. (Nathan, 2010, s. 75-80)

Looginen puu sisältää kuitenkin vain osajoukon näkymän todellisista elementeistä. Puuta, joka sisältää näkymän kaikki todelliset elementit kutsutaan visuaaliseksi puuksi. Kuvio 3 osoittaa, kuinka lähes jokainen elementti kätkee sisäänsä useita elementtejä, joita loogisessa puussa ei näytetä. Esimerkiksi Button-kontrolli sisältää loogiselta puulta kätketyt Border-, ContentPresenter- ja TextBlock-elementit. (Nathan, 2010, s. 75-80)



Kuvio 3. Visuaalinen puu, jossa loogisen puun elementit ovat korostettu (Perustuu: Nathan, 2010, s. 78)

Usein yksinkertaiselta vaikuttava elementti kätkee sisälleen useita elementtejä luoden monimutkaisen vaikutelman. Visuaalisesta puusta on hyvä olla tietoinen, mutta siitä ei onneksi tarvitse huolehtia, ellei ole tarvetta muokata kontrollien ulkoasua radikaalisti. Tarpeen tullessa visuaalisen puun rakenteeseen voidaan vaikuttaa korvaamalla kontrollien alkuperäiset sapluunat omatekoisilla sapluunoilla. (Nathan, 2010, s. 75-80)

3.4 Reititetyt tapahtumat

Tapahtumat ovat olion — kuten WPF-kontrollin — lähettämiä viestejä, jotka ilmoittavat, kun jotain merkittävää esiintyy (MacDonald 2012, s. 105). WPF-kehyksen kontrolleilla voi olla ja usein onkin useita tapahtumia. Esimerkiksi Button-kontrolli omaa muun muassa Click-, TouchDown-, ja TouchUp-tapahtumat. Tapahtumalla voi olla useita tilaajia, jotka määrittelevät tapahtumakäsittelijän kyseiselle tapahtumalle. (Sells & Griffiths, 2007, s. 109-110)

Reititetyt tapahtumat (routed events) ovat tapahtumia, jotka on suunniteltu toimimaan visuaalisen puun kanssa. Kun tapahtuma esiintyy, WPF kulkee visuaalisen puun lävitse kutsuen kaikkia tapahtuman tilanneita tapahtumakäsittelijöitä, jotka se matkallaan kohtaa. Tämä käytös on olennainen osa WPF-kehyksen tapahtumakäsittelyä. (Sells & Griffiths, 2007, s. 109-110)

Visuaalista puuta voidaan kulkea kahteen eri suuntaan. WPF-kehyksen reititetyille tapahtumille on olemassa kolme eri tavalla visuaalisessa puussa kulkevaa reititysstrategiaa:

– Laskevat tapahtumat (tunneling events) kulkevat puuta alaspäin. Tapahtuma käsitellään ensin juurielementissä, minkä jälkeen tapahtuman käsittely siirtyy hierarkkisen rakenteen alemmille elementeille, kunnes saavutetaan tapahtuman lähde-elementti. Laskevat tapahtumat mahdollistavat tapahtuman käsittelyjen tarkistamisen ja keskeytyksen ennen lähde-elementin saavuttamista. PreviewMouseDown on laskeutuva tapahtuma, joka saa alkunsa, kun hiiren kursoria painetaan elementissä.
– Nousevat tapahtuvat (bubbling events) nousevat nimensä mukaisesti puuta sen hierarkiassa ylöspäin. MouseDown on esimerkki nousevasta tapahtumasta. Se saa alkunsa samana tapaan kuin PreviewMouseDown. Kun MouseDown-tapahtuma havaitaan, käsitellään se ensiksi elementissä, josta se on lähtöisin. Tapahtuma siirretään sen jälkeen käsiteltäväksi hierarkiassa yläpuolella olevalle elementille. Tätä jatkuu, kunnes saavutetaan, kunnes saavutetaan elementtihierarkian huippu — juurielementti.
– Suorat tapahtumat (direct events) käsitellään vain elementissä, josta se on lähtöisin. Niitä ei välitä muille elementeille. Suoriin tapahtumiin lukeutuu muun muassa MouseEnter-tapahtuma on, joka esiintyy, kun hiiren kursori siirtyy elementin päälle. (Nathan, 2010, s. 161-162; MacDonald, 2012, s. 109-110)

Esimerkki 4. StackPanel-elementti, jonka sisällä on Image- ja Label-elementit

```
<StackPanel
  MouseDown="StackPanel_MouseDown"
  PreviewMouseDown="StackPanel_PreviewMouseDown">
  <Image
    MouseDown="Image_MouseDown"
    PreviewMouseDown="Image_PreviewMouseDown"/>
  <Label
    MouseDown="Label_MouseDown"
    PreviewMouseDown="Label_PreviewMouseDown"/>
</StackPanel>
```

Esimerkissä 4 on kuvattu XAML-merkintäkielellä käyttöliittymärakenne, jossa StackPanel-elementin sisään on asetettu Image- ja Label-elementit. Jokaiseen esimerkin elementtiin on lisätty viittaus PreviewMouseDown- ja MouseDown-tapahtumien käsittelijöihin. Molemmat tapahtumat saavat alkunsa hiiren painalluksella, mutta niiden reititysstrategia on eri — PreviewMouseDown on laskeva, kun taas MouseDown on nouseva tapahtuma. Jos Label-elementtiä painetaan, on tapahtumien käsittelyjärjestys seuraava:

– StackPanel_PreviewMouseDown
– Label_PreviewMouseDown
– Label_MouseDown

– StackPanel_MouseDown
(Sells & Griffiths, 2007, s. 109-113)

Järjestyksestä voidaan havaita, että laskevat, Preview-etuliitteen omaavat laskevat tapahtumat, käsitellään ennen nousevia tapahtumia. Järjestyksestä voidaan myös havaita, kuinka laskevat tapahtumat käsitellään hierarkkisesti laskevasti alkaen StackPanel-elementistä, ja kuinka nousevat tapahtumat käsitellään lähde-elementistä nousevasti. Kaikkia rakenteessa määritettyjä tapahtumakäsittelijöitä ei kuitenkaan kutsuta. Image-elementin tapahtumat eivät laukea, sillä elementti ei ole hierarkkisessa rakenteessa Label-elementin yläpuolella. (Sells & Griffiths, 2007, s. 113)

3.5 Tiedon sidonta

Tiedon näyttäminen ja säilyttäminen on yksi tärkeimmistä tehtävistä Windows-työpöytäsovelluksissa. Suurin osa sovelluksista näyttää tietoa käyttäjälle ja usein antaa niiden myös editoida sitä. Tiedon sidonta (data binding) tarkoittaa kahden ominaisuuden välille luotua kommunikaatiokanavaa, joka mahdollistaa ohjelmakoodissa olevan tiedon noutamisen käyttöliittymän elementille. Kaksisuuntainen tiedon sidonta mahdollistaa myös käyttöliittymän elementin ominaisuuteen sidotun tiedon päivittämisen. (MacDonald, 2012, s. 557)

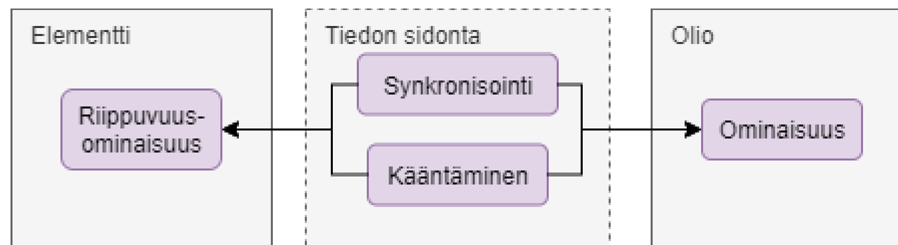
Tiedon sidonta voidaan tehdä myös kahden käyttöliittymäelementin välille. Esimerkissä 5 esitetään yksinkertainen kahden elementin välinen tiedon sidonta, jossa CheckBox-kontrollin IsChecked-ominaisuus on sidottu Button-kontrollin IsEnabled-ominaisuuteen. CheckBox-kontrollin IsChecked-ominaisuuden arvo voi olla joko tosi (true) tai epätosi (false), ja sen muuttuessa muuttuu myös Button-kontrollin IsEnabled-ominaisuuden arvo. Sidoksessa määritetty ElementName-ominaisuus kertoo, minkä nimisestä elementistä arvoa haetaan, ja Path-ominaisuus taas sen, mikä ominaisuus sidotaan. (MacDonald, 2012, s. 227-229)

Esimerkki 5. Yksinkertainen, kahden elementin välinen tiedon sidonta

```
<CheckBox Margin="10" Name="checkBox"/>
<Button Margin="10"
  Content="Simple Button"
  IsEnabled="{Binding ElementName=checkBox,
  Path=IsChecked}" />
```

Suurin osa tiedon sidonnasta tehdään käyttöliittymän elementtien ja tietoa sisältäviin olioiden välille, mutta elementtien välisestä tiedon sidonnasta on usein silti hyötyä. Elementtien välisellä tiedon sidonnalla voidaan automatisoida elementtien toiminnallisuutta, mikä voi säästää ylimääräisen koodin kirjoittamiselta. (MacDonald, 2012, s. 227-229)

WPF-kehyksen tiedon sidonnan hoitaa todellisuudessa System.Windows.Data.Binding-sidontaolio, jolle ominaisuudet rekisteröidään. Sidontaolio synkronisoi kaksi ominaisuutta ja tarpeen vaatiessa kääntäen tietomuodon yhteensopivaksi, kuten kuviosta 4 voidaan havainnoida. (Nathan, 2010, s. 363; Sells & Griffiths, 2007, s. 177)



Kuvio 4. Tiedon sidonnan vastuualueet (Perustuu Sells & Griffiths, 2007, s. 177)

Useimmiten tiedon sidonta on toteutettu käyttöliittymän elementin ja ei visuaalisen olion välille. ElementName-ominaisuutta ei voida käyttää viittauksessa, jos ei viitata elementtiin, vaan se pitää korvata jollakin seuraavista:

- Lähde (Source) viittaa suoraan lähdeolioon, eli olioon, joka tarjoaa halutun tiedon.
- Suhteellinen lähde (RelativeSource) on yksi tapa viitata lähdeolioon. Sen avulla viittaus lähdeolioon luodaan sen perusteella, missä suhteessa se on kohdeolioon. Suhteellista lähdeviittausta käytetään tiedon sidonnassa esimerkiksi tilanteessa, jossa tieto halutaan sitoa listan aiempaan soluun nykyisen solun sijaan tai tilanteessa, jossa jonkin elementin tieto halutaan sitoa sitä hierarkkisessa rakenteessa yläpuolella olevaan reunukseen (Border). Suhteellista lähdeviittausta käytetään usein kontrollien ja tietosapluunoiden kanssa.
- Tietokontekstia (DataContext) käytetään tilanteissa, joissa tarvitaan yhdistää useita ominaisuuksia saman lähdeolion ominaisuuksiin. Jos tiedon sidonnassa ei ole käytetty lähdettä tai suhteellista lähdettä, WPF lähtee etsimään tietokontekstia kulkien elementtipuuta ylöspäin. Jokainen elementti tarkistetaan tietokontekstilta, jonka löydyttyä, asetetaan se tiedon sidonnan tietokontekstiksi. Esimerkissä 6 on havainnollistettu, kuinka tiedon sidonnassa voidaan hyödyntää tietokontekstia. Jos tietokontekstia ei olisi määritetty, jokaiselle tiedon sidonnalle pitäisi luoda suora lähde viittaus staattiseen Brushes.Green-luokkaan. (MacDonald, 2012, s. 238-239)

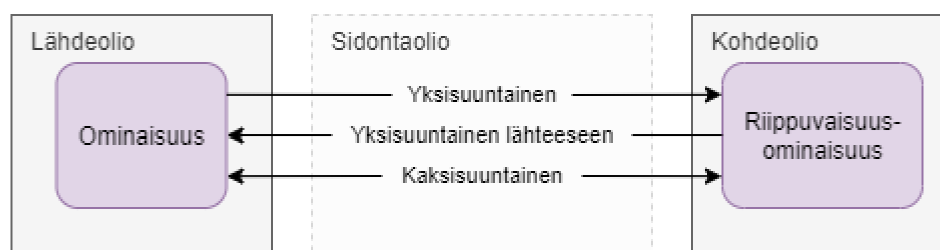
Esimerkki 6. Tietokontekstin käyttö

```
<StackPanel DataContext="{x:Static Brushes.Green}">
  <TextBlock Text="{Binding CanFreeze}"/>
  <TextBlock Text="{Binding Transform.IsFrozen}"/>
  <TextBlock Text="{Binding Color}"/>
  <TextBlock Text="{Binding Opacity}"/>
</StackPanel>
```

Tiedon sidontasuuntaa voidaan muokata määrittämällä sidonnalle tyyppi (Mode). Kuvio 5 osoittaa, kuinka tyypin valinta vaikuttaa siihen, miten ominaisuudet päivittyvät. Tiedon sidontaan voidaan valita jokin seuraavista tyypeistä:

- Yksisuuntainen (OneWay) — Kohdeominaisuus päivitetään, kun lähdeominaisuus muuttuu.
- Kaksisuuntainen (TwoWay) — Kohdeominaisuus päivitetään, kun lähdeominaisuus muuttuu ja lähdeominaisuus päivitetään, kun kohdeominaisuus muuttuu.
- Kertaluontoinen (OneTime) — Kohdeominaisuuteen asetetaan lähdeominaisuuden arvo sidonnan luontivaiheessa. Tämän jälkeiset muutokset lähdeominaisuuteen eivät päivitä kohdeominaisuutta. Tätä tyyppiä käytetään, jos tiedetään, ettei lähdeominaisuus muutu.
- Yksisuuntainen lähteeseen (OneWayToSource) — Yksisuuntainen lähteeseen on käänteinen yksisuuntaiselle tyypille, eli lähdeominaisuus päivitetään, kun kohdeominaisuus muuttuu.
- Oletus (Default) — Jos sidonnalle ei määritetä tyyppiä se käyttää kohdeominaisuudelle määritettyä oletusta. Oletustyyppiä ei useimmiten valita erikseen, mutta on tärkeä tiedostaa, että jokaisella oliolla on sellainen. Jos tyypiksi on valittu oletus, tiedon sidonnan tyyppi määräytyy kohdeominaisuuden mukaan. Ominaisuuden oletustyyppi on useimmiten yksisuuntainen, mutta käyttäjän muokattavissa oleville kohdeominaisuuksille se on kaksisuuntainen. (MacDonald, 2012, s. 230-232)

Esimerkin 6 sidonnat käyttävät TextBlock-kohde-elementin Text-ominaisuuden oletustyyppiä, joka on yksisuuntainen. Kohde-elementin Text-ominaisuuteen päivitetään Brushes.Green-lähteen tietoa, eikä toisin päin. (MacDonald, 2012, s. 230-232)



Kuvio 5. Tiedon sidontasuunnat (Perustuu MacDonald, 2012, s. 231)

Tiedon sidontaluokalla on monia muitakin ominaisuuksien sidontaan vaikuttavia ominaisuuksia. Yksi oleellinen ominaisuus, etenkin sidoksissa, jotka ovat kaksisuuntaisissa tai yksisuuntaisia lähteeseen on sidonnan synkronisointiajankohtaan vaikuttava UpdateSourceTrigger. UpdateSourceTrigger-ominaisuuden oletusarvo TextBox-elementin Text-ominaisuudessa on PropertyChanged, eli käytännössä lähdeominaisuus päivitetään jokaisella näppäinpainalluksella. Tätä toiminnallisuutta voidaan muokata

vaihtamalla UpdateSourceTrigger-ominaisuuden arvoa. (Nathan, 2010, s. 404-405)

UpdateSourceTrigger-ominaisuudella on kolme eri arvoa: PropertyChanged, LostFocus ja Explicit. PropertyChanged päivittää lähdeominaisuuden välittömästi kohdeominaisuuden muuttuessa, LostFocus päivittäessä sen vasta, kun kohde-elementti menettää fokuksen. Jos lähdeominaisuuden päivittämisajankohta halutaan tarkemmin hallita, käytetään Explicit-tilaa, jossa lähdeominaisuus päivitetään vasta, kun sitä erillisen päivityskutsun yhteydessä. Tämä päivityskutsu voidaan esimerkiksi yhdistää Button-elementin painallukseen. (Nathan, 2010, s. 404-405)

3.5.1 Tiedon muuntaminen

Tietoa joudutaan muuntamaan, jos lähdeominaisuuden ja kohdeominaisuuden tietotyypit poikkeavat toisistaan tai jos tietoa halutaan muotoilla eri tavalla. Esimerkissä 7 TextBlock-elementin taustaväri on sidottu elementin sisältämään tekstiin suhteellisella lähdeviittauksella. Background-ominaisuus on Brush-tyyppinen, mutta tiedon sidonta tekstimuotoiseen ominaisuuteen onnistuu silti. Tämä johtuu siitä, että Brush-luokassa on kääntämiseen tarvittu tyyppimuunnin. (Microsoft, 2017 b)

Esimerkki 7. TextBlock-elementin taustaväri on sidottu sen tekstiin

```
<TextBlock
  Text="Yellow"
  Background="{Binding RelativeSource={
  x:Static RelativeSource.Self}, Path=Text}"/>
```

Tyyppimuuntimet toimivat tyyppeihin pohjautuen, eli esimerkiksi kokonaislukuja voidaan muuttaa merkkijonoiksi ja takaisin. Kaikkeen muuntamistarkoitukseen tyyppimuuntimet eivät kuitenkaan riitä. Joskus voi olla tarve muuntaa tietoa sovelluskohtaisella tavalla. Tällöin käytetään arvomuuntimia, jotka räätälöidään käyttötapausta varten. Esimerkissä 8 on esitetty arvomuunnin, joka muuntaa sille annetut Celsius-arvot small talkiksi. Muuntimen sovelluslogiikka määrittelee sille annetun kokonaisluvun mukaan, minkä merkkijonon se palauttaa. Esimerkin kääntäjä toimii vain yhteen suuntaan, sillä ConvertBack-metodia ei ole toteutettu. (Sells & Griffiths, 2007, s. 188)

Esimerkki 8. Arvomuuntaja Celsius-arvosta small talkiksi

```
[ValueConversion(typeof(Int32), typeof(String))]
public class CelciusToSmallTalkConverter : IValueConverter
{
  public object Convert(object value, Type targetType, ob-
ject parameter, CultureInfo culture)
  {
```

```
    Int32.TryParse((String)value,   out   Int32   celciusDe-
grees);
    if (celciusDegrees > 25)
    {
      return "It's hot out there";
    } else if (celciusDegrees > 0)
    {
      return "The temperature is tolerable";
    } else
    {
      return "It's freezing out there";
    }
  }

  public  object  ConvertBack(object  value,  Type  targetType,
object parameter, CultureInfo culture)
  {
    return Binding.DoNothing;
  }
}
```

Esimerkissä 9 on hyödynnetty esimerkin 8 arvomuuntajaa. Jotta arvomuunnin voidaan määrittää tiedon sidonnalle parametrina, joudutaan siitä ensin luoda resurssi. TextBlock-elementeistä jälkimmäinen sisältää numeerisen Celsius-arvon. Toinen elementeistä tekee tiedon sidonnan Celsius-arvon sisältämän elementin Text-ominaisuuteen ja määrittää sidonnassa käytettävän muuntimen. Muunnin palauttaa tässä tapauksessa arvoksi arvoksi ” It's hot out there” -tekstin. (Microsoft, 2017 b)

Esimerkki 9. Esimerkin 8 arvomuuntimen hyödyntäminen tietoa sidottaessa

```
<StackPanel>
  <StackPanel.Resources>
    <local:CelciusToSpeechConverter
      x:Key="CelciusToSpeechConverter"/>
  </StackPanel.Resources>
  <TextBlock
    Text="{Binding ElementName=celciusAmount,
    Converter={StaticResource CelciusToSpeechConverter},
    Path=Text}"/>
  <TextBlock
    Name="celciusAmount">
    30
  </TextBlock>
</StackPanel>
```

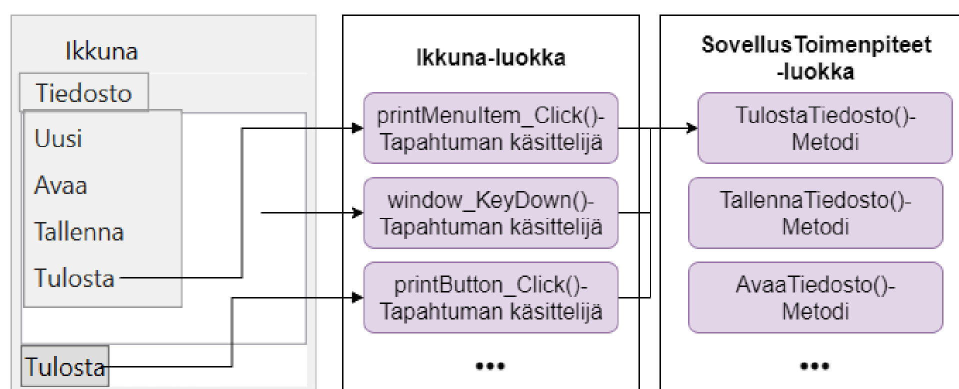
WPF tarjoaa muutamia yleiskäyttöön suunniteltuja arvomuuntimia. BooleanToVisibilityConverter on yksi näistä muuntajista ja se muuntaa totuusarvosta (boolean) elementtien näkyvyysarvoon ja takaisin. Näkyvyys (Visibility) on elementin ominaisuus, jolla on kolme eri arvovaihtoehtoa: näkyvä (Visible), piilotettu (Hidden) ja romautettu (Collapsed). BooleanToVisibilityConverter kääntää arvoja molempiin suuntiin. Näkyvyysarvot kääntyvät totuusarvoiksi seuraavasti: näkyvä tulee tosi ja epätodesta ja tyhjästä

tulee romautettu. Toiseen suuntaan tosi muuttuu näkyväksi ja epätosi ja tyhjä muuttuvat romautetuksi. (Nathan, 2010, s. 383)

3.6 Komennot

Komennot, kuten tapahtumat, ovat tapa käsitellä käyttäjän syötettä. Komennot erottelevat sen herättäneen olion siihen liitetystä toimintalogiikasta, mikä mahdollistaa logiikan suorittamisen useasta eri lähteestä. Komentojen lähde voi olla muun muassa näppäinyhdistelmä, valikkopainike, työkalupalkin painike tai muu käyttöliittymään asetettu elementti. (Sells & Griffiths, 2007, s. 124; Microsoft, 2017 c)

Hyvin suunnitelluissa WPF-sovelluksissa sovelluslogiikkaa ei ole kirjoitettu suoraan tapahtuman käsittelijöihin, vaan niitä korkeatasoisemmille metodeille, joista jokainen edustaa yhtä sovelluksen toimenpidettä. Kun sovelluslogiikkaa ei ole kirjoitettu tapahtuman käsittelijään, on sen uudelleen käyttö helpompaa. Kuvio 6 havainnollistaa tämän yhteyden. (MacDonald, 2012, s. 243)

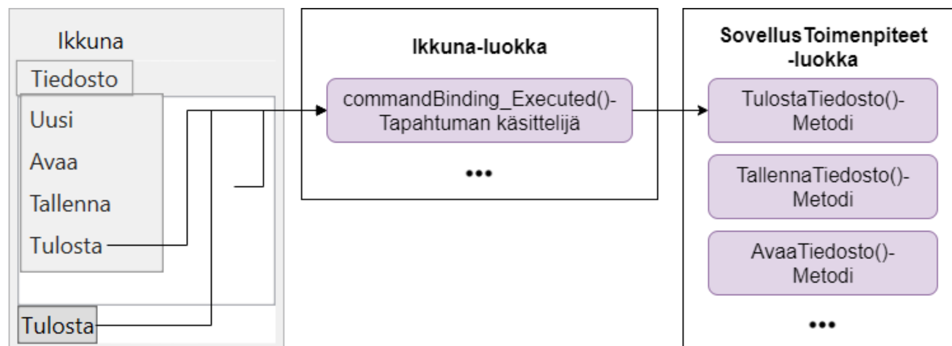


Kuvio 6. Tapahtuman käsittelijöiden yhdistäminen toimenpiteeseen (Perustuu MacDonald, 2012, s. 243)

Kuviossa tiedoston tulostaminen voidaan toteuttaa kolmella eri tavalla: valikkopainikkeesta, näppäinyhdistelmällä tai erillisestä painikkeesta. Jokaiselle eri tavalle on luotu oma tapahtuman käsittelijä, joista jokainen kutsuu samaa toimenpidettä. Toteutustapa on melko järkevä, mutta sisältää silti ongelmansa. Erityisesti käyttöliittymän tilan (state) käsittely muodostaa ongelmia, jos toimenpiteet on toteutettu erillisillä tapahtuman käsittelijöillä. Esimerkiksi tulostamistoiminnallisuuden estäminen hetkellisesti on työlästä. Kun toimenpiteen käyttö halutaan estää, usein siihen yhdistetyt elementit myös pois käytöstä. Jokainen tulostustoimenpidettä käyttävä kontrolli ja näppäinkomento joudutaankin poistamaan käytöstä erikseen. Myöhemmin sama toiminto halutaan kuitenkin takaisin päälle. Tämän kaltaisen toiminnallisuuden kirjoittaminen ohjelmakoodiin on sotkuista. (MacDonald, 2012, s. 243-244)

Kuvio 7 esittää, kuinka kuvion 6 kaltainen sovellus voidaan toteuttaa komentopohjaisesti. Käyttäjällä on edelleen samat tavat käynnistää

tulostamistoiminnallisuus, mutta ne on yhdistetty nyt samaan komentoon. Komennon sidontaa hyödyntämällä komento yhdistetään Ikkuna-luokan tapahtuman käsittelijään. (MacDonald, 2012, s. 244-245)



Kuvio 7. Tapahtumien yhdistäminen komentoon (Perustuu MacDonald, 2012, s. 244)

WPF sisältää monia sisäänrakennettuja komentoja, jotka havainnollistavat hyvin komentojen käyttämisen etuja. Sisäänrakennetut komennot mahdollistavat yleisimpien toiminnallisuuksien toteuttamisen. Muiden muassa monissa sovelluksissa olevat kopioimis-, leikkaamis- ja liittämistoiminnot löytyvät WPF-kehyksen sisäänrakennetuista komennoista. Esimerkki 10 esittää niiden käytön yksinkertaisuuden. (Microsoft, 2017 c)

Esimerkki 10. Komentojen käyttö Menu-elementissä

```
<StackPanel>
  <Menu>
    <MenuItem
      Header="Kopioi"
      Command="ApplicationCommands.Copy"/>
    <MenuItem
      Header="Leikkaa"
      Command="ApplicationCommands.Cut"/>
      <MenuItem
      Header="Liitä"
      Command="ApplicationCommands.Paste"/>
  </Menu>
  <TextBox Width="150"/>
</StackPanel>
```

Esimerkissä on valikko (Menu) ja sen sisäiset valikkopainikkeet (MenuItem), joille on määritetty niiden laukaisemat komennot. Kuvio sisältää myös TextBox-elementin, johon valikkopainikkeita voidaan hyödyntää. TextBox-elementtiin on ennalta määritetty, kuinka eri komennot toteutetaan. Vastaavasti valikkopainike-elementit sisältävät natiivin tuen komentojen laukaisemiseen. Komennolle voidaan määrittää sen suorittamisehdot. Esimerkiksi kopioimista ei voida toteuttaa, jos minkäänlaista valintaa ei ole tehty, eikä liittämistä voi tehdä, mikäli leikepöytä on tyhjä. Painikkeet, joiden komentojen suorittamisehdot eivät ole täyttyneet, otetaan pois käytöstä. (Microsoft, 2017 c)

3.7 Tyylit

Tyylien avulla elementtien muotoilumäärityksiä voidaan organisoida ja uudelleen käyttää. Yhdenmukaisien käyttöliittymäelementtien teko on haastavaa, jos jokaisen elementin ulkoasu joudutaan määrittää erikseen. Tyylien avulla ulkoasuissa toistuvista yksityiskohdista kuten marginaaleista, fonteista ja väreistä voidaan muodostaa uudelleenkäytettäviä tyylijoukkoja visuaalisia elementtejä varten. Elementti voi ottaa tällaisen tyylijoukon käyttöön vain yhden ominaisuuden määrittämisellä. (MacDonald, 2012, s. 283)

Esimerkki 11. TextBlock-elementeille määritetyt tyylit

```
<StackPanel>
  <StackPanel.Resources>
    <Style TargetType="TextBlock">
      <Setter Property="Foreground" Value="Green" />
    </Style>
    <Style
      BasedOn="{StaticResource {x:Type TextBlock}}"
      TargetType="TextBlock"
      x:Key="TitleText">
      <Setter Property="FontSize" Value="26"/>
    </Style>
  </StackPanel.Resources>
  <TextBlock
    Style="{StaticResource TitleText}">The Title</TextBlock>
  <TextBlock>A block of text</TextBlock>
  <TextBlock Foreground="Blue">A blue one</TextBlock>
</StackPanel>
```

Esimerkissä 11 on kolme erillistä TextBlock-elementtiä, joiden ulkoasu poikkeaa toinen toisestaan. StackPanel-elementin sisälle on määritetty resursseja (Resources), joita sen sisäiset elementit voivat hyödyntää. Ylempänä määritetty tyyli määrittää TextBlock-elementtien etualalle vihreän värin. Jokainen StackPanel-elementin sisäinen TextBlock perii tyylin mukaisen vihreän värin. Tämä väri voidaan kuitenkin yli kirjoittaa kuten alimmassa TextBlock-elementissä on tehty. Alempi tyyli on niin ikään TextBlock-elementin tyyli. Se ei kuitenkaan aktivoidu jokaiselle StackPanel-elementin TextBlock-elementille, vaan vaatii erillisen tyyliviittauksen. Tyyliviittaustapa voidaan nähdä StackPanel-elementin ylimmästä TextBlock-elementistä. (Microsoft, 2019)

Tyyleihin voi yhdistää laukaisimia (Trigger), joiden avulla tyylin käyttöä voidaan automatisoida. Laukaisimiin voidaan asettaa ehtoja, jotka täytyy täyttyä, jotta tyyli otetaan käyttöön. Tyylille voidaan asettaa useita laukaisimia, joista jokaiselle voidaan asettaa useita laukaisuehtoja. (MacDonald, 2012, s. 294)

Esimerkki 12. Button-elementeille määritetty yksinkertainen ominaisuuslaukaisin.

```
<Style TargetType="Button">
  <Style.Triggers>
    <Trigger Property="Button.IsMouseOver" Value="True">
      <Setter Property="Foreground" Value="Red" />
    </Trigger>
  </Style.Triggers>
</Style>
```

Esimerkissä 12 on tyyli, jolle on määritetty sen kohdetyyppi (TargetType), laukaisin, sen ehdot sekä sille määritetyt Setter-ominaisuudet. Tyyli on kohdistettu Button-elementeille ja siihen on asetettu laukaisin, joka laukaistaan, kun elementin IsMouseOver-ominaisuus muuttuu todeksi, eli kun kursori on elementin yllä. Laukaisimelle määritetyssä Setter-ominaisuudessa on kerrottu Button-elementin tyyli, kun laukaisimen ehdot ovat täyttyneet. Käytännössä esimerkin mukainen tyyli asettaa Button-elementin etualan eli tekstin väriksi punaisen, kun kursori on elementin yllä. (MacDonald, 2012, s. 294-297)

4 TYÖN TAVOITE

Opinnäytetyön käytännönosuuden tavoitteena on luoda tulostustyökalu, joka korvaa aiemman Java-sovelman. Korvattava sovelma on ominaisuuksiltaan ja käyttöliittymältään riittävä sovellus sen käyttötarkoitukseen. Sovelman käyttämän, monesta selaimesta tuen menettäneen NPAPI-rajapinnan käytön takia se joudutaan kuitenkin korvaamaan. Tehtävän Windows-työpöytäsovelluksen on tarkoitus mahdollistaa taulukon 1 kaltainen käyttötapaus.

Taulukko 1. Tulostyökalun käyttötapaus

Kuvaus	Tämä skenaario kuvaa tiedostojen massatulostustyökalun käytön. Skenaario kuvaa onnistuneen päätapauksen.
1	Käyttäjä käynnistää sovelluksen Windows-työpöydältä
2	Käyttäjä kirjautuu asianhallintajärjestelmän tunnuksilla
3	Käyttäjä hakee tulostettavat asiakirjat asianhallintajärjestämästä
4	Käyttäjä tulostaa valitut asiakirjat työaseman tulostimelle
5	Käyttäjä kirjautuu ulos ja/tai sammuttaa sovelluksen

Sovelluksen tulisi mahdollistaa tapaus, jossa käyttäjä kirjautuu asianhallintajärjestelmän tunnuksilla, hakee ja tulostaa halutut asiakirjat sekä kirjautuu ulos. Aiemmin luotua sovelmaa voidaan pitää vertauskohteena, jonka toiminnot työn on tarkoitus ensisijaisesti toteuttaa.

Työkalu toteutetaan WPF-käyttöliittymäkehystä käyttäen. Lisäksi sovellus suunnitellaan ja kehitetään MVVM-arkkitehtuurimallia noudattaen. Parhaimmillaan työkalu tarjoaisi käyttäjilleen aiempaa ratkaisua muistuttavat tulostustoiminnallisuudet, vain lisäten toiminnallisuuksia sen päälle ja parantaen sen toiminnallisuuksia entisestään. Voidaankin pitää hyveenä, jos työkalu muistuttaa käytöltään korvattavaa sovelmaa, sillä se saattaa vähentää kouluttamiseen ja dokumentointiin kuluvaa aikaa. Aiemman ratkaisun käyttöliittymä on ollut riittävä käyttötarkoitukseensa, joten poikkeamisia siihen tehdään vain, tavoiteltaessa parempaa käytettävyyttä.

Liite 1 esittää, kuinka taulukon 1 mukainen käyttötapaus on mahdollistettu korvattavassa sovelmassa. Käyttäjä kirjautuu asianhallintajärjestelmään ja hakee tiedostot käyttäen asianhallintajärjestelmän omaa asiakirjahakua. Hakunäkymässä on massatulostuspainike, jolla tulostustyökalun päänäkymä avautuu.

Päänäkymän yläreunassa on lyhyesti kerrottu ohjeet sovelluksen käyttöön. Näkymästä suurimman alan vie asiakirjojen valintakenttä, josta asiakirjoja voidaan valita tulostukseen yksitellen. Valintakentän ulkopuolella on myös painikkeet valintojen poistamiseen ja kaikkien valitsemiseen. Työasemaan

yhdistetty tulostin, jolle asiakirjat tulostetaan, näkyy näkymän alareunassa. Sovelma ei salli tulostimen vaihtamista, vaan se tulostaa aina työaseman oletustulostimelle.

Työkalussa on lisäksi valintakenttä, jolla asiakirjojen noutamista asianhallintajärjestelmästä voidaan simuloida ilman, että asiakirjoja lähetetään tulostimelle tulostettavaksi. Toiminnallisuutta voidaan hyödyntää tilanteessa, missä joudutaan selvittämään tulostukseen liittyviä ongelmia.

Kun asiakirjat on valittu ja tulostuspainiketta painetaan, avaa työkalu uuden näkymän, jossa tulostustyön etenemistä ilmaistaan käyttäjälle. Työkalu hakee asiakirjat yksitellen asianhallintajärjestelmän SOAP-protokollaa noudattavasta www-sovelluspalvelusta. Kun asiakirja on noudettu onnistuneesti, käynnistää työkalu sen tiedostopäätteeseen määritetyllä oletussovelluksella. Tämä voisi esimerkiksi olla doc- tai docx-päätteisille tiedostoille Microsoft Word. Sovellusta ei käynnistetä täysin normaalilla tavalla, sillä käynnistysvaiheessa sovellus annetaan lisätietona, että kyseessä on tulostustyö. Eli normaalitilanteessa sovellus käynnistyy vain hetkellisesti käynnistääkseen tulostustyön. Mikäli tiedoston haku ei jostain syystä onnistu, käyttäjälle annetaan mahdollisuus keskeyttää työkalun toiminta.

Asiakirjojen käsittelyn päätteeksi näytetään käyttäjälle yhteenveto. Tiedoston haussa ilmenneet ongelmat kirjataan päänäkymään ilmestyneeseen virhelokikenttään.

Mainitun toiminnallisuuden lisäksi työ on tarpeellista toteuttaa niin, että se mahdollistaa kertakirjautumisjärjestelmätuen. Työssä on myös huomioitava lokalisointi, jotta sovellus olisi käytettävissä myös eri kielisille käyttäjille.

5 SUUNNITTELU JA KÄYTETYT KIRJASTOT

Sovelluskehityksen alkumetreillä testattiin erilaisia tapoja, joilla sovelluksen perustoiminnallisuudet voitaisiin toteuttaa. Tarkoituksena oli tutkia eri tapoja toteuttaa halutut toiminnallisuudet ja analysoida niiden hyötyjä ja haittoja. Sovelluksen toiminnot voidaan jakaa karkeasti kolmeen osaan, asiakirjojen hakuun, noutamiseen sekä tulostamiseen.

Sovellus tulee tarvitsemaan muutamia kirjastoja, nopean kehitystyön mahdollistamiseksi. Käytettävät kirjastot ovat kolmannen osapuolen puolen luomia avoimen lähdekoodin kirjastoja, jotka sallivat kaupallisen käytön, jakelun ja julkaisun.

5.1 Asiakirjojen haku, noutaminen ja tulostaminen

Asiakirjojen haun toteuttamiseen löytyi työn edetessä kaksi eri vaihtoehtoa, joilla molemmilla on omat hyötynsä ja haittansa. Haku voitaisiin toteuttaa luomalla oma hakukäyttöliittymä, joka hakisi asiakirjat asianhallintajärjestelmän www-sovelluspalvelusta. Toinen vaihtoehto olisi luoda upotettu selainnäkymä ja käyttää asianhallintajärjestelmän omaa hakukäyttöliittymää tiedostojen hakuun.

Oman hakukäyttöliittymän luomisen eduksi voidaan nähdä se, että haun käytettävyyttä voitaisiin parantaa. Koska sovellus hakisi tietonsa www-sovelluspalvelun kautta, olisi se riippuvainen vain sen toiminnoista haun toteutuksessa. Www-sovelluspalvelun tarkastelussa havaittiin, että mikään sen yksittäisistä toiminnoista ei tuota samoja tuloksia, mitä asianhallintajärjestelmän hakunäkymä tuottaa. Tämä tarkoittaa sitä, että täysin samojen hakutuloksien toteuttaminen tällä menetelmällä olisi hyvin haastavaa, ellei mahdotonta. Www-sovelluspalvelujen käyttö ei tulisi kuitenkaan rajoittumaan vain tiedostojen hakuun. Sitä tultaisiin hyödyntämään käyttäjän kirjautumistietojen tarkistamiseen ja asiakirjojen noutamiseen.

Upotetun selaimen ja asianhallintajärjestelmän oman hakunäkymän hyödyntämisessä on myös omat puolensa. Upotettu selain voitaisiin lisätä omana kontrollinaan näkymään. Sopivan kontrollin löytäminen ja käyttäminen vaatii kuitenkin oman aikansa. Toiminnallisuus olisi kuitenkin riippuvainen asianhallintajärjestelmän sivuston rakenteesta ja elementeistä. Sovellus ei välttämättä toimisi halutulla tavalla tai ollenkaan, mikäli sivuston rakenteeseen tulisi suuria muutoksia. Tällä toteutustavalla hakutulokset tulisivat aina olemaan samat kuin korvattavassa tuotteessa, eikä uutta käyttöliittymää tarvitsisi luoda, koska asianhallintajärjestelmä tarjoaisi sen. Asiakirjojen tunnisteet löytyvät asianhallintajärjestelmän hakunäkymän HTML-elementeistä. Asiakirjojen tiedot voitaisiin niiden avulla hakea www-sovelluspalvelusta myöhempää käyttöä varten.

Asianhallintajärjestelmän oman käyttöliittymän hyödyntäminen nähtiin lopulta parempana ratkaisuna. Upotettua selainta käyttämällä käyttäjälle

voidaan tarjota lähes identtinen käytettävyys, mitä korvattava sovelma tarjoaa. Asianhallintajärjestelmän käyttöliittymän rakenteeseen ei odoteta muutoksia, mikä tarkoittaa, että vaihtoehdon huonot puolet eivät todennäköisesti pääse korostumaan.

Asiakirjojen noutamiseen vain yksi varteenotettava toteutustapa – www-sovelluspalvelun käyttäminen. Asiakirjojen tunnisteiden avulla asiakirjoihin liittyvät tiedostot haetaan www-sovelluspalvelusta. Sovelluspalvelu palauttaa tarvittavat tiedoston tiedot kuten tiedoston tyypin ja sen bitit. Kun asiakirjasta tiedetään kyseiset tiedot, se voidaan kirjoittaa väliaikaisesti koneen levylle odottamaan tulostamista.

Kun asiakirja on kirjoitettu levylle talteen, voidaan se avata tulostustarkoituksessa. Vaikka tiedostot ovat kirjoitettu biteiksi levylle, ei niitä voi suoraan lähettää tulostimelle tulostettavaksi. Kaikista tulostimista ei esimerkiksi löydy tukea doc- tai docx-tiedostotyyppisien tiedostojen suoralle tulostamiselle, mitkä ovat asiakkaan liiketoiminnassa hyvin yleisiä tiedostotyyppejä. Jotta edellä mainittujen tiedostotyyppien omaavat tiedostot voidaan tulostaa, täytyy sitä tiedostotyyppiä ymmärtävän sovelluksen toimia tulkkina tulostimelle.

Windows-ympäristössä voidaankin avata tiedostoja ohjelmallisesti, antaen niille lisäparametrina tieto siitä, että tiedosto halutaan tulostaa. Tiedosto avataan sen tiedostotyypille määritetyssä oletussovelluksessa ja sulkeutuu tulostustyön lähettämisen jälkeen.

5.2 Käytetyt kirjastot

Sovelluskehitystä halutaan tehostaa hyödyntämällä muutamia ohjelmistokirjastoja. Merkittävin käytettävistä kirjastoista on Caliburn Micro, joka avustaa MVVM-mallin toteuttamisessa. Lisäksi käytössä on CefSharp, joka tarjoaa upotetun Chromium-selaimen sovellukseen.

Caliburn Micro on ohjelmistokirjasto, joka avustaa MVVM-mallia noudattavien sovellusten kehityksessä. Se sisältää monia toiminnallisuuksia, jotka helpottavat muun muassa tiedon sidontaa ja useiden saman aikaisten näkymien hallintaa. Se tukee kaikkia XAML-merkkikieltä hyödyntäviä ohjelmointikehyksiä. (Caliburn.Micro, n.d. a) Caliburn Micro mahdollistaa XAML-elementin x:Name-ominaisuuteen perustuvan tiedon sidonnan. Jos näkymämallissa on saman niminen ominaisuus kuin jokin näkymän elementti, Caliburn Micro pyrkii tekemään tiedon sidonnan niiden välille. (Caliburn.Micro, n.d. b)

CefSharp on kirjasto, joka mahdollistaa upotetun Chromium-verkkoselaimen asettamiseen C#-, tai VB.NET-sovellukseen. Se tarjoaa WPF-sovelluksissa hyödynnettävän selainkontrollin lisäksi ajon ilman käyttöliittymää. Selaimessa sisältöä voidaan manipuloida käyttäen .NET—JavaScript-siltaa. Ohjelmallisesti voidaan laukaista ja syöttää JavaScriptiä

sivuille sekä saada palautusviestejä JavaScript tapahtumien laukaisusta. (CefSharp, n.d.)

6 TOTEUTUS

Sovelluksen rakenne jaottuu MVVM-mallin mukaisesti kolmeen eri osa-alueeseen: malliin, näkymään ja näkymämalliin. Malli sisältää käyttöliittymästä irrallisen sovelluslogiikan, näkymä käyttöliittymän rakenteen ja tyylin. Näkymämalli taas toimii näkymän ja mallin yhdistävänä tekijänä ja tarjoaa näkymälle sen tarvitsemat toiminnot mallista.

Malli sisältää datamallin lisäksi myös sovelluksen kannalta oleelliset toiminnallisuudet, jotka on jaoteltu omiin luokkatiedostoihinsa. Toiminnallisuuksia on muun muassa asiakirjojen tiedustelu, noutaminen ja tulostaminen. Mallin sisälle on myös luotu luokkatiedosto sovelluksen sisältämille asetuksille.

Näkymä taas määrittelee sovelluksen käyttöliittymän ulkoasun ja rakenteen. Sovelluksessa on kolme näkymää. Kirjautuminen ja asiakirjojen haku on toteutettu yhteen näkymään käyttäen CefSharp-kirjaston tarjoamaa upotettua Chromium-verkkoselainta ja toiminnanohjausjärjestelmän omaa käyttöliittymää. Sovelluksessa on myös kaksi tulostukseen liittyvää näkymää: tulostuksen päänäkymä ja tulostuksen tilannenäkymä.

Jokaisella näkymällä on oma näkymämallinsa. Näkymämalleissa on hyödynnetty mallien tarjoamia toiminnallisuuksia, jotka on tiedon sidonnan avulla tarjottu näkymille.

6.1 Malli

Malli pitää sisällään datamallin, joka tämän sovelluksen tapauksessa kuvaa asiakirjaa ja siihen liittyvää dataa. Asiakirjalle olennaista tietoa on muun muassa sen yksilöivä tunnus, nimike ja luontiajankohta. Lisäksi datamalli käsittää tiedoston sisältämät bitit ja tiedostopäätteen. Www-sovelluspalvelu tarjoaa tarvittavan datamallin, joten erillisiä luokkatiedostoja ei tarvittu muodostaa.

Malliin on toteutettu sovelluksen tarvitsemat toiminnallisuudet. Www-sovelluspalvelun väliseen kommunikointiin luotiin oma käsittelijäluokka, johon lisättiin metodit asiakirjojen tietojen noutamiseen, asiakirjojen tiedostojen noutamiseen sekä käyttäjän autentikointiin.

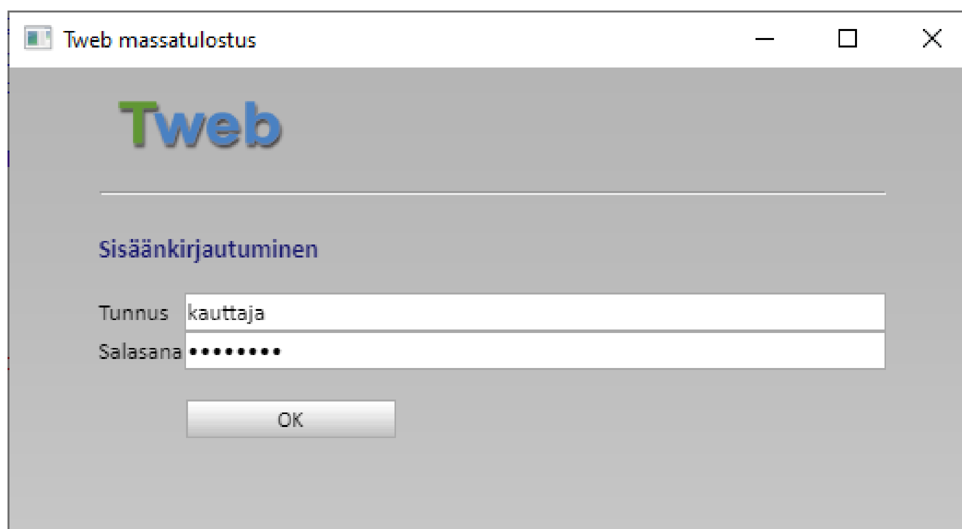
Tiedoston tulostamiseen luodut metodit liitettiin myös osaksi mallia. Kun tiedosto on noudettu, se asetetaan tietokoneelle väliaikaiskansioon. Tiedosto käynnistetään ohjelmallisesti ja sille annetaan parametrina tieto, että kyseessä on tulostustyö. Tiedosto aukeaa oletussovelluksessa vain tulostustyötä varten ja sulkeutuu heti sen suoritettuaan. Väliaikaiskansio ja kaikki sen sisältämät tiedostot poistetaan tulostustyön päätyttyä.

Sovelluksella on muutama asetus, jotka luetaan Windows-käyttöjärjestelmän rekisteristä sovelluksen käynnistyessä. Asetukset pitävät sisällään

muun muassa URL-tiedot www-sovelluspalvelusta ja asianhallintajärjestelmästä sekä lokalisaatiotiedon. Nämä asetukset määritetään Windowsin rekisteriin sovelluksen ulkopuolisella rekisterin muokkaustiedostolla. Lisäksi sovellus lukee ja kirjoittaa käyttäjän sessioon liittyviä tietoja kuten käyttäjänimen ja sessiotunnuksen rekisteriin, mitä käytetään www-sovelluspalveluun tehdyissä kyselyissä.

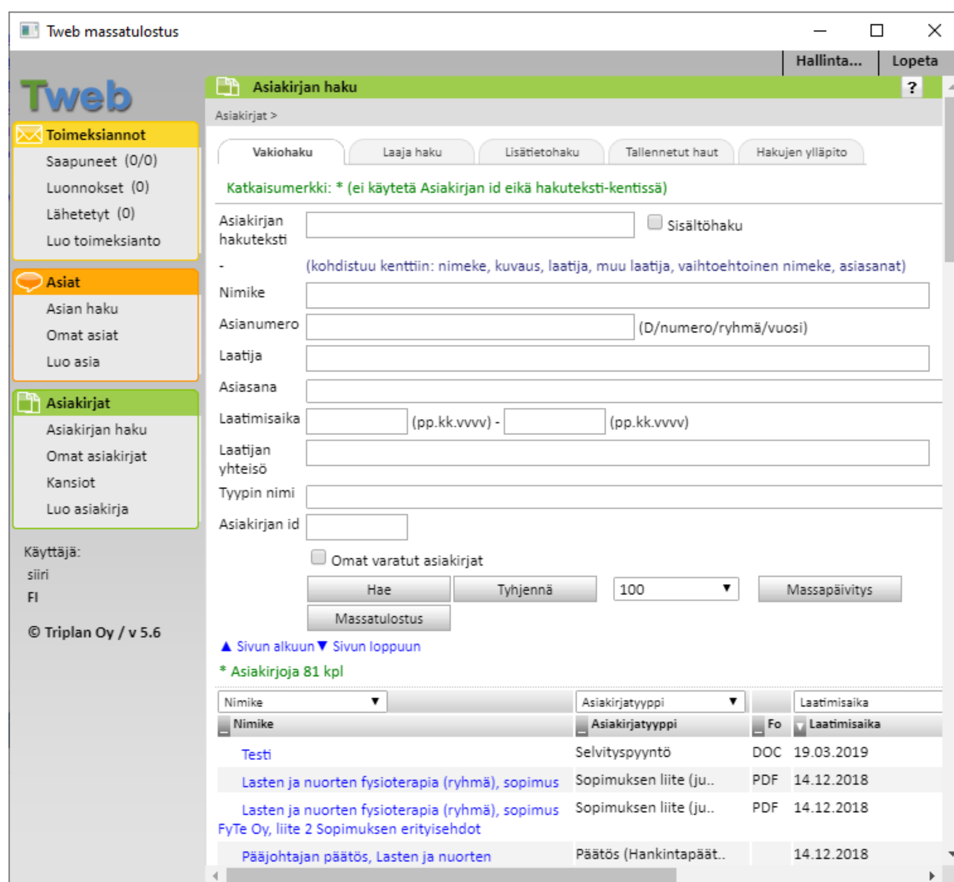
6.2 Asiakirjojen hakuikkuna

Asiakirjojen hakuikkuna on yksinkertainen, upotetun Chromium-verkkoselaimen sisältämä ikkuna. Käyttäjän tunnistautuminen ja asiakirjojen haku on toteutettu upotettua selainta hyödyntämällä.



Kuva 3. Käyttäjän tunnistautuminen

Kuvassa 3 on esitetty asianhallintajärjestelmään kirjautuminen upotetussa selaimessa. Käyttöliittymä on asianhallintajärjestelmän oma, mutta käyttöliittymän taustalla olevaa HTML-dokumenttia on manipuloitu JavaScript-ohjelmointikielellä niin, että kirjautumislomakkeen lähetyshetkellä käyttäjän tunnistautumistiedot otetaan talteen. Kyseisillä tiedoilla käyttäjä autentikoidaan www-sovelluspalveluun.



Kuva 4. Asianhallintajärjestelmän hakukäyttöliittymä

Asianhallinta järjestelmän käyttöliittymää hyödynnetään myös asiakirjojen hakuun (Kuva 4). Tunnistautunut käyttäjä voi hakea ja avata tulostustyökalun upotetusta selaimesta. Kun sovellus havaitsee, että käyttäjä on siirtynyt asiakirjan hakuun, manipuloidaan HTML-dokumenttia JavaScriptillä niin, että massatulostuspainike avaa sovelluksen toisen ikkunan — Asiakirjojen tulostusikkunan.

Asiakirjojen haun näkymästä tuli hyvin yksinkertainen, mikä näkyy esimerkistä 13. Se sisältää juurielementin lisäksi vain ChromiumWebBrowser-elementin, jolle on määritetty tiedon sidonnalla sen selaimen osoite. Selaininstanssi (WebBrowser) on upotetun selaimen ominaisuus, joka on sidottu yksisuuntaisesti lähteeseen, mikä tarkoittaa, että kyseisen ominaisuuden päivittyessä se päivittää myös siihen sidotun, näkymämallissa olevan lähdeominaisuuden. Tämä tehtiin, jotta CefSharp-kirjaston .NET—JavaScript-sillan käyttäminen onnistuisi näkymämallissa.

Esimerkki 13. Selainelementin sisältävä XAML-merkintäkielinen näkymä

```
<Window
  …>
  <cefsharp:ChromiumWebBrowser
    Address="{Binding TwebURL}"
    WebBrowser="{Binding Browser, Mode=OneWayToSource}"/>
</Window>
```

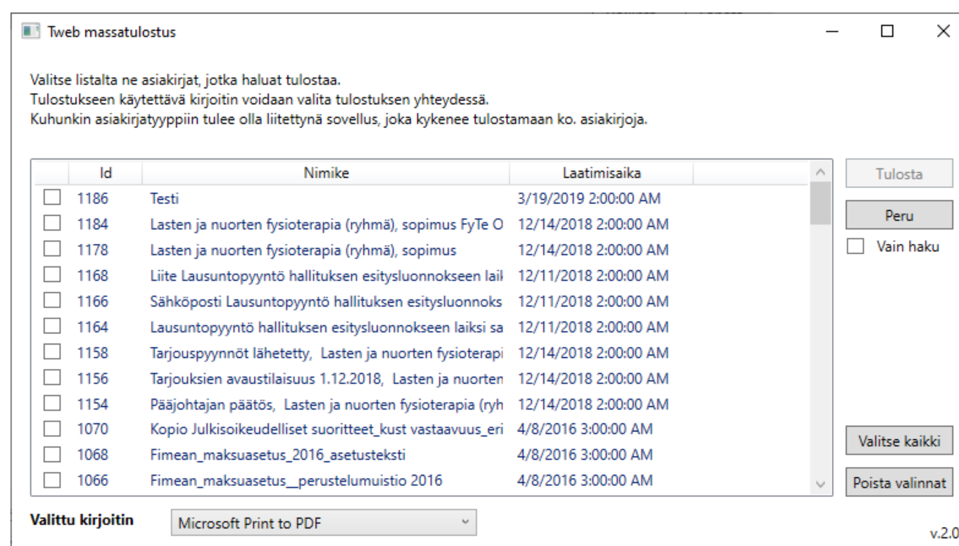
Näkymä sisälsi vain upotetun selainelementin ja näkymämallin metodit liittyivät sen toiminnallisuuksien toteuttamiseen. Näkymämallissa selaimelle lisättiin OnLoadingStateChangedAsync-tapahtuman käsittelijä, joka laukaistaan, kun selaimen latauksen tila muuttuu. Tapahtuman käsittelijän avulla HTML-dokumenttia manipuloidaan, mikäli sovellukseen asetetut ehdot täyttyvät.

HTML-dokumentista tarkistetaan, löytyykö kirjautumislomakkeeseen tai hakulomakkeeseen ja -tuloksiin viittaavia HTML-elementtejä. Mikäli elementtejä löytyy, manipuloidaan sivustoa asianmukaisesti. Kirjautumisnäkymään viittaavien elementtien ollessa läsnä manipuloidaan HTML-elementtiä niin, että kirjautumistiedot otetaan talteen ja kirjaudutaan www-sovelluspalveluun käyttäen mallista löytyvää www-sovelluspalvelun apuluokkaa. Hakunäkymään viittaavien elementtien ollessa läsnä manipuloidaan HTML-elementtiä taas niin, että massatulostus painikkeelle asetetaan uusi tehtävä — tulostuksen pääikkunan avaaminen.

Tulostuksen pääikkunan avaamisen kannalta on kuitenkin oleellista noutaa hakutuloksissa olevien asiakirjojen tiedot. Asiakirjojen tunnukset on piilotettu hakutulostaulukon HTML-elementteihin. Näiden avulla asiakirjatiedot voidaan hakea www-sovelluspalvelulta ennen tulostuksen pääikkunan avaamista.

6.3 Asiakirjojen tulostuksen pääikkuna

Asiakirjojen tulostuksen pääikkuna (Kuva 5) mahdollistaa aiemmin haettujen asiakirjojen valitsemisen tulostamiseen. Ikkunan toteutuksessa pyrittiin luomaan käyttäjälle liitteessä 1 näkyvän aiemman toteutuksen kaltainen käyttökokemus. Käyttöliittymä on pyritty pitämään samanlaisena koulutus- ja ohjeistustyön vähentämiseksi.



Kuva 5. Tulostuksen pääikkuna

Asiankirjat voidaan valita yksitellen tai valitsemalla kaikki. Aiemmassa toteutuksessa ollut ”Vain Haku”-toiminnallisuus on tuotettu myös tähän sovellukseen. Se mahdollistaa asiakirjojen noutamisen asianhallintajärjestelmästä ilman tulostustyötä. Kyseinen toiminnallisuus on tarpeellinen tilanteissa, joissa joudutaan selvittämään tiedostojen noutamiseen liittyviä ongelmia.

Aiemmassa toteutuksessa tulostuksessa käytettävä kirjoitin oli aina työaseman oletuskirjoitin. Tästä poiketen sovellukseen on luotu käytettävän kirjoittimen valinta, minkä oletus arvoksi asetetaan työaseman oletuskirjoitin. Lisätoiminnallisuus saatiin lisättyä muokkaamatta käyttöliittymän ilmettä paljoa. Sitä käyttäen käyttäjä voi määrittää mille kirjoittimelle tulosteet halutaan.

Tulostuksen pääikkunan näkymä on sovelluksen monimutkaisin. Näkymästä pyrittiin saada skaalautuva eri näyttöresoluutioille. Näkymän juurielementtinä on Window-elementti, jonka sisällä on näkymän muut elementit sisältävä Grid-elementti. Grid-elementtiä hyödyntämällä voitiin helposti valita, mitkä elementit vievät Window-elementin kasvussa muodostuneen tilan. Tämä toteutettiin muokkaamalla Grid-elementin ColumnDefinitions- ja RowDefinitions-ominaisuuksia esimerkin 14 mukaisella tavalla.

ColumnDefinition- ja RowDefinition-ominaisuuksien ja määrä kertoo, kuinka monta saraketta ja riviä Grid-elementin sisältää. Sarakkeissa on sarakeleveyden määrittävä Width-ominaisuus ja riveissä rivikorkeuden määrittävä Height-ominaisuus, joille on annettu joko lukittu numeerinen pikselimäärä, asteriski tai sarakesisällön mukaan määrittyvä ”Auto”-arvo. Jos rivikorkeudeksi tai sarakeleveydeksi on määritetty asteriski, vie se näkymässä jäljellä olevan tilan.

Esimerkki 14. Grid-elementin rivien ja sarakkeiden leveyksien ja korkeuksien määrittäminen

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="15"/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="15"/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="15"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
    <RowDefinition Height="Auto"/>
```

```
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="15"/>
  </Grid.RowDefinitions>

  <TextBlock
    Grid.Row="1"
    Grid.Column="1"
    Grid.ColumnSpan="4"
    …/>

  <ListView
    Grid.Row="2"
    Grid.Column="1"
    Grid.ColumnSpan="2"
    Grid.RowSpan="6"
    …/>
  …
</Grid>
```

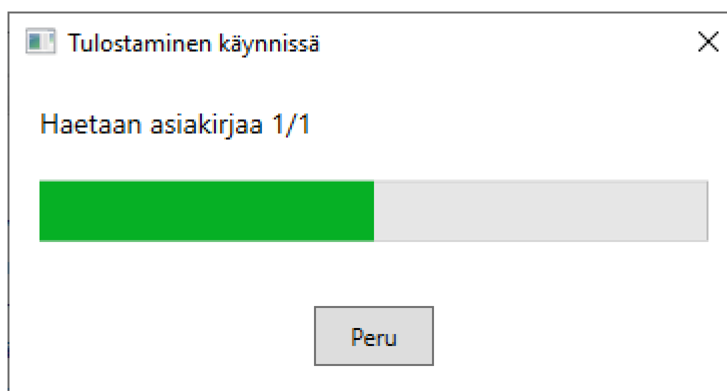
Esimerkistä näkyy myös, kuinka Grid-elementin sisäisille elementeille voidaan asettaa ColumnSpan- ja RowSpan-ominaisuudet, joilla elementti voidaan asettaa useamman sarakkeen levyseksi tai rivin korkuiseksi. Esimerkiksi ikkunasta suurimman tilan vievä ListView-elementti on kuuden rivin korkuinen ja kahden sarakkeen levyinen.

Näkymämallissa kerättiin näkymän tarvitsemat tiedot. Tulostuksen pääikkuna tarvitsi tiedon oletuskirjoittimista ja muista kirjoittimista. Nämä tiedot on jaettu näkymälle tiedon sidonnalla.

Asiakirjojen tiedot on kerätty ennen kuin tulostuksen pääikkunan näkymämallista luodaan instanssi. Nämä tiedot välitetään näkymämallille sille luodussa rakentimessa ja asetetaan näkymämallin ominaisuuteen tiedon sidontaa varten. Kyseinen lähdeominaisuus on näkymän ListView-elementin ItemSource-ominaisuudelle, joka määrittää elementin sisällön.

6.4 Asiakirjojen tulostuksen tilanneikkuna

Kun tulostuksen pääikkunasta on painettu tulostuspainiketta, avataan tulostuksen tilanneikkuna (Kuva 6). Ikkuna informoi käyttäjää tulostuksen tilanteesta tekstillä, ja tilannepalkilla. Tulostusoperaatio voidaan keskeyttää näkymän painikkeesta.



Kuva 6. Tulostuksen dialogi

Tulostustilanteen näkymä on rakenteellisesti yksinkertainen. Esimerkissä 15 kuvataan tulostustilannenäkymän XAML-rakenne. Ikkunan sisälle on tulostuksen pääikkunan mukaisesti asetettu Grid-elementti, jonka sisällä näkymän muut elementit ovat. Grid-elementin sisällä on elementit tilannetekstille, -palkille ja perumispainikkeelle.

Esimerkki 15. Tulostustilanteen näkymän XAML-tiedosto

```
<Window ...
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="15"/>
      <ColumnDefinition Width="*"/>
      <ColumnDefinition Width="15"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="15"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
      <RowDefinition Height="15"/>
    </Grid.RowDefinitions>
    <TextBlock Grid.Column="1" Grid.Row="1" .../>
    <ProgressBar Grid.Column="1" Grid.Row="2" .../>
    <Button Grid.Column="1" Grid.Row="3" .../>
  </Grid>
</Window>
```

Näkymämalli tekee tiedostojen noudon ja tulostuksen taustatyönä, jotta käyttöliittymä voidaan pitää responsiivisena. Tiedon sidontaa hyödyntämällä käyttöliittymän tiedot päivitetään tiedoston noudon ja tulostustyön siirron päätteeksi.

Asiakirjojen käsittelyn päätteeksi näytetään käyttäjälle yhteenveto. Tiedoston haussa ilmenneet ongelmat kirjataan päänäkymään ilmestyneeseen virhelokikenttään.

7 YHTEENVETO

Opinnäytetyön teoriaosuudessa käsiteltiin MVVM-arkkitehtuurimallia ja sen rakennetta, hyötyjä ja haittoja. Työn teoriaosuudessa käytiin myös läpi WPF-käyttöliittymäkirjastoa ja sen oleellisimpia toiminnallisuuksia etenkin MVVM-mallia noudattavan sovelluksen näkökulmasta.

Työn toiminnallisessa osuudessa suunniteltiin ja luotiin MVVM-mallia noudattava, WPF-kehystä käyttävä tulostustyökalu. Sovellukselle määritettiin käyttötapaus, joka kuvaa karkeasti sovelluksen tavoitetta. Työkalu pohjautui vahvasti sitä edeltäneeseen tulostusratkaisuun, joka täytti toimeksiantajan tarpeet toiminnallisuuksiltaan, muttei vanhentuneiden teknologioiden vuoksi ollut riittävä.

Toiminnallisen työn tuotoksena syntyi toimiva Windows-työpöytäsovellus, joka täytti sovellukselle asetetun käyttötapauksen ja muistutti toivotusti aiempaa työkalua. Pienenä lisätoiminnallisuutena sovellukseen lisättiin kirjoittimen valintakenttä, joka mahdollistaa eri kirjoittimen käytön tulostuksessa. Luotu sovellus on esitelty toimeksiantajalle, joka oli tuotteeseen tyytyväinen. Sovellukseen ei luotu testejä, mikä on sovelluksen selkeä kehityskohta. Testien teko on kuitenkin MVVM-mallin noudattamisen vuoksi helpompaa.

MVVM-malli ja WPF-kehys olivat tekijälle ennestään tuntemattomia ja niihin perehtyminen vei työn alkumetreillä huomattavan määrän aikaa ennen kuin sovellusta pystyttiin kunnolla aloittamaan. MVVM-malli tulee hyödyllisemmäksi, mitä suuremmasta sovelluksesta on kyse. MVVM-mallin oppimiseen meni odotettua kauemmin aikaa ja sen hyödyt jäivätkin vähäisemmiksi. Luodun, pienikokoisen sovelluksen ylläpito on kuitenkin helpompaa MVVM-mallin noudattamisen vuoksi.

LÄHTEET

Caliburn.Micro. (n.d.). Caliburn.Micro 'Xaml made easy'. Haettu 25.2.2020 osoitteesta https://caliburnmicro.com/

Caliburn.Micro. (n.d.). Introduction. Haettu 25.2.2020 osoitteesta https://caliburnmicro.com/documentation/introduction

CefSharp. (n.d.). General Usage. Haettu 25.2.2020 osoitteesta https://github.com/cefsharp/CefSharp/wiki/General-Usage

Garofalo, R. (2011). *Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern*. Sebastopol: Microsoft Press.

Nathan, A. (2010). *WPF 4 Unleashed*. Indianapolis: Sams.

MacDonald, M. (2012). *Pro WPF 4.5 in C#*. London: Apress.

Microsoft. (2017 a). The Model-View-ViewModel Pattern. Haettu 16.1.2020 osoitteesta https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

Microsoft. (2017 b). Data binding overview - WPF. Haettu 18.2.2020 osoitteesta https://docs.microsoft.com/fi-fi/dotnet/desktop-wpf/data/data-binding-overview

Microsoft. (2017 c). Commanding Overview - WPF. Haettu 20.2.2020 osoitteesta https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/commanding-overview

Microsoft. (2018 a). Data binding and MVVM. Haettu 1.2020 osoitteesta https://docs.microsoft.com/en-us/windows/uwp/data-binding/data-binding-and-mvvm

Microsoft. (2018 b). Getting Started (WPF). Haettu 20.1.2020 osoitteesta https://docs.microsoft.com/fi-fi/dotnet/framework/wpf/getting-started/

Microsoft. (2019). Styles and templates - WPF. Haettu 24.2.2020 osoitteesta https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/styles-templates-overview

Sells, C. & Griffits, I. (2007). *Programming WPF, Second Edition*. Sebastopol: O'Reilly.

Vice, R. & Siddiqi, M. (2012). *MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF*. Birmingham: Packt Publishing.

Yuen, S. (2017). *Mastering Windows Presentation Foundation*. Birmingham: Packt Publishing.

Liite 1

TPrinter – Tulostaminen aiemmalla sovelluksella

Tweb

Sisäänkirjautuminen

Tunnus käyttäjä

Salasana ●●●●●●●●

OK

Kuva 7. Kirjautuminen asianhallintajärjestelmään

Linkit... Hallinta... Lopeta

Asiakirjan haku ?

Asiakirjat >

Vakiohaku | Laaja haku | Lisätietohaku | Tallennetut haut | Hakujen ylläpito

Katkaisumerkki: * (ei käytetä Asiakirjan id eikä hakuteksti-kentissä)

Asiakirjan hakuteksti: Perehdyttämissuunnitelma: Tomi ☐ Sisältöhaku

- (kohdistuu kenttiin: nimeke, kuvaus, laatija, muu laatija, vaihtoehtoinen nimeke, asiasanat)

Nimike

Asianumero (D/numero/ryhmä/vuosi)

Laatija

Asiasana

Laatimisaika (pp.kk.vvvv) - (pp.kk.vvvv)

Laatijan yhteisö

Tyypin nimi

Asiakirjan id

☐ Omat varatut asiakirjat

Hae | Tyhjennä | 100 | Massapäivitys

Massatulostus

▲ Sivun alkuun ▼ Sivun loppuun

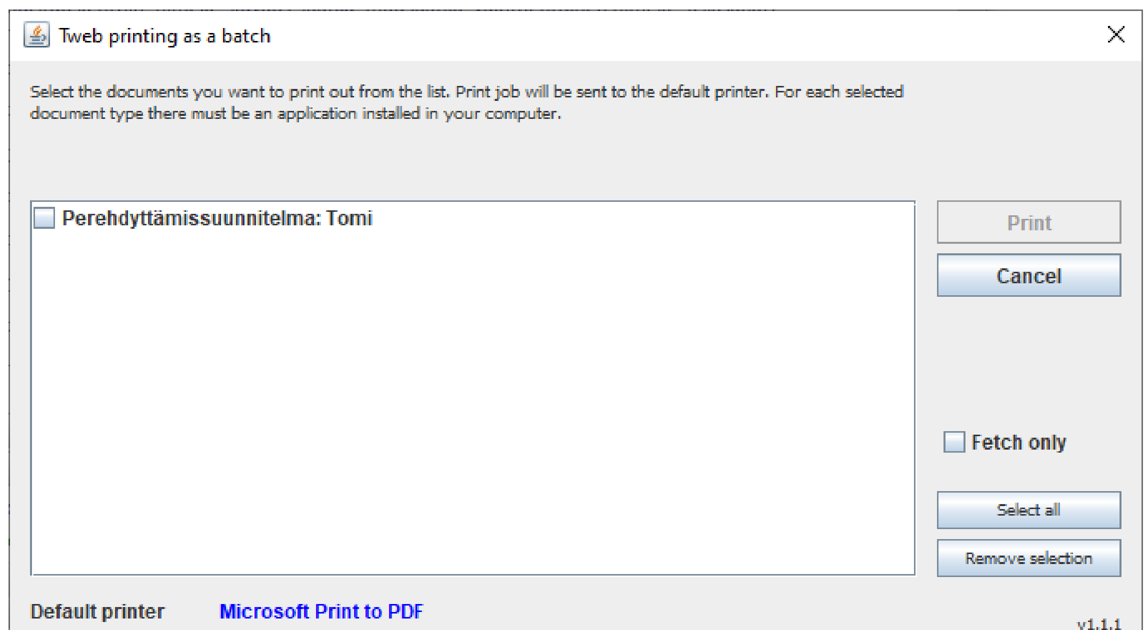
* Asiakirjojä lähetetty tulostettavaksi: 1/1

Nimike	Asiakirjatyyppi	Fo	Laatimisaika	Oik Ti
Perehdyttämissuunnitelma: Tomi	Perehdyttämismateria..	DOC	08.05.2019	- O

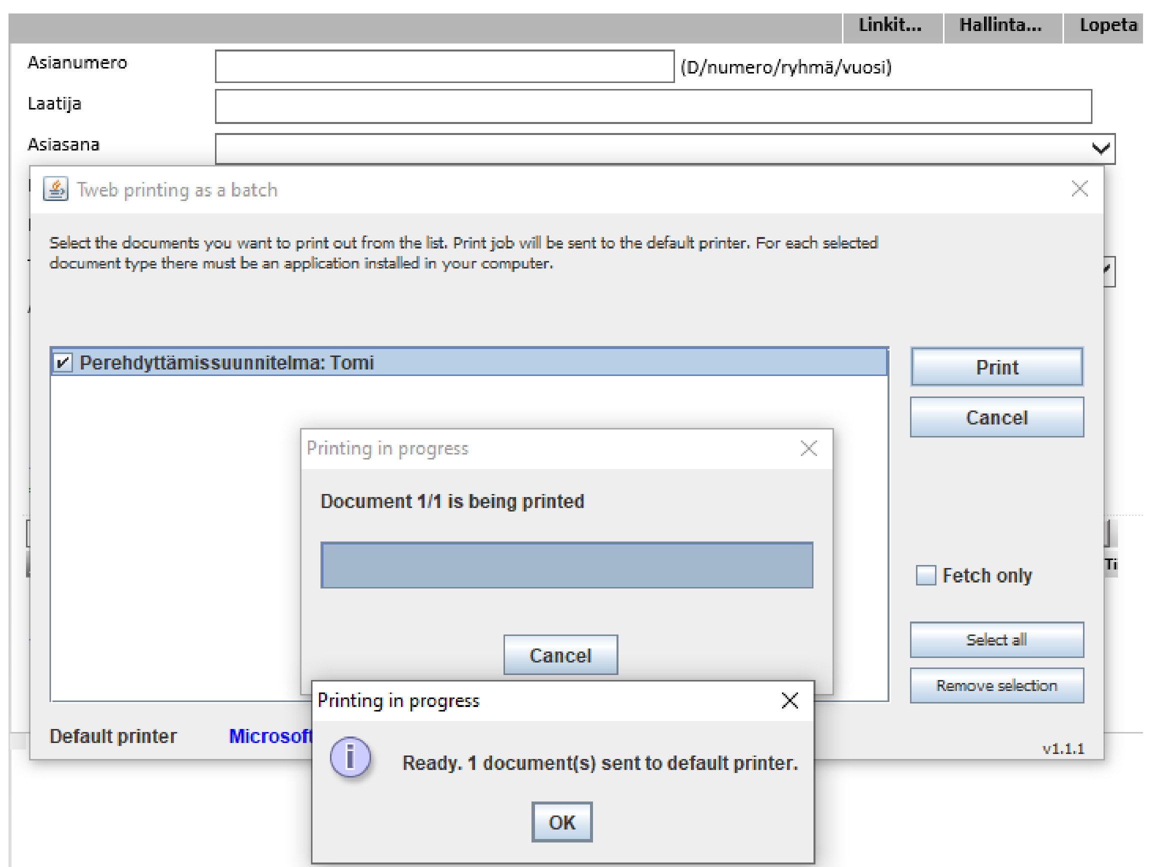
▲ Sivun alkuun

Arkistoluettelo/Asiakirjan nimikkeen mukaan | PDF | Tulosta

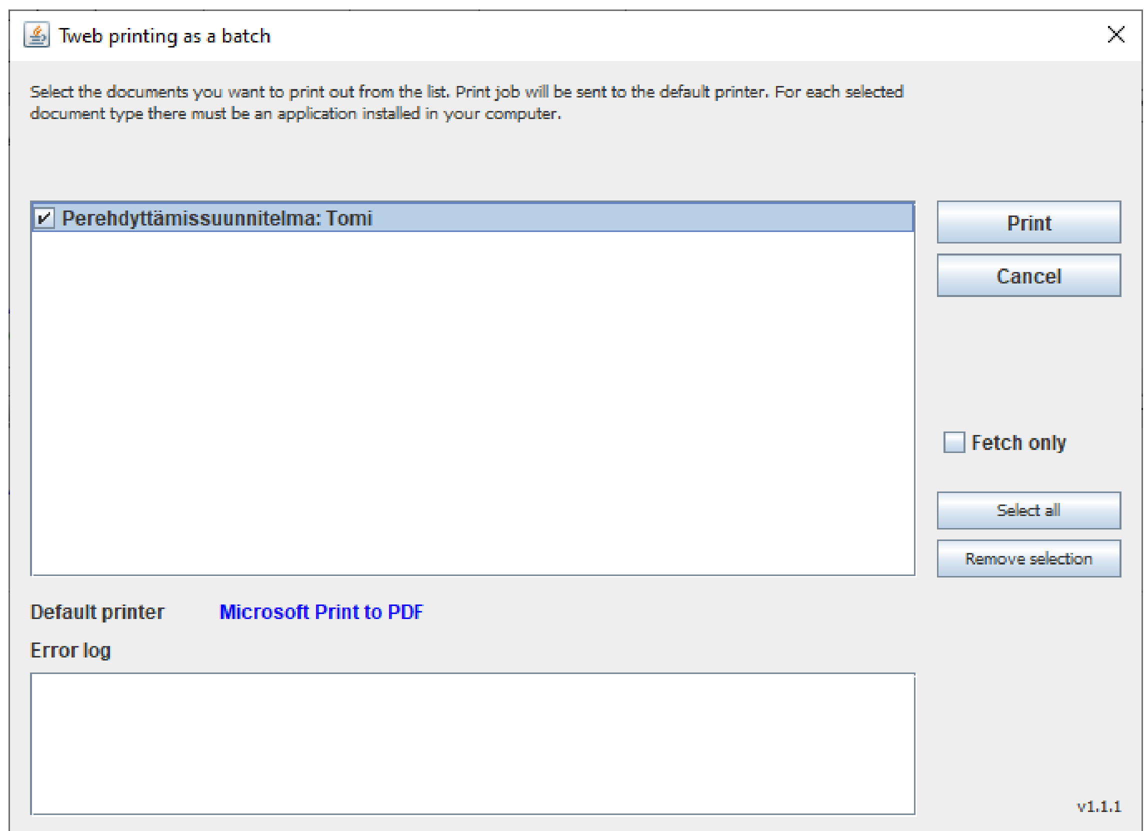
Kuva 8. TWebin hakunäkymä asiakirjoille



Kuva 9. TPrinter-sovelman päänäkymä



Kuva 10. Sovelluksen antamat tilannetiedot



Kuva 11. Tulostuksen jälkeinen virhelokin ilmaantuminen