Tampereen ammattikorkeakoulu

Visual Studio -laajennus: Älykäs koodin täyttö XML-tiedostoille

Samu Koivulahti

OPINNÄYTETYÖ
Huhtikuu 2021

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

KOIVULAHTI, SAMU:
Visual Studio -laajennus: Älykäs koodin täyttö XML-tiedostoille

Opinnäytetyö 32 sivua
Huhtikuu 2021

Visual Studio on yksi laajennetuimmista kehitysympäristöistä. Laajennukset auttavat päivittämään isäntäsovellusta pitäen käyttäjän sovelluksen ympäristössä. Visual Studio -laajennusten kehityksen aloitukseen löytyy paljon tietoa, mutta oikeaoppisen rakenteen lisääminen projektiin voi tuottaa hankaluuksia.

Opinnäytetyön toimeksiantajana toimi Entteri Professional Software Oy, joka on suomalainen terveydenhuollon tietojärjestelmätoimittaja. Entteri Oy kehittää päätuotteenaan AssisDent-potilastietojärjestelmää. Entterin sisäiseen käyttöön toivottiin Visual Studio -kehitysympäristöön laajennusta, jonka tarkoituksena oli tehostaa XML-editorin toiminnallisuutta.

Tämän opinnäytetyön tarkoituksena on selventää Visual Studio -laajennuksen kehitysprosessia käyttäen malli-näkymä-näkymämalli -suunnittelumallia. Opinnäytetyö haki vastauksia kysymyksiin: “Mitkä suunnittelumallit ovat parhaiten yhteensopivia Visual Studio -laajennusten kehityksessä?”, ja “Mitä MVVM-suunnittelumallin kanssa yhteensopivia rajapintoja Microsoftin kirjastot tarjoavat?”

Opinnäytetyön tuloksena kehitettiin Visual Studio -laajennus kehitysympäristön XML-editoriin. XML-tiedoston muokkaamisen tehostamiseksi laajennus automatisoi ohjelmoijalle annettavia täyttöehdotuksia.

Keskeisimpinä havaintona todettiin Microsoftin kirjastojen tarjoavan laajan valikoiman rajapintoja, joita voidaan hyödyntää Visual Studio -laajennusten kehittämiseen. Huomattiin myös, että Microsoft ei ole julkaissut MVVM-suunnittelumallista virallista dokumentaatiota, jonka seurauksena virheettömän MVVM-mallin toteuttaminen voi olla hidasta ja hankalaa.

Asiasanat: Visual Studio laajennus MVVM kehitysprosessi

ABSTRACT

Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

KOIVULAHTI, SAMU:
Visual Studio -extension: Code Completion for XML-files

Bachelor's thesis 32 pages
April 2021

Visual Studio is one of the most extended development environments. Extensions assist updating the parent program whilst keeping the user within its environment. There exists a lot of information on how to start developing Visual Studio extensions but implementing the correct design pattern to the program can cause some difficulties.

The client of this bachelor's thesis was Entteri Professional Software Oy, a Finnish information systems provider. Entteri Oy wished for a Visual Studio extension that increases the functionality of the XML-editor.

The purpose of this thesis was to clarify the process of creating Visual Studio extensions that implements the Model-View-ViewModel pattern. The thesis sought to answer the following questions: "Which design patterns are most compatible with Visual Studio extension development" and "What interfaces do Microsoft's libraries offer that are the most compatible with the MVVM pattern?"

An extension for Visual Studio's XML editor was developed as a result of this thesis. In the way of improving the editing process of XML-files, the extension automates the code completions for the user.

The most significant findings of this thesis were that there is an extensive range of interfaces within Microsoft's libraries that can be used in developing Visual Studio extensions.

Key words: Visual Studio extension MVVM software development process

SISÄLLYS

LYHENTEET JA TERMIT

Alentuva näkymä	(engl. humble view) Käyttöliittymä, joka ei itse hallitse näyttämänsä datan logiikkaa. Näkymän logiikka on siirretty omaan luokkaansa, jota kutsutaan näkymämalliksi.
Esittäjä	(engl. presenter) MVP-suunnittelumallin osa, joka hoitaa datan esittämistä näkymälle rajapinnan kautta.
Laajennus	(engl. extension) tietokonekoneohjelma, joka lisää uusia ominaisuuksia isäntäohjelmaan.
Malli	(engl. model) MVVM-suunnittelumallin liikelogiikan käsittelevä osa. Käytetään myös nimellä istunnon tila.
MVC	(engl. Model-View-Controller) Malli-näkymä-ohjain eli MVC-suunnittelumalli.
MVP	(engl. Model-View-Presenter) Malli-näkymä-esittäjä eli MVP-suunnittelumalli.
MVVM	(engl. Model-View-ViewModel) Malli-näkymä-näkymämalli eli MVVM-suunnittelumalli, jonka tarkoituksena on erottaa käyttöliittymä ohjelman muusta toimintalogiikasta.
Näkymä	(engl. view) MVVM-suunnittelumallin käyttöliittymä.
Näkymämalli	(engl. viewmodel) MVVM-suunnittelumallin näkymässä esitettävän datan muokkaava osa.
Näkymätila	(engl. view state) Seurantatekniikka, jota käytetään seuraamaan käyttäjän antamia pyyntöjä.
Ohjain	(engl. controller) MVC-suunnittelumallin näkymän syötettä käsittelevä osa.
Ominaisuusolio	(engl. property object) Olio, joka osaa lähettää tiedonannon sen ominaisuuksissa tapahtuvista muutoksista.
SaaS	(engl. Software as a Service) Ohjelmiston hankkimistapa, jossa ohjelmisto hankitaan palveluna.
Suunnittelumalli	(engl. design pattern) Ohjelmistorakenteen määrittelevä kuvaus, kuinka luokat ja oliot ovat yhteydessä toisiinsa.

WPF	(engl. Windows Presentation Foundation) .NET sisältämä kirjasto, jolla muodostetaan sovelluksen graafinen rajapinta.
XML	(engl. Extension Markup Language) Tapa, jolla määritetään tietojen merkitsemisen rakennetta.

1 JOHDANTO

1.1 Tausta

Visual Studio on yksi laajennetuimmista kehitysympäristöistä, joiden valmistamiseen löytyy paljon informaatiota kirjoista, artikkeleista ja internetistä. Kehitysympäristön laajennettavuuden takia on Visual Studion virallisella sivustolla julkisesti ladattavissa yli 10 000 laajennusta.

Opinnäytetyössä halutaan parantaa ymmärrystä oikean suunnittelumallin valitsemiseen ja selventää miksi malli-näkymä-näkymämalli -suunnittelumalli on tämänlaisen työn kehitysprosessin kannalta paras valinta. Opinnäytetyössä tutkitaan myös erilaisia Microsoftin ja Visual Studion tarjoamia rajapintoja ja niiden käyttöä kehitysprosessissa.

Opinnäytetyön päätavoitteena on selventää Visual Studio -kehitysympäristön laajennuksien kehitysprosessia käyttämällä Visual Studio -laajennuksiin tarkoitettuja rajapintoja sekä MVVM -suunnittelumallia. Visual Studio -laajennusten kehityksestä löytyy paljon tietoa, mutta oikeaoppisen rakenteen valitseminen ja toteuttaminen voi olla hankalaa. Oikeaoppisesti rakennettu projekti mahdollistaa projektien laajentamista.

Entteri Professional Software Oy toimii tämän projektin toimeksiantajana. Visual Studio -laajennuksia tarvitaan yrityksen sisäisiin kehitysprosesseihin. On hyvä tietää, miten laajennusten kehitysprosessi tapahtuu.

1.2 Toimeksiantaja

Entteri Professional Software Oy (Entteri Oy) on vuonna 1994 perustettu suomalainen terveydenhuollon tietojärjestelmätoimittaja. Se työllistää n. 40 tuotekehityksen ja suun terveydenhuollon ammattilaista. Yritys on osa Planmeca-konsernia, joka on yksi johtavista hammashoitolaitteiden valmistajista. Entteri Oy keskittyy yksityisen suun terveydenhuollon asiakkaiden palvelemiseen.

Päätuotteenaan Entteri Oy kehittää Suomen markkinajohtavaa, Saas-pohjaista pilvipalvelua, AssisDent-potilastietojärjestelmää. Entteri Oy haluaa tarjota asiakkailleen tehokkaimmat ja kattavimmat työkalut, jotka edesauttavat potilastyötä ja liiketoiminnan kehittämistä. AssisDent hyödyntää uusimpia teknologioita, joiden avulla tuotetta voidaan käyttää tehokkaammin ja uusilla tavoilla. Entteri Oy on myös vahvasti sitoutunut viranomaisyhteistyöhön ottamalla muun muassa ensimmäisenä suun terveydenhuollon potilastietojärjestelmänä Potilastiedon arkiston tuotantokäyttöönsä.

1.3 Tehtävä

Opinnäytetyön tehtävänä on rakentaa Visual Studio -laajennus XML-tiedostojen editoriin kehittämällä Visual Studio IntelliCode -tyylinen ratkaisu XML-tiedostojen muokkaamiseen. IntelliCode on Visual Studion sisäinen koodin täydennykseen kehitetty työkalu. Laajennuksen tarve tulee yrityksen sisäisistä prosesseista, joissa Visual Studion tarjoaman XML-editorin toiminnallisuutta halutaan kehittää. Editori tarvitsee paremman ja nopeamman ratkaisun XML-tiedostotyyppien muokkaamiseen.

Visual Studion XML-editorin IntelliCode on puutteellinen ja se ei osaa tarjota tarvittavia täyttöehdotuksia yrityksen sisäisiin käyttötarkoituksiin. Tarpeena on saada Visual Studion XML-editori automatisoimaan ohjelmoijalle annettavat täyttöehdotukset.

2 VISUAL STUDIO -LAAJENNUS

2.1 Visual Studio

Visual Studio on Microsoftin kehittämä ohjelmankehitysympäristö. Se tarjoaa tehokkaan ja tuottavan kehitysympäristön ohjelmistoprojektien rakentamiseen, kehittämiseen sekä testaukseen. Visual Studio auttaa käyttäjää rakentamaan muun muassa korkealaatuisia ohjelmistoja, verkkosivuja sekä verkko- ja mobiilisovelluksia nopeasti ja intuitiivisesti. Useat sen ominaisuudet on rakennettu yleisten kehitystehtävien ympärille. Nämä ominaisuudet pyritään rakentamaan mahdollisimman virtaviivaisesti ja optimoidusti yhden työkalun alle. (Fukizi & de Oliveira & Bruchet 2019.)

Visual Studio käyttää useita Microsoftin kirjastoja ja ohjelmistokehyksiä, kuten Windows APIa, Windows Formsia, Windows Presentation Foundationia ja Microsoft Silverlightia (Microsoft 2021). Visual Studio -kehitysympäristöstä on tarjolla useita eri versioita, joista opinnäytetyön kirjoitushetkellä uusin on Visual Studio 2019.

Fukizi & de Oliveira & Bruchet (2019) mainitsevat Visual Studio 2019 -kehitysympäristön olevan yksi tehokkaimmista laajennettavista kehitysympäristöistä, joka tukee Microsoftin kehittämiä ohjelmointikieliä. Visual Studio 2019 -kehitysympäristöä käytettiin tämän opinnäytetyön teknisen projektin valmistukseen.

2.2 Laajennus

Laajennus on tietokoneohjelma, joka lisää uusia ominaisuuksia isäntäohjelmaan. Laajennukset auttavat päivittämään isäntäsovellusta pitäen käyttäjän sovelluksen ympäristössä.

Suurta tietokoneohjelmaa kehittäessä ei ole mahdollista tiedostaa kaikkia käyttäjien käyttötarkoituksia ja tarpeita, vaikka ohjelma olisi suunniteltu erinomaisesti.

Jos ohjelmasta ei löydy tarvittavaa ominaisuutta, käyttäjät usein vaihtavat sovelluksesta toiseen tai odottavat ominaisuuden lisäämistä seuraavien päivitysten mukana. Laajennukset kiertävät tämän ongelman yhdistämällä uudet ominaisuudet isäntäsovellukseen. (Sterne 2014.)

2.3 XML

XML-tiedostoilla on useita eri käyttötarkoituksia. Tämän opinnäytetyön teknisen toteutuksen kannalta XML-tiedostoja käytetään Visual Studio -laajennuksen käyttöliittymän määrittelyyn. Laajennus tehdään XML-tiedostotyyppien editorin kanssa yhteensopivaksi, jolloin on hyvä ymmärtää XML-merkintäkielen ja XML-tiedostojen taustaa.

XML on merkintäkieli, joka suunniteltiin helposti käytettäväksi internetin välityksellä ja tukemaan monenlaisia sovelluksia. Vaihtoehtoiset ominaisuudet haluttiin pitää mahdollisimman vähäisinä ja ohjelmat, jotka käyttävät XML-merkintäkieltä, haluttiin helposti kirjoitettaviksi. XML kuvaa dataolioiden luokkaa, joita kutsutaan XML-tiedostoiksi. XML-tiedostot haluttiin olevan selkolukuisia ja helposti valmistettavia. (W3C 2007.)

XML-tiedosto sisältää loogisen ja fyysisen rakenteen. Fyysisesti XML-tiedosto rakentuu entiteetti-nimisistä varastointiyksiköistä. Entiteetti voi viitata toisiin entiteetteihin, jotka vaativat niiden lisäämistä tiedostoon. XML-tiedostojen looginen rakenne koostuu deklaraatiosta, elementeistä, kommenteista, merkkiviittauksista ja käsittelyohjeista, jotka ovat osoitettu tiedostossa täsmällisellä kuvauksella. Sekä fyysisen että loogisen rakenteen tulee olla sisäkkäistetty oikeaoppisesti (Kuvio 1). (W3C 2007.)

```
<?xml version="1.0" encoding="UTF-8"?>
<item>
  <name>Item1</name>
  <type>Type</type>
  <description>Description text</description>
</item>
```

KUVIO 1. Esimerkki XML-tiedoston rakenteesta

XAML (engl. extensible application markup language) on XML-pohjainen merkintäkieli. XAML-merkintäkieltä käytetään usein WPF-projekteissa näkymän muodostamiseen. Kaikki XAML-tiedostot ovat valideja XML-tiedostoja, mutta kaikki XML-tiedostot ei välttämättä ole valideja XAML-tiedostoja.

2.4 WPF

Windows Presentation Foundation eli WPF on Microsoftin kehittämä Windows-pohjaisten sovellusten kehittämiseen tarkoitettu graafinen alijärjestelmä. Ennen WPF:n kehittämistä 3D:n, videon ja äänen lisääminen käyttöliittymään oli hankalaa. Näiden ominaisuuksien lisääminen vaatii usean eri teknologian osaamista ja yhdistämistä ilman tähän tarkoitettua sisäistä tukea. WPF mahdollistaa näiden kaikkien ominaisuuksien lisäämisen käyttöliittymään ja niiden muokkaamisen esitysmuotoon. WPF tarjoaa myös resoluution skaalautuvuutta. (Nathan 2013.). Liikuttamalla sovellus isompaan tai pienempään resoluutioon, WPF osaa skaalata sovelluksen kokoa eri resoluutioiden välillä.

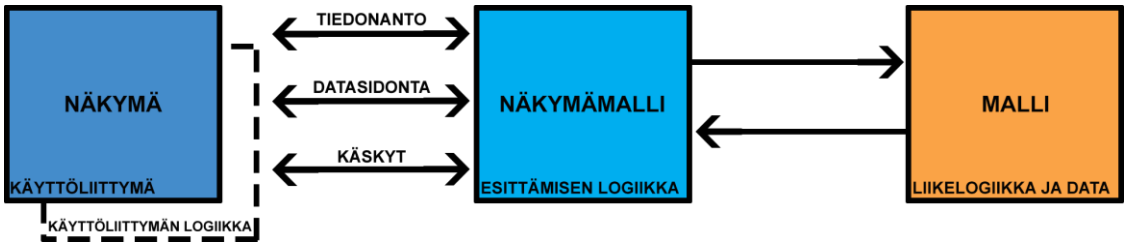
WPF käyttää resurssitiedoistoinaan XAML-tiedostoja. Ennen WPF:n julkaisua Win16 ja Win32 sovellusten käyttöliittymän määrittelemiseen käytettiin XML-merkintäkieltä. WPF:n ja XAML-merkintäkielen yhdistelmä on verrattavissa käyttöliittymän kehittämiseen HTML-merkintäkielellä. XAML:n avulla WPF sovellusten graafinen suunnittelu yksinkertaistuu ja komponenttien käyttäytymislogiikan kirjoittaminen vähenee. WPF mahdollistaa komponenttien rakentamista ja kustomointia, mitä aiemmat teknologiat eivät tarjoa. (Nathan 2013.)

3 MALLI-NÄKYMÄ-NÄKYMÄMALLI -SUUNNITTELUMALLI

3.1 Mikä on MVVM-suunnittelumalli

Malli-näkymä-näkymämalli on tarkoitettu erottamaan käyttöliittymä ohjelman muista osista. Ohjelman jako eri osiin parantaa sen testattavuutta ja uudelleen käytettävyyttä.

MVVM-suunnittelumallin tarkoitus on tehdä näkymästä alentuva näkymä tarkoittaen, että näkymä ei itse tiedä ominaisuuksistaan, jotka näkymä esittää. Näkymän ominaisuuksia käsittelevä logiikka siirretään näkymämallille, joka sitoo ominaisuuden ominaisuusolioon ja antaa sen datasidonnan avulla näkymälle näytettäväksi. Malli hakee tarvittavat ominaisuudet, jotka annetaan näkymämallille käsiteltäväksi näytettävään muotoon (Kuvio 2). MVVM-suunnittelumallista tehdään virheetön, kun käyttöliittymän logiikka ei sisällä minkäänlaista ohjelmakoodia. (Vice & Siddiqi 2012.)



KUVIO 2. Malli-näkymä-näkymämallin rakenne (Vice R. & Siddiqi, M. S. 2012, muokattu)

MVVM-suunnittelumalli kehottaa SoC:n (separation of concerns) ja SRP:n (single-responsibility principle) käyttöön (Vice & Siddiqi 2012). SoC:n tarkoituksena on jakaa tietokonesovellus erillisiin osa-alueisiin, jotta jokainen osa-alue vastaa yhdenlaisesta toiminnasta. SRP:n tarkoituksena on antaa moduulille, luokalle tai funktiolle ratkaistavaksi ainoastaan yksi sovelluksen toiminnan osa-alue. Kaikki moduulin, luokan tai funktion palveluiden kuuluu asettua tämän ongelman ratkaisuun. SoC sekä SRP tehostavat sovelluksen uudelleen käytettävyyttä.

3.1.1 Malli

MVVM-suunnittelumallin malli tai istunnon tila on vastuussa pitää muistissaan ominaisuudet. Ne haetaan pysyvästä lähteestä esimerkiksi tietokannasta. Mallin tehtävänä on antaa näkymämallille tiedonanto mahdollisista ominaisuuksien muutoksista. (Vice & Siddiqi 2012.)

Mallin suunnittelu ei käytännössä liity MVVM-rakenteeseen, sillä näkymämallille voidaan antaa laajasti erilaista dataa, jonka se voi antaa eteenpäin näkymälle. Malli ei välttämättä sisällä mitään dataa, joka määrittää sen olevan käytössä MVVM-rakenteessa tai edes WPF-sovelluksessa. Malli ei välitä mistä se datansa saa. (Smith 2009.)

3.1.2 Näkymä

MVVM-mallin näkymä on vastuussa ominaisuuksien esittämisestä sekä käyttäjän syötteen keräämisestä ja sen välittämisestä eteenpäin näkymämallille. Näkymä kommunikoi näkymämallin kanssa käyttäen datasidoksia. Datasidos tapahtuu WPF:än tai Silverlightin avulla, jolloin näkymä ei tarvitse koodia käyttöliittymän logiikkaan. (Vice & Siddiqi 2012.)

WPF-pohjaisissa projekteissa näkymä koostuu yhdestä tai useammasta XAML-tiedostosta, joihin määritetään näkymän ulkoasu sekä näkymän logiikasta vastaavasta XAML.cs -tiedostosta. Virheettömässä MVVM-suunnittelumallissa näkymän logiikan ei tule sisältää koodia.

3.1.3 Näkymämalli

Näkymämalli tai esitysmalli on vastuussa näkymätilasta ja näkymän logiikasta. Näkymämalli kommunikoi näkymän kanssa datasidonnan avulla. Näkymämalli toimii mallin ja näkymän välittäjänä antamalla mallista saatu data näkymälle. Näkymämalli antaa mallille näkymästä saadut käyttäjän syötteet. (Vice & Siddiqi 2012.)

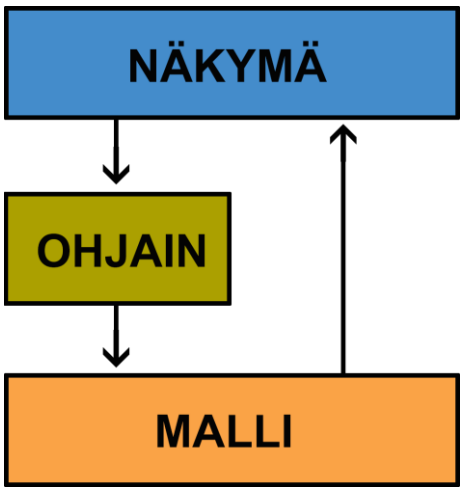
Näkymämallin tarkoitus on toimia näkymän mallina. Näkymämalli ottaa vastaan näkymän istunnon tilan ja muokkaa sen näkymätilan käytettäväksi. Näkymämalli vastaa tarvittaessa näkymän logiikan muuttamisesta näkymätilaksi. (Vice & Siddiqi 2012.)

3.2 Muita yleisiä suunnittelumalleja

Vaihtoehtoisia suunnittelumalleja ennen MVVM-mallin kehittämistä ovat olleet malli-näkymä-esittäjä (MVP) - ja malli-näkymä-ohjain (MVC) -mallit. Vice & Siddiqi (2012) mainitsevat näiden suunnittelumallien parantavan testattavuutta ja SoC:tä.

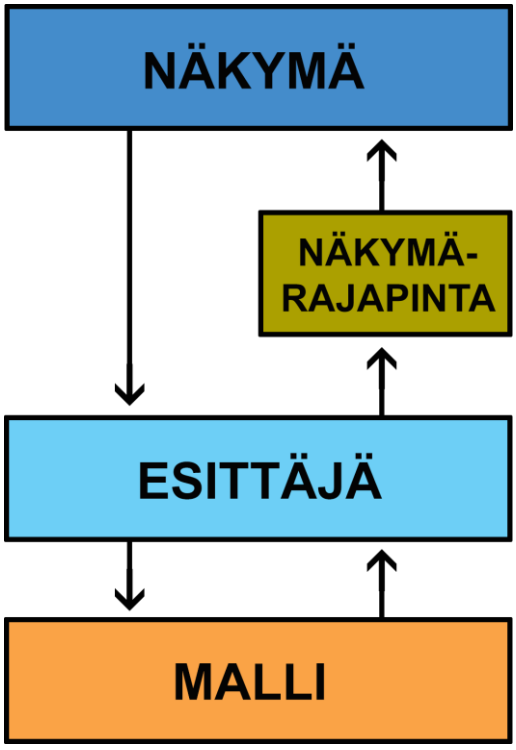
Malli-näkymä rakennetta kutsutaan esitysmalliksi. Esitysmallit ovat hyvin joustavia ja muokattavia, jolloin niitä on vaikea hallita (Vice & Siddiqi 2012). Tästä johtuen MVC-suunnittelumallista on useita versioita.

MVC-mallissa näkymä vastaa ominaisuuksien esittämisestä ja käyttäjän syötteen keräämisestä. Näkymä saa ominaisuutensa mallilta, sisältäen tiedonannot ominaisuuksien ja näkymän päivittymisestä. Näkymä kerää käyttäjän antamat ominaisuudet ja lähettää ne ohjaimelle käsiteltäväksi. Ohjain vastaa syötteen vastaanottamisesta ja viestittää sen mallille. Ohjaimen päähyöty on logiikan erittely näkymästä, joka parantaa automaattitestien toteutusta. Malli on vastuussa ominaisuuksien hakemisesta ja näkymän tiedottamisesta ominaisuuksien muutoksista (Kuvio 3). (Vice & Siddiqi 2012.)



KUVIO 3. Malli-näkymä-ohjain rakenne (Vice R. & Siddiqi, M. S. 2012, muokattu)

MVP-suunnitelumalli periytyy MVC-suunnittelumallista, mutta hieman eri lähestymistavalla. MVP-mallissa näkymän ei tarvitse kuunnella mallilta tulleita tiedonantoja. Esittäjä päivittää näkymää mallissa tapahtuneista datamuutoksista. Esittäjä ottaa myös vastaan näkymän syötteen ja antaa sen mallille. Esittäjä kommunikoi näkymän kanssa näkymärajapinnan avulla (Kuvio 4). Tämä mahdollistaa näkymän testausta, jos mallista tehdään valeolio, joka antaa valedataa esittäjälle. (Vice & Siddiqi 2012.)



KUVIO 4. Malli-näkymä-esittäjä rakenne (Vice R. & Siddiqi, M. S. 2012, muokattu)

3.3 MVVM-suunnittelumallin hyödyt

MVC-mallissa näkymän logiikka ja näkymätila ovat yhdessä näkymässä. Täten ne ovat vaikeasti testattavissa. Näkymän logiikka sekä näkymätila olivat myös sidoksissa malliin, joka ei mahdollista niiden uudelleen käytettävyyttä. (Vice & Siddiqi 2012.)

MVC-mallin ongelmat korjataan MVP-mallissa tekemällä näkymästä alistuva näkymä ja liikuttamalla näkymän logiikka ja näkymätila eri luokkiin. Alistuvan näkymän luonti tapahtuu näkymän toteuttaessa rajapinnan, jonka avulla esittäjä kommunikoi näkymän kanssa. MVP-malli ratkaisee MVC-mallin ongelman, mutta laatii sidoksen esittäjän ja näkymän rajapinnan kanssa. Tämä vaatii näkymän rajapinnan ja esittäjän manuaalista päivittämistä. (Vice & Siddiqi 2012.)

MVVM-suunnittelumallissa siirretään näkymän logiikka ja näkymätila näkymämalliin, joka tekee näkymästä alistuvan näkymän. MVP-mallin näkymän ja esittäjän sidosongelma ratkaistaan ominaisuusolioiden avulla. Näkymän ominaisuudet saadaan säilöttyä ulkoiseen ominaisuusolioon. Nämä ominaisuusoliot osaavat kuunnella ja lähettää tiedonannon ominaisuuksien muutoksista. Ominaisuusoliot käyttävät kaksisuuntaista datasidosta, joka poistaa MVP-mallin mukaisen näkymän rajapinnan toteuttamisen tarpeen. (Vice & Siddiqi 2012.)

MVVM-malli tarjoaa liikelogiikan ja käyttöliittymän erotuksen, joka edesauttaa ylläpitoa ja yksikkötestien rakentamista. Yksikkötesteihin ei vaadita näkymää, vaan yksikkötestauksen voi suorittaa näkymämallin tai mallin kautta. MVVM-mallin mukaisesti rakennetut komponentit ovat myös helposti uudelleen käytettävissä toisissa projekteissa. (Gossman 2006.)

3.4 MVVM-suunnittelumallin haitat

MVVM-malli vaatii paljon pakollista ohjelmakoodia, joka voi pienissä projekteissa olla liiallista. Isoissa projekteissa näkymämallin suunnittelu mahdollisimman yleiskäyttöiseksi voi olla haastavaa. (Gossman 2006.)

Datasidokset ovat ennalta määrättyjä ja vaikeasti testattavissa verrattuna ehdottomiin vaihtoehtoihin, missä testausta voidaan suorittaa pysäytyspisteiden avulla. Datasidokset tuottavat paljon kirjapidollista dataa. Datasidokset voivat viedä paljon suorituskykyä, jos kyseessä on suuri projekti. (Gossman 2006.)

Microsoft ei ole antanut virallista dokumentaatiota MVVM-mallin toteuttamiseen, ja kolmannen osapuolen lähteet voivat antaa ristiriitaista tietoa. Vain SilverLight ja WPF tukevat MVVM-mallin käyttöä. ASP.NET ja WinForms ovat muutamia esimerkkejä, missä MVVM-mallin käyttö on vaikeasti toteutettavissa. (Vice & Siddiqi 2012.)

4 VISUAL STUDIO -LAAJENNUS PROJEKTINA

4.1 Projektirakenne ja kehitysympäristö

Projektin kehitysympäristönä toimii Visual Studio 2019. Visual Studio tarjoaa kattavan kehitysympäristön tietokoneohjelmien ja laajennuksien kehittämiseen. Opinnäytetyössä kehitettävä laajennus tehdään myös Visual Studio 2019 -kehitysympäristöön, jossa sovelluksen testaus tapahtuu helposti ja tehokkaasti.
Visual Studio tukee Microsoftin luomaa C#-ohjelmointikieltä, jota käytetään sovelluksen kehittämiseen.

Projektirakenteesta tutkimusta tehdään parhaan rakenteen saavuttamiseksi. Harkittuja suunnittelumalleja ovat MVVM-, MVC- ja MVP-suunnittelumallit. Parhaaksi suunnittelumalliksi todetaan MVVM-suunnittelumalli, sen laajennettavuuden ansiosta. Projektin suunnittelumallin valintaan vaikuttaa myös tekijän aiempi kokemus MVVM-suunnittelumallin käytöstä aikaisemmissa projekteissa. MVVM-malli antaa myös mahdollisuuden mallin ja näkymämallin yksikkötestien valmistukseen, mitä MVC- tai MVP-suunnittelumallit eivät tue.

4.2 Kehitysprosessi

Kehitysprosessi aloitetaan tutkimalla ensin WPF-sovelluksen suunnittelumallin eri vaihtoehtoja. MVVM-suunnittelumalli toimi sovelluksen kehittämiseen erinomaisesti. MVVM-malli mahdollistaa sovelluksen laajentamista, joka oli merkittävin syy suunnittelumallin valintaan.

Kehitysprosessi vaatii toteutuksen suunnittelun, jossa tutkitaan mahdollisia rajapintoja Visual Studio IntelliCode -tyyliseen, älykkään koodin täyttö -sovellukseen. Täysin uusi WPF-projekti, joka kuuntelee tietokoneelle annettuja näppäinkomentoja, on mahdollinen lähestymistapa. Yksinkertaisempi lähestymistapa löytyy Visual Studion valmiista laajennuksen kehitykseen tarkoitetuista rajapinnoista, jotka ovat sovelluksen käyttötarkoitukseen erinomaisia.

4.2.1 Projektin aloitus

VSIX (engl. Visual Studio extension installer) on Microsoftin tarjoama Visual Studio -laajennusten hallitsemiseen kehitetty paketti. VSIX-tiedosto sisältää yhden tai useamman laajennuksen, jota Visual Studio käyttää laajennuksien asentamiseen. Laajennuksen kehitykseen tarvitsee siis tehdä uusi VSIX-projekti. VSIX-projektipohja on valmiiksi saatavilla sekä Visual Basic - että C#-ohjelmointikielillä. VSIX-projekti sisältää vain source.extension.vsixmanifest -tiedoston, joka pitää hallussaan tiedot laajennuksesta ja sen ominaisuuksista.

Visual Studio laajennuksen kehittämiseen voidaan tarvita Visual Studio SDK:n asentamista. Visual Studio SDK antaa mahdollisuuden käyttäjälle lisätä komentoja, painikkeita, menuja sekä muita käyttöliittymäelementtejä kehitysympäristöön.

4.2.2 Microsoft.VisualStudio -kirjaston rajapinnat

Microsoft tarjoaa älykäs koodin täyttö -laajennukseen useita eri rajapintoja Microsoft.VisualStudio -kirjaston alta. Rajapinnat tarjoavat eri käyttöominaisuuksia, joista projektiin tarvitaan rajapinnat näkymän luontiin, täydennyksen ja työkaluvinkin käsittelyyn sekä käskyjen kuunteluun. Osa rajapintoja toteuttavista luokista vaatii luokka-attribuutteja, jotka määrittelevät luokan toimintoja. Näitä toimintoja voi olla esimerkiksi suoritusjärjestys tai käsiteltävän tiedoston tiedostotyyppi.

TAULUKKO 1. Microsoft.VisualStudio -kirjaston tarjoamat hyödylliset rajapinnat älykäs koodin täyttö -laajennukseen

Rajapinnan nimi	**MVVM osuus**	**Käytetty toteutuksessa**
IAsyncQuickInfoSession	Malli & Näkymämalli	X
IAsyncQuickInfoSource	Malli	X
IAsyncQuickInfoSourceProvider	Malli	X
ICompletionBroker	Näkymämalli	X
ICompletionSession	Malli & Näkymämalli	X
ICompletionSource	Malli	X
ICompletionSourceProvider	Malli	X
IIntellisenseCommandTarget	Näkymä	X
IIntellisenseController	Näkymämalli	
IIntellisenseControllerProvider	Näkymämalli	
IIntellisenseFilter	Näkymämalli	
IIntellisensePresenter	Näkymä	(X)
IPopupIntellisensePresenter	Näkymä	X
IIntellisensePresenterProvider	Näkymä	X
IIntellisenseSession	Malli & Näkymämalli	(X)
IIntellisenseSessionStack	Näkymämalli	
IntellisenseKeyboardCommand	Näkymämalli	X

Taulukosta 1 nähdään, että muun muassa *IIntellisenseSessionStack*- ja *IIntellisenseFilter*-rajapinnat jätetään toteutuksesta pois. Nämä rajapinnat kuitenkin tarjoavat hyödyllisiä ominaisuuksia. *IIntellisenseSessionStack*-rajapinta auttaa käsittelemään useampaa *IIntellisenseSession*-rajapintaa. Tämän projektin toteutukseen tarvitaan kuitenkin vain yksi *IIntellisenseSession*-rajapinta, jonka *ICompletionSession*-rajapinta toteuttaa. *IIntellisenseFilter*-rajapinta tarjoaa täydennysten suodatusta. Tässä projektissa suodatus toteutetaan käyttämättä *IIntellisenseFilter*-rajapintaa. *IAsyncQuickInfoSession*-rajapinta, sekä *ICompletionSession*-rajapinta sisältävät tiedot työkaluvinkin ja täydennyksen sisällöstä. Ne liikkuvat MVVM-mallissa eri osien välillä.

Näkymämallin rajapinnat

Käskyjä kuunteleva rajapinta löytyy nimellä *IVsTextViewCreationListener*. *IVsTextViewCreationListener*-rajapinnan funktiot käynnistyvät automaattisesti taustaprosessina Visual Studio -kehitysympäristön käynnistyessä. Tämä rajapinta toteuttaa *VsTextViewCreated*-funktion.

VsTextViewCreated-funktio käynnistää *IOleCommandTarget*-rajapinnan toteuttavat funktiot. Nämä funktiot käsittelevät Visual Studio 2019 -kehitysympäristössä käsiteltävän XML-tiedostoon annettuja komentoja (Kuvio 5). *IOleCommandTarget*-rajapinnan toteuttavassa luokassa käynnistetään *ICompletionSession*-rajapinnalle uusi istunto, jota *ICompletionSource*-rajapinnan funktiot käsittelevät.

```
public int Exec(ref Guid pguidCmdGroup, uint commandID, uint nCmdexecopt, IntPtr pvaIn, IntPtr pvaOut)
{
    if (VsShellUtilities.IsInAutomationFunction(m_provider.ServiceProvider))
    {
        return m_nextCommandHandler.Exec(ref pguidCmdGroup, commandID, nCmdexecopt, pvaIn, pvaOut);
    }

    var typedChar = char.MinValue;
    if (pguidCmdGroup == VSConstants.VSStd2K && commandID == (uint)VSConstants.VSStd2KCmdID.TYPECHAR)
    {
        typedChar = (char)(ushort)Marshal.GetObjectForNativeVariant(pvaIn);
    }
    else if (IsAccept(commandID))
    {
        if (IsCompletionSessionActive())
        {
            if (CommitSession())
            {
                return VSConstants.S_OK;
            }
        }
    }

    bool handled = false;
    int retVal = m_nextCommandHandler.Exec(ref pguidCmdGroup, commandID, nCmdexecopt, pvaIn, pvaOut);

    if (IsTypedChar(typedChar))
    {
        if (IsCompletionSessionActive())
        {
            m_completionSession.Filter();
        }
        else
        {
            TriggerCompletion();
        }
        handled = true;
    }

    return handled ? VSConstants.S_OK : retVal;
}
```

KUVIO 5. Näyte *IOleCommandTarget*-rajapinnan käskyjä käsittelevästä funktiosta

Mallin rajapinnat

ICompletionSession-rajapinta sisältää Visual Studio -kehitysympäristössä olevan aktiivisen tiedoston, sekä tiedot annettavista täydennyksistä. *ICompletionSession*-rajapinta toteuttaa *IIntellisenseSession*-rajapinnan. Täydennykset saavat sisällensä datan *ICompletionSourceProvider*-rajapinnan funktioiden avulla (Kuvio 6).

ICompletionSourceProvider-rajapinnan funktiot käynnistyvät, kun *IOleCommandTarget*-rajapinnan toteuttavasta luokasta kutsutaan *ICompletionSession*-rajapinnan *Commit*-funktiota. *ICompletionSourceProvider*-rajapinnan *TryCreateCompletionSource*-funktio käynnistää *ICompletionSource*-rajapinnan *AugmentCompletionSession*-funktion.

AugmentedCompletionSession-funktio on vastuussa *CompletionSet*-olion täyttämisestä. *CompletionSet*-olio sisältää käynnistetyn istunnon *Completion*-oliot eli täyttöehdotukset.

```
[Export(typeof(ICompletionSourceProvider))]
[ContentType("XML")]
[Order(After = "default")]
[Name("XML Intellisense Extension")]
internal sealed class CompletionSourceProvider : ICompletionSourceProvider
{
    [Import]
    internal ITextStructureNavigatorSelectorService NavigatorService { get; set; }
    public ICompletionSource TryCreateCompletionSource(ITextBuffer textBuffer)
    {
        return new CompletionSource(this, textBuffer);
    }
}
```

KUVIO 6. Näyte *ICompletionSourceProvider*-rajapinnan toteuttavasta luokasta

Näkymän rajapinnat

ICompletionSource-rajapinta antaa *ICompletionSession*-rajapinnalle CompletionsSet-olion, joka käynnistää *IIntellisensePresenterProvider*-rajapinnan funktiot.

IIntellisensePresenterProvider-rajapinta käynnistää *TryCreateIntellisensePresenter*-funktion (Kuvio 7).

```
[Export(typeof(IIntellisensePresenterProvider))]
[ContentType("XML")]
[Order(After = "default")]
[Name("XML Intellisense Extension")]
public class IntellisensePresenterProvider : IIntellisensePresenterProvider
{
    public IIntellisensePresenter TryCreateCompletionSource(IIntellisenseSession session)
    {
        return new IIntellisensePresenterControl(session as ICompletionSession);
    }

}
```

KUVIO 7. Näyte IIntellisensePresenterProvider-rajapinnan toteuttavasta luokasta

TryCreateIntellisensePresenter-funktiossa käynnistetään luokka, joka toteuttaa *IIntellisensePresenter*-rajapinnan. Tässä projektissa vaaditaan näkymän olevan ponnahdusikkuna. Tähän käyttötarkoitukseen löytyy *IPopupIntellisensePresenter*-rajapinta, joka toteuttaa *IIntellisensePresenter*-rajapinnan. Tämä rajapinta vastaa näkymän luonnista.

Työkaluvinkin rajapinnat

IAsyncQuickInfoSourceProvider-rajapinta toteuttaa *TryQuickInfoSource*-funktion. Rajapinta kuuntelee hiiren osoittimen sijaintia Visual Studio 2019 aktiivisessa XML-tiedostossa.

IAsyncQuickInfoSource-rajapinta toteuttaa *GetQuickInfoItemAsync*-funktion. Tämä funktio vastaa työkaluvinkin tiedon hakemisesta ja rakentamisesta.

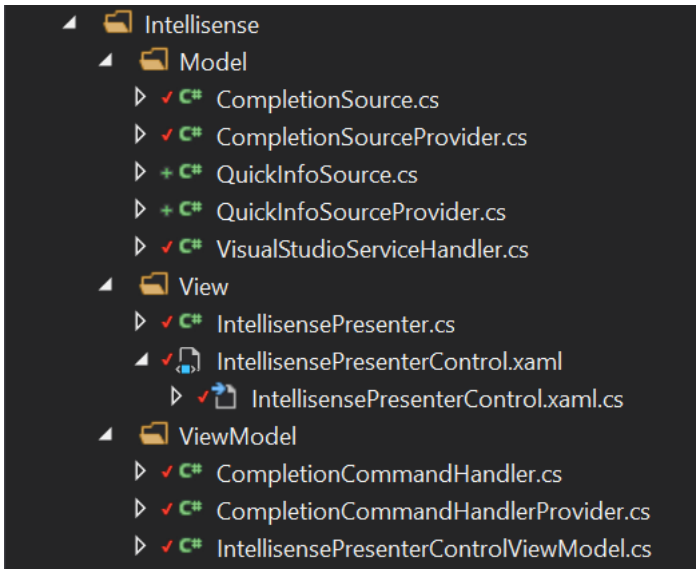
4.2.3 MVVM-suunnittelumallin toteutus

Visual Studio -laajennus projektipohjan rakentamisen jälkeen toteutetaan tarvittavat rajapinnat. Microsoft.VisualStudio -kirjaston rajapintojen toteuttamat luokat asetetaan projektirakenteessa omaan kansioonsa. Tämä kansio sisältää

mallin, näkymän ja näkymämallin kansiot (Kuva 1). Näihin kansioihin jaetaan luokat toimintansa perusteella.

Malli-kansion alle asetetaan kaikki datan keräävä ja säilövä luokka. Rajapinnat, jotka vastaavat tiedonkeruusta ovat *ICompletionSource* ja *ICompletionSourceProvider*. Näkymä-kansion alle asetetaan näkymä ja näkymän logiikka. Näkymän luonnista vastaava *IIntellisensePresenterProvider*-rajapinnan toteuttava luokka asetetaan myös näkymästä vastaavaan kansioon. Näkymämallin kansioon asetetaan kaikki ominaisuuksia muokkaavat ja käyttäjän syötteitä käsittelevät luokat. *IVsTextViewCreationListener*- ja *IOleCommandTarget*-rajapinnat käsittelevät Visual Studio -kehitysympäristössä tapahtuvia komentoja, joten niiden toteuttavat luokat kuuluvat näkymämalliin.

Työkaluvinkin esittäminen tapahtuu taustaprosessista, eikä näytettävää työkaluvinkin ponnahdusikkunaa voida muokata. Työkaluvinkin esittämän datan haku ei kuulu näkymän puolelle. Näistä syistä työkaluvinkin luokat luokitellaan malliin kuuluvaksi.



KUVA 1. Näyte MVVM-suunnittelumallin rakenteesta projektissa

Ominaisuusolio

Näkymämalli periytyy *INotifyPropertyChanged*-rajapinnasta, joka tekee näkymämallista ominaisuusolion (Kuvio 8). Täten näkymämalli pystyy lähettämään ominaisuuksissa tapahtuvista muutoksista tiedonantoja näkymälle. MVVM-suunnittelumallin mukaisesti, myös mallista voi tehdä ominaisuusolion (Vice & Siddiqi 2012).

Tämän projektin toteutuksen kannalta mallista ei tarvitse tehdä ominaisuusoliota. Tämän projektin mallissa pidettävä ja haettava data ei muutu näkymän istunnon aikana. Tästä syystä mallin ei tarvitse antaa tiedonantoja näkymälle. Näkymämalli on vastuussa näkymän muutoksista kuten täydennyksien suodatuksesta.

```
public class CompletionTypeFilter : INotifyPropertyChanged
{
    private string m_completionFilterType;
    public string CompletionFilterType
    {
        get { return m_completionFilterType; }
        set
        {
            if (value != this.m_completionFilterType)
            {
                this.m_completionFilterType = value;
                NotifyPropertyChanged();
            }
        }
    }
}
```

KUVIO 8. Näyte suodatuksen ominaisuusoliosta

Näkymä ja sidokset

Näkymän ulkoasu rakennetaan useasti käyttäen WPF:ää tai SilverLightia. Tämän projektin toteutukseen käytettiin WPF:ää, joka tarjoaa näkymän ulkoasun määrittämisen XAML-tiedostoon. Tämä tiedosto rakennetaan elementeistä, joilla kuvataan komponentin ulkoasu ja määritetään sen toiminnallisuus. Elementin sisältäessä muuttuvia ominaisuuksia, ne asetetaan tiedostoon datasidonnan avulla (Kuvio 9).

```
<Image HorizontalAlignment="Center"
       VerticalAlignment="Center"
       Width="13"
       Height="13"
       Source="{Binding AllFilterImage,
         Mode="OneWay",
         UpdateSourceTrigger="PropertyChanged",
         NotifyOnSourceUpdated="True"}"/>
```

KUVIO 9. Näyte datasidonnasta XAML-tiedostossa

Käyttäjän syötteen antaminen näkymältä näkymämalliin on yksi MVVM-mallin haastavimmista osista. Tämä korostuu virheettömässä MVVM-mallissa, jolloin näkymän logiikka ei sisällä koodia. Käskyjen antaminen tapahtuu periyttämällä luokka *ICommand*-rajapinnalla. Näkymästä voi lähettää syötettä näkymämallille käyttämällä syötesidoksia (engl. input binding). Nämä sidokset mahdollistavat syötteen liittämisen käskyyn. Syötesidoksia on kahdenlaisia: näppäinsidoksia ja hiirisidoksia. (Vice & Siddiqi 2012.)

Käyttäytymisliitos (engl. attached behavior) -malli mahdollistaa syötteen sitomisen käyttöliittymän komponentteihin. Käyttäytymisliitos -malli vaihtaa ominaisuusolion ominaisuuden takaisinkutsufunktiota. Tätä funktiota kutsutaan aina, kun ominaisuus muuttuu, jolloin siihen voidaan lisätä tapahtumankäsittelijä. Näin päästään käsiksi ominaisuuden riippuvuusolioon eli näkymän komponenttiin, jolle voidaan asettaa syötesidos. (Vice & Siddiqi 2012.)

Malli

Kuten mallin esittelyssä mainittiin, sen toteutukseen on useita eri tapoja. Mallille ei ole väliä mistä se datansa saa. Mallin tarkoitus on vain hakea ja varastoida data, joka lähetetään näkymämallille muokattavaksi.

Tämän projektin toteutuksessa malli ei ollut tietoinen olevansa osana MVVM-mallin toteutusta, eli mallista ei tehty ominaisuusoliota. Projektin malli käynnistetään instanssina, joka hakee ja varastoi datan yrityksen sisäisistä tiedostoista. In-

stanssi tarkoittaa, että malli luodaan vain kerran. Mallin ei siis tarvitse hakea dataa uudestaan jokaisen istunnon alkaessa. Laajennuksen istunto hakee datan mallilta, joka annetaan näkymämallille muokattavaksi esitysmuotoon.

4.2.4 Laajennuksen asennus

Visual Studio on tehnyt kehitettyjen laajennusten asentamisen helpoksi ja vaivattomaksi. Visual Studion rakentaessa VSIX-projektin, luodaan automaattisesti asennustiedosto projektin sisälle.

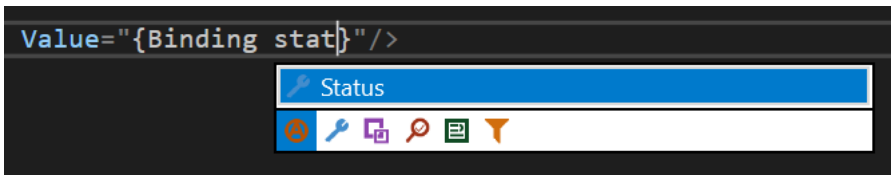
Visual Studion sisäinen VSIX Installer -ohjelma käynnistyy avaamalla luotu asennustiedosto. Ohjelman ikkunasta voi valita asennustiedostot, jotka halutaan asentaa, jolloin VSIX Installer hoitaa loput. Laajennus on heti käyttövalmis, kun Visual Studio -kehitysympäristö avataan asennuksen jälkeen.

4.2.5 Visual Studio -laajennuksen käyttö

Lopullinen laajennus toimii kaikissa rajapintojen luokka-attribuuttiin merkityissä tiedostoissa. Käyttäjän syötteestä tarkistetaan, vaaditaanko älykästä koodin täyttöä. Syötteen ollessa laajennuksen tarjoamassa kontekstissa malli luo olion, joka sisältää kaikki kontekstiin sopivat täyttöehdotukset.

Täyttöehdotuksille määritetään ehdotuksen tyyppi ja täyttötiedot. Nämä täyttöehdotukset annetaan näkymämallille, joka muokkaa ne näytettävään muotoon. Näkymä näyttää listan kaikki täyttöehdotukset.

Käyttäjän syöttäessä lisää merkkejä tarkistetaan, onko syötetty merkki vielä täyttöehdotuksen kontekstissa, vai kuuluuko istunnon loppua. Istunnon ollessa aktiivinen näkymämalli suodattaa täyttöehdotukset (Kuva 2). Käyttäjä voi myös suodattaa täyttöehdotuksia sen tyypin perusteella. Käyttäjä voi valita listasta haluamansa ehdotuksen ja hyväksyä automaattisen täytön.



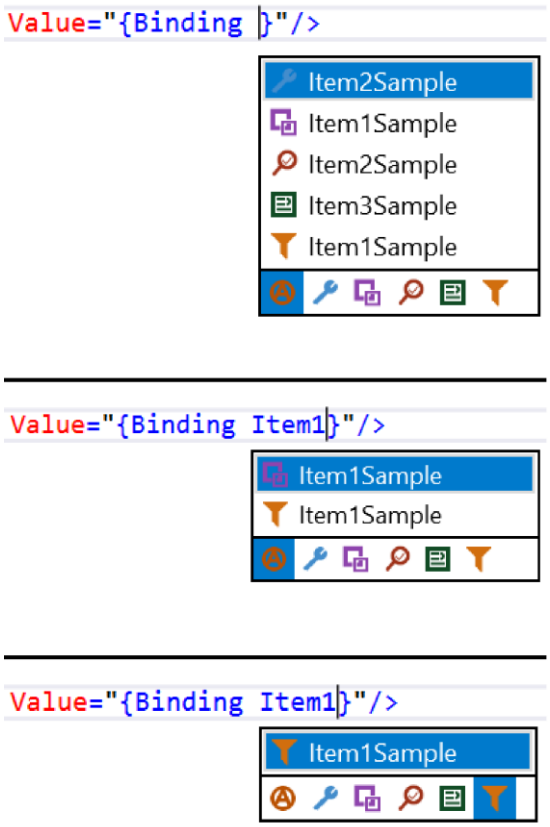
KUVA 2. Laajennus käytössä Visual Studio 2019 -kehitysympäristössä

4.3 Yhteenveto

Visual Studio -laajennusprojektin aloittamiseen löytyy paljon tietoa Microsoftin dokumentaatiosta. Rajapintoja on tarjolla yleisimpiin laajennuksiin, joita projekteissa kannattaa hyödyntää. Rajapintojen hyöty ja käyttötarkoitus on selvästi kuvattu niiden dokumentaatiossa. Esimerkit rajapintojen käytöstä projekteissa on kuitenkin hyvin vähäistä.

MVVM-suunnittelumallin käyttö useassa WPF-pohjaisessa projektissa on hyödyllistä, vaikka pakollista koodia joutuu kirjoittamaan enemmän. MVVM-malli mahdollistaa osien uudelleen käyttöä ja parantaa testattavuutta. Tosin virheettömän MVVM-mallin toteuttaminen on hyvin työlästä ja vaatii paljon ominaisuusolioiden muokkaamista.

Valmistetun Visual Studio -laajennuksen kehitys oli kuitenkin ongelmatonta, kun ymmärsi rajapintojen käyttötarkoituksen. Laajennuksen asennustiedoston luonti projektista ja sen käyttöönotto on yksinkertaista ja nopeaa VSIX:än avulla. Toteutettu laajennus toimii tarkoituksenmukaisesti. Laajennus tehostaa ohjelmakoodin kirjoitusta tarjoamalla täyttöehdotuksia, jotka voi suodattaa tarkemmin tyyppinsä tai kirjoitetun tekstin perusteella (Kuva 3).



KUVA 3. Näyte Visual Studio -laajennuksen täyttöehdotusten suodatuksen toiminnasta

Projektin Visual Studio -laajennukseen voi lisätä toiminnallisuutta vaivattomasti. *IOleCommandTarget*-rajapinnan toteuttavaan luokkaan lisätään ehto, milloin istunnon kuuluu alkaa, sekä *ICompletionSource*-rajapinnan toteuttavaan luokkaan lisätään haluttu täyttöehdotus. Samat toiminnot lisätään *IAsyncQuickInfoSource*-rajapinnan toteuttavaan luokkaan, jolloin laajennus osaa käynnistää työkaluvinkin ponnahdusikkunan oikeassa kontekstissa.

5 POHDINTA

Visual Studio -kehitysympäristön käyttö yleistyy ja uusia käyttötarkoituksia tulee esille päivittäin. Microsoft ei kaikkia tarvittavia ominaisuuksia pysty tunnistamaan, mutta laajennettavuutensa vuoksi kuka tahansa voi kehittää uusia lisäosia ja laajennuksia Visual Studio -kehitysympäristöön. Tämä opinnäytetyö korostaa laajennusprojektien rakentamista oikealla suunnittelumallilla, sekä rajapintojen valinnoilla. Microsoftin tarjoamista kirjastoista löytyy useita valmiita rajapintoja laajennuksien kehittämiseen, mutta niiden toteutuksesta ei välttämättä löydy ohjeita tai esimerkkejä. MVVM-mallista löytyy paljon tietoa kirjoista sekä internetistä, mutta tiedot lähteiden välillä saattavat olla ristiriidassa. Microsoftin ei ole antanut virallista dokumentaatiota tai ohjetta MVVM-mallin valmistukseen, joka tekee sen virheettömästä toteutuksesta hankalaa.

Tämän opinnäytetyön tarkoitus oli selventää Visual Studio -kehitysympäristön laajennusten kehitysprosessia käyttäen MVVM-mallia, sekä Microsoftin kirjastojen tarjoamia rajapintoja. Kaikkiin opinnäytetyön tavoitteisiin päästiin. Opinnäytetyön teoreettinen osa antoi vastauksen suunnittelumallin sekä rajapintojen valintaan. Teknisessä toteutuksessa valmistettiin toimiva Visual Studio -laajennus, joka automatisoi käyttäjälle täyttöehdotukset.

Seuraava vaihe teknisessä toteutuksessa olisi täyttöehdotuksien lisääminen useampaan kontekstiin. Projektin laajennettavuuden ansiosta tämä onnistuu yksinkertaisesti. Laajennuksen toimintaa ei tarvitse muuttaa. Käynnistysehtoihin lisätään vain uusi ehto sekä tarvittaessa malliin lisätään uutta tarjottavaa dataa. Uusien toimintojen lisääminen vaatii lisää rajapintojen tutkimista ja toteutusta. MVVM-suunnittelumallin ansiosta uusien toimintojen lisääminen on vaivatonta.

Opinnäytetyön teknisessä toteutuksessa yksi eniten aikaa vievin osuus oli mallissa tapahtuva datan haku ja sen oikeaan muotoon asettaminen. Opinnäytetyöstä tämä osa jätettiin raportoimatta, sillä datan sisältö on yrityksen sisäistä tietoturvattua dataa. Yrityksen tietoturvatut osat vaikeuttivat kehitysprosessin kulun selventämistä. Kehitysprosessia olisi pystynyt laajentamaan suuremmassa mittakaavassa, jos kaikki työhön käytetty data olisi julkista. Toinen eniten aikaa vievä

osuus oli rajapintojen toteutus. Rajapintojen toteutusta hidasti niiden vähäiset esimerkit ja ohjeet. Näkymän luonti tapahtui vaivattomasti WPF:n avulla.

Microsoftin kuuluisi antaa virallinen MVVM-mallin määritelmä ja toteutusohjeet, sekä antaa rajapintojen toteutuksista esimerkkejä. Omasta ohjelmistosuunnittelijan näkökulmasta nämä tehostaisivat kirjastojen käytettävyyttä, sekä MVVM-mallin suunnittelua ja toteutusta.

On hyvinkin mahdollista, että Microsoftin kirjastojen rajapintoihin lisätään esimerkkejä niiden käytöstä eri projekteissa Visual Studio -kehitysympäristön laajennusten lisääntyessä. WPF-sovellusten kehittäminen MVVM-mallin avulla on saanut suurta suosiota, jonka seurauksena Microsoft todennäköisesti julkistaa virallisen MVVM-suunnittelumallin ohjeet tulevaisuudessa.

LÄHTEET

Fukizi, K. Y. & de Oliveira, J. & Bruchet, M. 2019. Learn ASP.NET Core 3: Develop Modern Web Applications with ASP.NET Core 3, Visual Studio 2019, and Azure, 2nd Edition. Birmingham: Packt Publishing, Limited.

Gossman J. Advantages and disadvantages of M-V-VM. Julkaistu 3.4.2006. Luettu 16.4.2021.
https://docs.microsoft.com/en-us/archive/blogs/johngossman/advantages-and-disadvantages-of-m-v-vm

Microsoft. Visual Studio. Sovelluksen virallinen sivusto. Luettu 15.3.2021.
https://visualstudio.microsoft.com/vs/

Nathan A. 2013. WPF 4.5 Unleashed. Sams.

Smith J. Patterns – WPF Apps With The Model-View-ViewModel Design Pattern. Digitaalinen artikkeli. Julkaistu helmikuussa 2009. Luettu 25.4.2021 https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern

Sterne J. Plug-in. Digitaalinen artikkeli. Julkaistu 6.6.2014. Luettu 15.4.2021. https://www.britannica.com/technology/plug-in

Vice R. & Siddiqi, M. S. 2012. MVVM survival guide for enterprise architecturs in Silverlight and WPF eliminate unnecessary code by taking advantage of the MVVM pattern – less code, fewer bugs. Birmingham; Mumbai: Packt Publishing.

W3C. Extension Markup Language (XML) (Fifth Edition). Julkaistu 25.9.2007. Luettu 2.4.2021
https://www.w3.org/TR/REC-xml/