



Jere Salmensaari

Graafisen C#-ohjelmiston suoritus Linux-ympäristössä WINE-käännöskerroksella

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

30.5.2023

Tiivistelmä

Tekijä: Jere Salmensaari
Otsikko: Graafisen C#-ohjelmiston suoritus Linux-ympäristössä
WINE-käännöskerroksella
Sivumäärä: 29 sivua
Aika: 30.5.2023

Tutkinto: Insinööri (AMK)
Tutkinto-ohjelma: Tieto- ja viestintätekniikka
Ammatillinen pääaine: Ohjelmistotuotanto
Ohjaajat: Tekninen arkkitehti Harri Ruuskanen
Lehtori Ilpo Kuivanen

Insinöörityössä tutkitaan Nokian käyttämän Base Tranceiver Station -ohjelmistojen testityökalun Endeavourin kääntämistä Linux-käyttöjärjestelmälle.

Endeavour on tehty käyttäen Microsoft Windows -rajapintoja ja -käyttöliittymiä, eikä sitä pysty suoraan suorittamaan Linuxilla. Työhön valikoitiin Avalonia UI -käyttöliittymäkirjasto sekä WINE-käännöskerros tutkittaviksi vaihtoehtoiksi, joista päädyttiin valitsemaan WINE työn painopisteeksi.

Työstä ei saatu aikaan toimivaa prototyyppiä useiden ratkaisematta jääneiden ongelmien takia, mutta työn tulosten perusteella pystytään toteamaan käännöksen olevan mahdollinen WINEn kautta. Lisäksi työstä saatiin arvokasta tietoa käännöksen vaativasta työmäärästä ja parhaista vaihtoehtoista tulevaisuutta varten.

Avainsanat: c#, csharp, linux, wine, dotnet

Abstract

Author:	Jere Salmensaari
Title:	Running Graphical C# Application in Linux through WINE Translation Layer
Number of Pages:	29 pages
Date:	30 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Software Engineering
Supervisors:	Harri Ruuskanen, Technical architect Ilpo Kuivanen, Senior Lecturer

The purpose of this thesis is to investigate the possibility of creating a Linux version of the Base Tranceiver Station software testing tool Endeavour used by Nokia.

Endeavour is created using Windows application programming interfaces, and as such cannot be run under Linux. Avalonia UI user-interface library and WINE translation layer were picked for the focus of this thesis, with WINE being the primary focus.

No working prototypes were created as the outcome thesis due to multiple problems risen during the development. However, a Linux version through WINE should be possible, based on the study conducted. Furthermore, important information of the amount of work needed for a proper, working Linux version was gained.

Keywords: c#, csharp, linux, wine, dotnet

Sisällys

Sanasto ja lyhenteet

1	Johdanto	1
2	Ongelmanasettelu	2
2.1	Mikä on Endeavour?	2
2.2	Miksi käännöstyö pitää tehdä?	8
3	Vaihtoehdot	9
3.1	Avalonia UI	10
3.2	WINE	11
4	Toteutus	14
4.1	AloitUS	14
4.2	Työkalut	15
4.3	Prosessi	16
4.4	Testaus	23
4.5	Ongelmat	24
5	Yhteenveto	27
	Lähteet	28

Sanasto ja lyhenteet

.NET:	Microsoftin luoma avoimen lähdekoodin ohjelmointikehys, joka perustuu suljetun lähdekoodin .NET Framework -kehykseen.
.NET Framework:	Windows-alustalle luotu Microsoftin omistama suljetun lähdekoodin ohjelmointikehys.
APM:	Asynchronous Programming Model (Asynkroninen ohjelmointimalli).
Avalonia UI:	Järjestelmäriippumaton käyttöliittymäkirjasto .NET-ohjelmille.
Avalonia XPF:	Avalonia UI -yhteensopiva suljetun lähdekoodin kirjasto, joka toteuttaa WPF-kirjaston rajapinnan.
BTS:	Base Tranceiver Station (Radiolähettimen tukiasema). Verkkoinfrastruktuurin osa, joka mahdollistaa kommunikoinnin käyttäjän laitteen ja verkon välillä.
CI:	Continuous Integration (Jatkuva Intgeraatio).
DirectX:	Microsoftin luoma multimedian käsittelyyn tehty ohjelmointirajapinta. DirectX on laajasti käytössä esimerkiksi pelikehityksessä Windows-alustalle.
DLL:	Dynamic Link Library (Dynaamisesti linkittyvä kirjasto).
emulaattori:	Ohjelmisto, joka on suunniteltu käyttäytymään toisen järjestelmän tavalla.
Endeavour:	Nokian viestipohjainen SCT-testityökalu Base Tranceiver Station (BTS) (<i>Radiolähettimen tukiasema</i>) -ohjelmistokomponenteille.
Linux:	Linus Torvaldsin 1991 luomaan, UNIX-pohjaiseen, Linux-käyttöjärjestelmäytimeen pohjaavien käyttöjärjestelmien perhe.
macOS:	Apple Inc:n luoma UNIX-pohjainen käyttöjärjestelmä.

MIT-lisenssi:	Massachusetts Institute of Technology -lisenssi. Yksi ensimmäisistä avoimen lähdekoodin lisensseistä, joka sallii ohjelmiston käyttämisen, muokkaamisen, jakamisen sekä myymisen, jos johdannaiset ohjelmistot ovat lisensoitu saman lisenssin alla.
Mono:	Avoimen lähdekoodin toteutus .NET-rajapinnalle.
NSN Oy:	Nokia Solutions and Networks Oy.
PoC:	Proof of Concept (Konseptin todistus). Kehityksen vaihe, jossa on tarkoitus tutkia onko kehittäminen kannattavaa.
POSIX:	Portable Operating System Interface (Siirrettävä käyttöjärjestelmärajapinta).
RAM:	Random Access Memory (Hajasaantimuisti).
SCT:	System Component Testing (Järjestelmäkomponenttitestaus). Ohjelmistotestaustyyppi, jossa testataan ohjelmistokomponentin toimintaa simuloidussa ympäristössä simuloimalla vastakkaiset prosessit ilman integraatiota.
TAP:	Task-based Asynchronous Pattern (Tehtäväpohjainen asynkroninen malli).
TPL:	Task Parallel Library (Tehtävä-Rinnakkais-kirjasto).
UML:	Unified Modeling Language (Yhdistetty mallinnuskieli).
UNIX:	Bell Labsin luoma käyttöjärjestelmien perhe.
virtuaalikone:	Ohjelmallisesti toteutettu tietokone, jossa tietokoneen komponentit on toteutettu <i>virtualisoimalla</i> .
Windows:	Microsoft Corporationin luoma graafisten käyttöliittymien ja käyttöjärjestelmien perhe.

- WINE:** WINE Is Not an Emulator (WINE ei ole emulaattori).
Yhteensopivuuskerros Windows-ohjelmien suorittamiseen useilla POSIX-yhteensopivilla käyttöjärjestelmillä kuten Linuxilla.
- WinForms:** Windows Forms.
.NET Frameworkiin sisältyvä Windows-alustan avoimen lähdekoodin käyttöliittymäkirjasto.
- WPF:** Windows Presentation Foundation.
.NET Frameworkiin sisältyvä Windows-alustan avoimen lähdekoodin käyttöliittymäkirjasto, joka hyödyntää DirectX-rajapintaa käyttöliittymän piirtämiseen.
- XML:** Extensible Markup Language (Laajennettava merkintäkieli).

1 Johdanto

Insinöörityön tarkoituksena on tutkia Windows-alustalle tehdyn Endeavour-työpöytäsovelluksen käyttöönoton mahdollisuutta Linux-käyttöjärjestelmille. Työtä varten on tutkittu kahta vaihtoehtoa: Avalonia UI ja WINE (WINE Is Not an Emulator) (*WINE ei ole emulaattori*), joilla voisi kokeilla tehdä Proof of Concept (PoC) (*Konseptin todistus*) -versiota Linux versiosta. Endeavour-sovellus on *Nokia Solutions and Networks Oy (NSN Oy)*:n käyttämä testityökalu, joka on rakennettu Windows-käyttöjärjestelmälle käyttäen C# -ohjelmointikieltä sekä .NET Framework -ohjelmointikehystä.

Luvussa 2 tutustutaan tarkemmin syihin, joista tätä työtä halutaan tehdä, sekä käydään syvemmin läpi Endeavour-työkalun toimintaa sekä historiaa. Luvussa 3 tutustutaan mahdollisiin käännökseen käytettäviin työkaluihin, ja tekniikkoihin ja käydään läpi niiden hyviä ja huonoja puolia. Luku 4 käy läpi työn toteutusta, käytettyjä työkaluja ja testimenetelmiä. Lisäksi tutustutaan erinäisiin ongelmiin, joita työn aikana ilmeni. Lopuksi luvussa 5 käydään läpi lopputulokset sekä pohditaan, miten työn aihetta jatketaan tulevaisuudessa.

2 Ongelmanasettelu

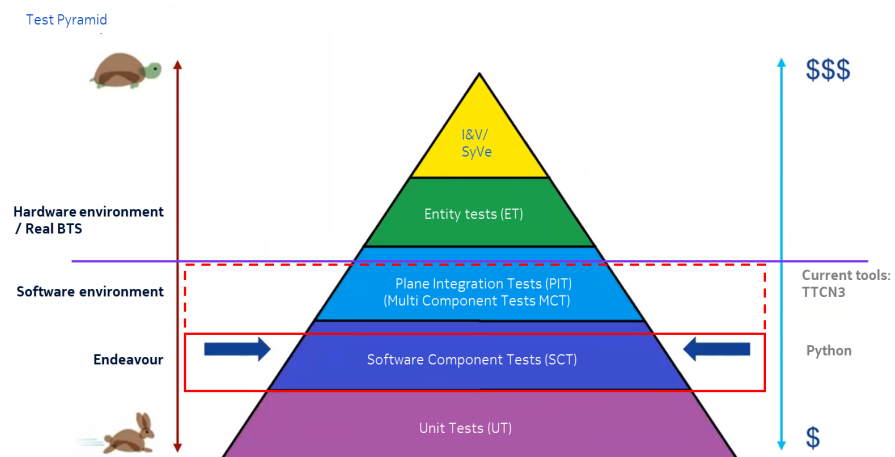
Työssä pyritään tuottamaan Endeavour-työkalusta jollakin tasolla toimiva Linux-PoC-versio. Koska kyseessä ei ole tuotantosovellus, eikä käyttäjiä Linux-alustalla vielä ole, tuotoksen ei tarvitse olla täydellinen tai ilman bugeja. Yritys kuitenkin olisi tuottaa Endeavour-Linux-versiolle vähintään *Bash*-asennusscripti, joka luo tarvittavat kansiot sekä siirtää sovelluksen oikealle paikalle, jotta käyttäjä voisi suorittaa asennusscriptin ja alkaa suoraan käyttämään työkalua. Mahdollisesti ajan salliessa voitaisiin myös tuottaa *rpm*-asennuspaketti *Red Hat* -pohjaisille Linux-käyttöjärjestelmille.

2.1 Mikä on Endeavour?

Endeavour on vuonna 2011 luotu Nokian työkalu radioverkon tukiasemien (Base Transceiver Station) (BTS) -ohjelmistolle, jonka tarkoitus oli tulla korvaamaan vanhempaa saman kaltaista System Component Testing (SCT) (*Järjestelmäkomponenttitestaus*) -työkalua, joka on ollut Nokian käytössä 90-luvun loppupuolelta. Kuvassa 1 on kuvattu Nokian BTS-ohjelmistokomponenttien testipyramidi sekä käytössä olevat työkalut, ja mihin kohtaa pyramidia ne sijoittuvat.

Endeavour sijoittuu pyramidissa pääasiassa SCT-testauksen paikoille. SCT-testauksessa testataan ohjelmistokomponenttien toimintaa simuloitussa ympäristössä simuloitujen vastakappaleiden kanssa. Endeavour on ollut käytössä SCT-testauksessa vuodesta 2012.

Uusi Endeavour-PoC-projekti aloitettiin vuoden 2022 alussa. PoC-vaiheen tarkoituksena oli tutkia mahdollisuutta tuoda Endeavour Nokian uusille ohjelmistoille korvaamaan käytössä ollut Python-pohjainen testikirjasto, joka oli koettu hitaaksi ja kankeaksi. Kirjoittamisen aikana Endeavour on *pilot*-vaiheessa, eli se on suppeassa kokeilukäytössä.



Kuva 1: Testipyramidi, johon on merkitty Endeavourin osa-alueet sekä muiden Nokialla BTS-komponenttien testaukseen käytettyjen työkalujen osa-alueet.

Toisin kun vanha koodipohjainen testikirjasto, Endeavour on viestipohjainen testityökalu, joka tarkoittaa sitä, että Endeavour ei ilmaise testejä funktioiden kautta vaan testitapauksissa piirretään viestejä testattavalta prosessilta toiselle. Aikaisemmin tehtyjä viestejä voi myös tallentaa kirjastoon ja usein käytettyjä sanomasekvenssejä voidaan tallettaa ns. *SubMessageiksi*, eli alitestitapauksiksi, joka käyttäytyy samalla tavalla kuin python-funktiot. Nämä viestit Endeavour visualisoi kuvan 2 mukaisesti, Unified Modeling Language (Yhdistetty mallinnuskieli) (UML) -viestikaavoiden kaltaisiksi kuvaajiksi, joista on helppo seurata testien kulkua. Kuten kuvasta 2 näkee Endeavour-testitapaukset ovat suhteellisen helppoja seurata ja ymmärtää. Vaikka python-testitapaus on rivillisesti lyhyempi, sen sisäinen toteutus jää katsojalta täysin piiloon, jollei katsoja ala etsimään käytettyjen funktioiden toteutusta. Endeavour-testitapaukset koostuvat prosesseista, joilla on nimi ja osoite, viesteistä prosessien välillä, alitestitapauksista ja C#-skripteistä. Prosessit voivat olla valeprosesseja, testattavia prosesseja tai aitoja prosesseja ja niitä voi olla testissä yksi tai useampi.

```

@pytest.mark.feature('test')
@viper_execution_timeout(20.0)
def test_startup_rest_validation_only(
    sut: None, system: System, im_defaults: DefaultBehavioursHandler,
    init_fms: Callable[[], None] -> None:
    init()
    functions.activate_initial_alternatives(defaults, functions.EmptyResponse)
    activate(defaults.change_req())

    functions.handle_registration()
    functions.perform_startup_rest_validation_only(functions.EmptyResponse, defaults)

```



Kuva 2: Python-testi (ylä), ja sama testi Endeavourilla (ala). Testeistä on poistettu tunnistettavat tiedot testattavasta ohjelmistosta.

Endeavour tukee useita eri viestityyppejä, kuten JSON, XML ja bittivirta, mutta yleisin viestityyppi 5G-ympäristössä on JSON. Viestit voivat olla monitoroituja, eli Endeavour odottaa niitä testattavalta komponentilta, tai simuloituja, jotka Endeavour lähettää valeprosessilta testattavalle. Endeavour-testitapauksia voi tehdä työkalun piirto-ominaisuuden kautta, tai kirjoittaa käsin esimerkin 1 mukaisena Extensible Markup Language (Laajennettava merkintäkieli) (XML)-tiedostoformaattiin pohjaavana tiedostona.

Suuri etu Endeavourin viestipohjaisuudessa ja visuaalisuudessa on sen helppo opittavuus. Testitapaukset ilmaisevat suoraan, mitä niissä tapahtuu, ja testien tekijän ei tarvitse opetella erillistä koodikieltä ja kirjastoa testitapauksien tekemistä varten. Helppo opittavuus auttaa varsinkin uudempien työntekijöiden opettamisen kanssa, mutta myös pitempiaikaisempienkin työntekijöiden on helpompi luoda testejä spesifikaatioiden pohjalta, jotka usein ovat viestikuvaajamallisia.

```

1  <Endeavour Version="1.1">
2    <TestCase>
3      <Name>new test case</Name>
4      <PidList>
5        <BtsProcessInstance Id="3">
6          <Alias>RealProcess</Alias>
7          <ProcessInstanceType>Mut</ProcessInstanceType>
8        </BtsProcessInstance>
9        <BtsProcessInstance Id="4">
10         <Alias>Fake_1</Alias>
11         <ProcessInstanceType>Fake</ProcessInstanceType>
12        </BtsProcessInstance>
13        <BtsProcessInstance Id="5">
14         <Alias>Fake_2</Alias>
15         <ProcessInstanceType>Fake</ProcessInstanceType>
16        </BtsProcessInstance>
17      </PidList>
18      <CommandList>
19        <JsonMessage>
20          <ToBts>@4</ToBts>
21          <FromBts>@3</FromBts>
22          <Header>{
23            "Method": "POST",
24            "url": "http://test-url.net/testing"
25          }</Header>
26          <Body>{
27            "message": "This is a monitored test message"
28          }</Body>
29        </JsonMessage>
30        <JsonMessage>
31          <ToBts>@3</ToBts>
32          <FromBts>@4</FromBts>
33          <Header>{
34            "Method": "POST",
35            "url": "http://test-url.net/testing"
36          }</Header>
37          <Body>{

```

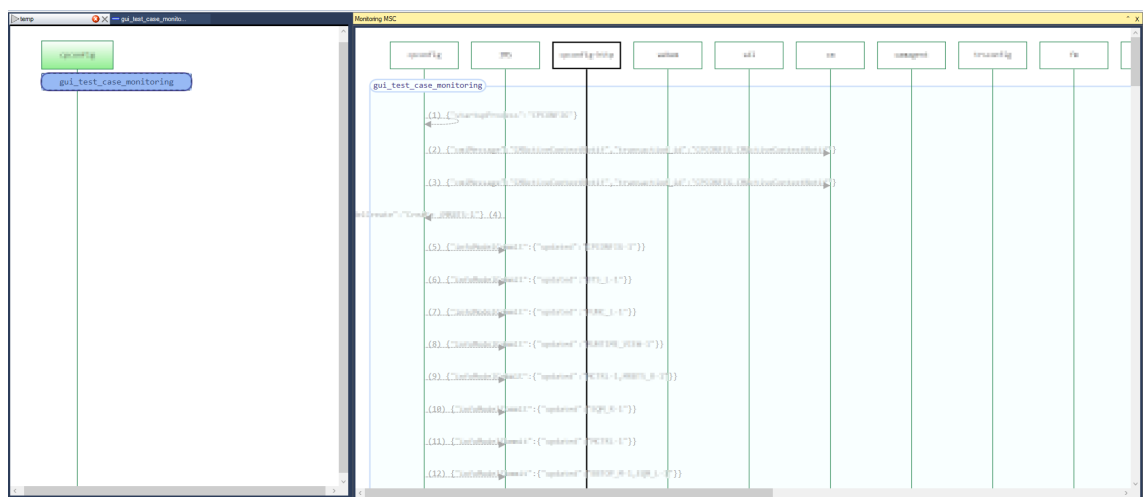
```

38         "response": "This is the response"
39     }</Body>
40 </JsonMessage>
41 <JsonMessage>
42     <ToBts>@3</ToBts>
43     <FromBts>@5</FromBts>
44     <Header>{
45         "Method": "PUT"
46     }</Header>
47     <Body>{
48         "message": "This is a message from fake process 2"
49     }</Body>
50 </JsonMessage>
51 </CommandList>
52 </TestCase>
53 </Endeavour>

```

Koodiesimerkki 1: Yksinkertainen Endeavour-testi XML -formaattissa

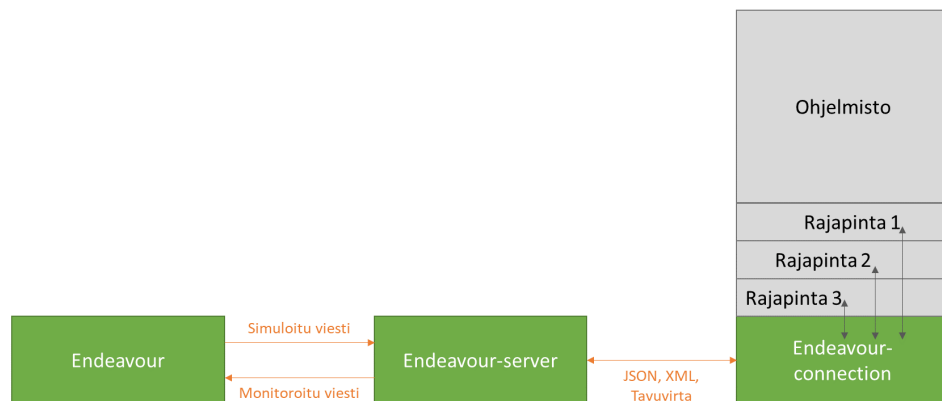
Testien epäonnistuessa Endeavour on voimakas työkalu, myös vanhan Python-pohjaisen työkalun kanssa. Endeavour-testejä suorittaessa tarkkaillaan todellisten testitapauksien viestien kulkemista, josta testaaja voi helposti nähdä, jos testattava komponentti lähettää esimerkiksi viestin väärään paikkaan. Tätä tarkkailua voi myös hyödyntää Python-testitapauksien seuraamiseen (katso kuva 3), ja kääntämiseen Endeavour-testitapauksiksi.



Kuva 3: Endeavour-monitorointi Python-testitapauksen ajosta, monitoroidut viestit vasemmalla.

Endeavour on osa laajempaa tukiasemien diagnostiikkaohjelmaa, josta se lainaa joitakin ominaisuuksia, kuten useiden bittivirtatyypin sanomien purun.

Endeavour kootaan Dynamic Link Library (Dynaamisesti linkkiytyvä kirjasto) (DLL) -kirjastona, joka on Windowsin dynaamisesti ladattava kirjastotyyppi. Lisää DLL-kirjastoista luvussa 3.2. Endeavour testityökaluna on jaettavissa kolmeen osaan (katso kuva 4): Endeavour-käyttöliittymä, Endeavour-palvelin sekä Endeavour-connection-kirjasto.



Kuva 4: Endeavour-työkalun järjestelmäarkkitehtuuri

Endeavour-palvelin

Endeavour palvelin on pythonilla tehty tcp-palvelin, jonka tarkoituksena on ohjata Endeavour-käyttöliittymältä tulevat sanomat Endeavour-connection-kirjastolle. Endeavour-palvelin myös käynnistää testattavan komponentin ja luo sille oman hiekkalaatikkoympäristön. Ilman testitapauksen ajoa palvelin ei itsessään tee mitään.

Endeavour-connection-kirjasto

Endeavour-connection on Endeavour-työkalun testattavaan komponenttiin yhdistävä kirjasto. Endeavour-connection keskustelee Endeavour-palvelimen kanssa tcp-viesteillä. Endeavour-connection upotetaan testattavaan komponenttiin korvaamaan erinäisiä rajapintoja, jolloin rajapinnan funktioita kutsuessa kutsu meneekin Endeavour-connection-kirjastolle, joka ohjaa viestin Endeavour-työkalulle, jolta se saa tarvittaessa vastauksen.

2.2 Miksi käännöstyö pitää tehdä?

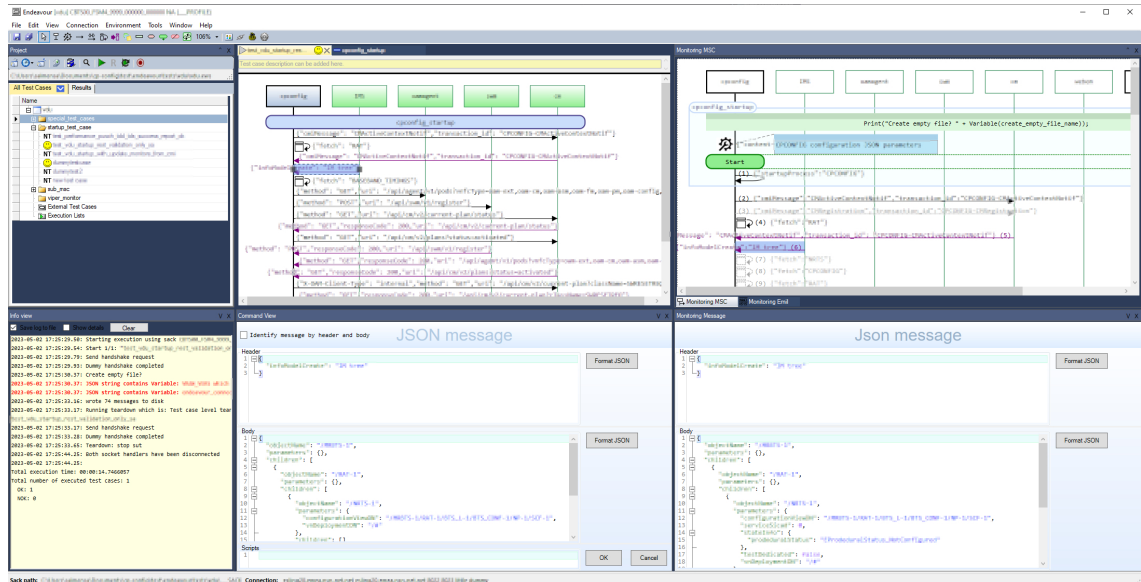
Linux ja Windows ovat molemmat pääasiallisesti x86-suoritinarkkitehtuuria hyödyntäviä käyttöjärjestelmiä. Miksi sitten toiselle koottuja ohjelmia ei voi suorittaa toisessa? Linux ja Windows käyttävät suoritettavien ohjelmien luomiseen eri muotoa. Linuxin suoritettavat ohjelmat luodaan käyttäen Executable and Linkable Format (ELF) -formaattia, kun taas Windows käyttää Portable Executable (PE) -formaattia. Tämän lisäksi Windows- ja Linux-ytimet omaavat erilaiset rajapinnat, joiden kanssa ohjelmien täytyy osata kommunikoida.[1.]

Monille Nokian työntekijöistä Endeavour voi tulevaisuudessa olla ajankohtainen työkalu. Tällä hetkellä ainoa tapa käyttää Endeavouria Linuxilla on Windows-virtuaalikoneen käyttäminen. Tämä ratkaisu on teoreettisesti toimiva, mutta ohjelmien suorittaminen virtuaalikoneen kautta on huomattavasti raskaampaa kuin ohjelman suorittaminen natiivisti, ja tämä voi hidastaa työntekoa merkittävästi. Täten näillekin ohjelmistokehittäjille haluttaisiin tarjota hyvä käyttökokemus, joka vaatii Endeavourin tukemista Linux-käyttöjärjestelmillä.

Ohjelmointikehyksenä Endeavour käyttää Microsoftin .NET Framework -kehystä. .NET Framework on Windows-alustalla työpöytä- ja verkkosovelluksille kehitetty suljetun lähdekoodin ohjelmointikehys. .NET Framework tukee useita ohjelmointikieliä, kuten C#ia, F#ia ja Visual Basicia. Endeavour, yliohjelmansa kanssa, on ohjelmoitu käyttäen C#-kieltä. Vaikkakin C# pystyy kääntymään natiivisti suoritettavaksi Linux ympäristössä, .NET Framework on kuitenkin tehty vain Windows-käyttöjärjestelmälle eikä toimi Linuxin alla.

Endeavour-käyttöliittymä pohjaa osittain sen yläohjelman luokkiin, jonka käyttöliittymä on toteutettu käyttäen Windows Forms (WinForms) -kirjastoa. WinFormsin lisäksi Endeavour käyttää myös Windows Presentation Foundation (WPF) -kirjastoa testitapauksien piirtämiseen (kuva 5 keskimäinen ikkuna ja oikeanpuoleinen ikkuna). WPF ja WinForms ovat molemmat osa .NET Framework- ja .NET-ohjelmointikehyksiä Windows-alustalla. WinForms-pohjaisia ohjelmia pystytään suorittamaan Mono-ohjelman kautta Linux-ympäristössä, mutta WPF-kirjastolle

ei ole tehty samankaltaista käännöstyötä, sillä se hyödyntää vahvasti Microsoftin DirectX-multimediarajapintaa, joka on tuettu vain Windows- ja Windows-pohjaisilla käyttöjärjestelmillä.



Kuva 5: Endeavour-käyttöliittymä, jossa näkyy testitapaus (keskimäinen ikkuna), sekä ajettujen testitapauksen monitorointi (oikea ikkuna).

3 Vaihtoehdot

Linux-käyttäjät ovat usein hyvin uskollisia valitsemalleen käyttöjärjestelmälle. Täten he ovat vuosien aikana kehittäneet useita tapoja suorittaa erilaisia Windows-käyttöjärjestelmälle kehitettyjä sovelluksia Linuxissa. Esimerkiksi aikaisemmin luvussa 2.2 mainittu Mono on yksi näistä tavoista, joilla Linux-käyttäjät tuovat Windows-pohjaisia ohjelmistoja käyttöönsä. Mono on avoimen lähdekoodin toteutus Microsoftin .NET Framework -rajapinnalle, joka mahdollistaa tälle rajapinnalle tehtyjen ohjelmien ajon Linux-ympäristössä. Endeavour on hyödyntänyt monoa Continuous Integration (Jatkuva Intgeraatio) (CI) -ajoihin, jotka tapahtuvat Linux-palvelimella, mutta Mono ei sovi Endeavourin käyttöliittymän suorittamiseen, sillä kuten aiemmin on mainittu, se ei sisällä toteutusta WPF-kirjastolle. Tätä työtä alustettaessa päädyttiin kahteen mahdolliseen vaihtoehtoon, Avalonia UI:iin ja WI-NE:iin joita lähteä tutkimaan sekä tekemään alustavaa konseptiversiota.

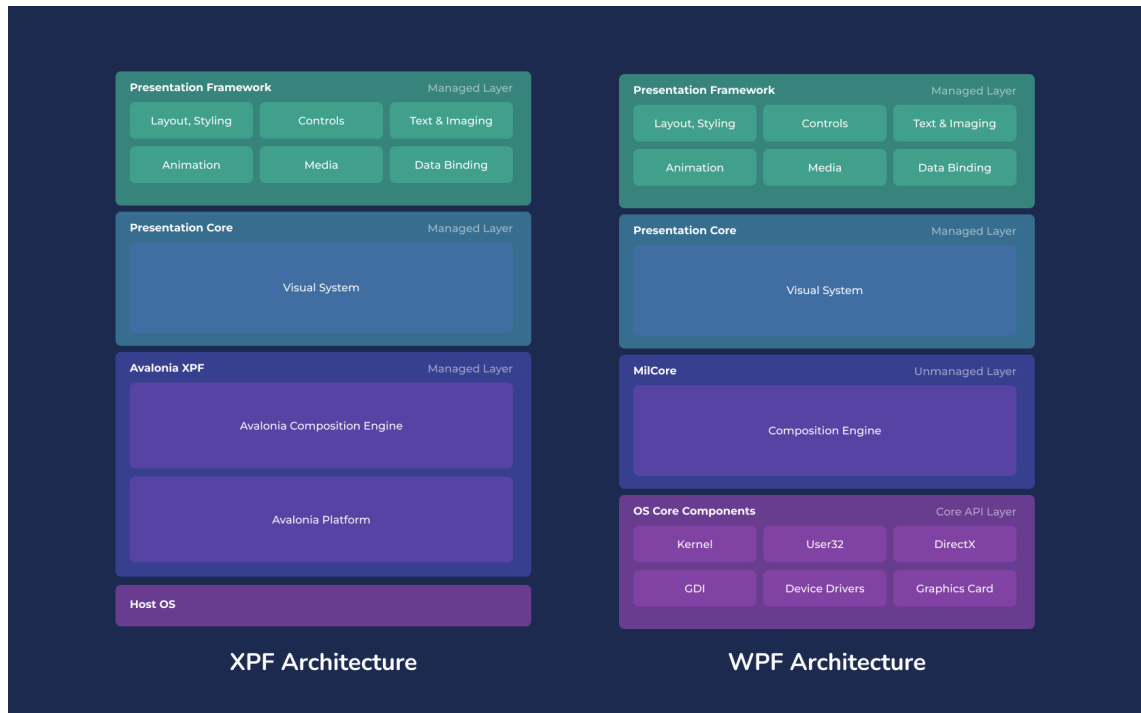
3.1 Avalonia UI

Yksi vaihtoehto Endeavourin Linux-käännökselle on Avalonia UI -käyttöliittymä-kirjasto. Avalonia on avoimen lähdekoodin järjestelmäriippumaton käyttöliittymä-kirjasto .NET-ohjelmistoille, jonka kehitys aloitettiin vuonna 2013 [2]. Avalonia UI julkaistiin käyttäen MIT-lisenssiä ja kirjaston kehitystä tehtiin yhteistoiminnallisesti kirjaston erinäisten kehittäjien tahosta vuoteen 2019 saakka, jolloin sen kehitystä varten perustettiin *AvaloniaUI OÜ* [3]. Vaikka kehitystä koordinoi nykyisin sille tarkoitettu yritys, Avalonian kehitystä on silti jatkettu myös yhteisön kanssa samalla avoimella MIT-lisenssillä. Avalonia on kehitetty moderniksi jälleen toteutukseksi WPF-kirjastolle, joka varsinkin Avalonia UI:n luomisaikoihin oli rakennettu nopeasti vanhentuvalle alustalle [4], eikä tukenut uusia .NET Framework -ominaisuuksia.

AvaloniaUI OÜ julkisti helmikuussa 2023 uuden Avalonia XPF -kirjaston, jonka tarkoituksena on mahdollistaa WPF-kirjastoa käyttävien ohjelmien helppo kääntäminen modernimmalle avalonia-alustalle ilman koko käyttöliittymäkoodipohjan uudellenrakennusta [5]. Toisin kuin Avalonia UI, Avalonia XPF on suljetun lähdekoodin kirjasto, jonka lisenssit ovat maksullisia [6]. Syyksi muutokselle lisensöinnissä *AvaloniaUI OÜ*:n toimitusjohtaja Mike James sanoi:

Olemme kahden vaiheilla. Haluamme varmistaa, että kehittäjät rakentavat edelleen [sovelluksia] Avalonialla. Emme halua kannibalisoida omaa yhteisöämme ja kannustaa rakentamaan uusia sovelluksia WPF:llä. Mutta saatamme avata lähdekoodin myöhemmin. [7.]

Avalonia XPF hyödyntää WPF-kirjaston avoimen lähdekoodin lisenssiä, sillä XPF-kirjasto on toteutettu haarauttamalla WPF-koodipohja, josta Avalonia on kuvan 6 arkkitehtuurirakenteen mukaisesti pitänyt WPF-arkkitehtuurin korkeimmat rakenteet, mutta korvannut MilCore-kokoamismoottorin Avalonia UI -kokoamismoottoriksi. Lisäksi XPF käyttää Avalonia UI -ydintä pohjimmillaan käyttöliittymän piirtämiseen Windows-ydinkomponenttien sijaan.



Kuva 6: Avalonia XPF -arkkitehtuuri (vasen) ja Microsoft WPF -arkkitehtuuri (oikea)

Kuten on selitetty luvussa 2.2 Endeavour on tehty osittain WPF-kirjastoa käyttäen. Täten Avalonia UI näyttää lupaavalta vaihtoehdolta Linux-tuen näkökulmasta. Avalonia UI -kirjaston järjestelmäriippumattomuus tarkoittaa, että sen ylläpitäminen tulisi olemaan kevyempää kuin täysin erillisen version ylläpitäminen.

3.2 WINE

WINE (WINE Is Not an Emulator) (*WINE ei ole emulaattori*) on avoimen lähdekoodin yhteensopiivuuskerros, joka kykenee suorittamaan Windows-ohjelmia useilla Portable Operating System Interface (Siirrettävä käyttöjärjestelmärajaus) (POSIX) -yhteensopivilla käyttöjärjestelmillä, kuten Linuxilla ja macOS:lla. Kehitys WINElle alkoi vuonna 1993. Muutamat tuoreen Linux-käyttöjärjestelmän käyttäjät aloittivat keskustelun Sun Microsystemsin *Wabi*-yhteensopiivuuskerroksen kaltaisen työkalun luomisesta Linuxille.[8.] WINE:n koodipohjasta noin puolet on kirjoitettu vapaaehtoisvoimin, ja toinen puoli CodeWeavers-yrityksen sponsoimana, joka myy tuettuja versioita WINEsta macOS-, Linux- ja ChromeOS-käyttöjärjestelmille [9]. Kuten nimi kertoo, se ei ole emulaattori, eli WINE ei

pyri kopioimaan Windowsin sisäistä toimintaa, vaan pyrkii kääntämään Windows-rajapintakutsut Linux-rajapintoihin sopiviksi. [10.]

Dynamic Link Library (Dynaamisesti linkkiytyvä kirjasto) (DLL), eli dynaaminen linkattu kirjasto, joiden uudelleentoteutuksista WINE koostuu, ovat Windowsin jaetun kirjastotyyppin toteutus [11]. Jaetut kirjastot ovat ohjelmointikirjastoja, jotka voivat olla usealla ohjelmalla käytössä samanaikaisesti. Toisin kuin .exe- tai .msi-päätteisiä Windows ohjelmia, DLL-kirjastoja ei voi suorittaa itsenäisesti, vaan jonkin ulkopuolisen ohjelman täytyy ladata ne muistiin, jotta niiden funktioita ja luokkia voi käyttää. Jaettujen kirjastojen, kuten DLL:n tai Linuxin .so-kirjastojen käyttö kannustaa koodin modularisointiin ja jakamiseen sekä tehostaa ohjelmien tilankäyttöä, ja täten on hyvin yleistä. Tästäkin syystä suuri osa Windows- ja Linux-käyttöjärjestelmien ja ohjelmien toiminnasta pohjaa erilaisiin DLL- tai .so-kirjastoihin. [12.] WINEn toiminta pohjaa myös näihin DLL-kirjastoihin ja Windowsin riippuvuuteen niistä.

Yksinkertainen esimerkki WINEn Windows-rajapintakutsujen käännöksestä voisi olla ohjelma, joka yrittää luoda tiedoston käyttäen Windows *CreateFileA*-rajapintakutsua (koodiesimerkki 2). [13.]

```

1  CreateFileA(
2      "C:\some_file.txt",      //lpFileName
3      GENERIC_WRITE,          //dwDesiredAccess
4      0,                      //dwShareMode
5      NULL,                   //lpSecurityAttributes
6      CREATE_ALWAYS,          //dwCreationDisposition
7      FILE_ATTRIBUTE_NORMAL,  //dwFlagsAndAttributes
8      NULL                    //hTemplateFile
9  );

```

Koodiesimerkki 2: Esimerkki Windows *CreateFileA*-rajapintakutsutusta [13]

WINE omassa tämän funktion sisältävän DLL:n toteutuksessa voi sitten napata rajapintakutsun ja kääntää sen UNIX open-rajapintakutsuksi (koodiesimerkki 3). UNIX-kutsulta voitaisiin sitten palauttaa kahva tiedostoon ohjelmalle, joka kutsui WINE DLL:ää ja ohjelma voi jatkaa toimintaansa. [13.]

```

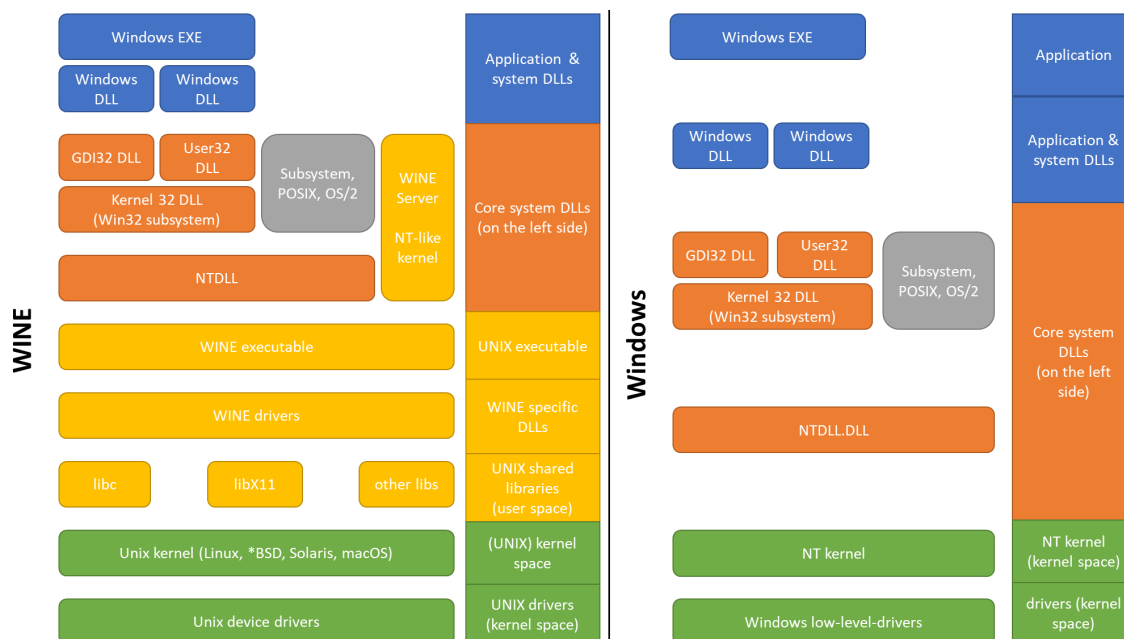
1  open(
2      "/home/salmensa/.wine/drive_c/some_file.txt",    //path
3      O_WRONLY | O_CREAT,                               //oflag
4      0644                                              //creation
5      mode
6  );

```

Koodiesimerkki 3: Esimerkki UNIX *open*-rajapintakutsusta [13]

WINE:n arkkitehtuuri on suunniteltu Windows NT -käyttöjärjestelmän arkkitehtuuria imitoiden, ja koostuu pääasiassa Windows-käyttöjärjestelmän tärkeimpiä DLL-kirjastoja korvaavista kirjastoista sekä *wineserver*-palvelimesta. Wineserver-palvelin hallinnoi WINE:n alaisten prosessien välistä kommunikaatiota, synkronointia ja prosessien sekä säikeiden hallintaa. WINE-palvelin käynnistetään, kun WINE-prosessi käynnistyy, ja se luo UNIX-pistokkeen ns. *wineprefix*-kansioon, joka sisältää WINE:n käyttämät dll-kirjastot, sekä yleisen Windows-tietorakenteen. Kaikki samaan *wineprefix*-kansioon suunnatut WINE-prosessit hyödyntävät samaa WINE-palvelinta. [10.] WINE ei anna itsensä alla suoritettujen Windows-prosessien käyttää Windows-ajureita, sillä se ei toteuta Windows-ydintä. Täten WINE:n on täytynyt luoda välityskerros UNIX-ytimen päälle (WINE drivers katso kuva 7), joka antaa Windows *NTDLL*- ja *KERNEL32*-kerroksille tarvittavat toiminnot. [10.]

Windowsin järjestelmän koodi, sisäänlukien WINE:n hyödyntämät DLL-kirjastot, on lisenssöity suljetulla lisenssillä. Täten WINE:n kehityksessä ei vahingossakaan saa hyödyntää mitään Microsoftin omistamaa koodia. Muuten teoriassa Microsoft pystyy lopettamaan koko projektin tekijänoikeuksien rikkomisen takia. [14.] WINE-kehittäjät hyödyntää kehitykseen ns. *black-box* -testausta, jossa tutkitaan ohjelman toimintaa ilman sen sisäisten rakenteiden tutkimista [15]. WINE-kehittäjät eivät siis voi yrittää tutkia esimerkiksi Windowsin *System.dll*-kirjaston toimintaa syöttämällä sitä koodin purkajaan, sillä se paljastaisi kirjaston sisäistä toteutusta, jonka Microsoft omistaa. Kehittäjät pyrkivät siis tutkimaan kirjastojen rajapintojen kuvauksia, joita on useita saatavilla Microsoftin koulutus-sivuilta, ja toteuttamaan korvaavia kirjastoja näihin rajapintoihin sopiviksi. [14; 10]



Kuva 7: WINE-toteutuksen arkkitehtuuri (vasen) sekä Windows NT -arkkitehtuuri (oikea) [10]

4 Toteutus

4.1 Aloitus

Opinnäytetyön aihetta tutkiessa käytiin läpi, mistä luvussa 3 mainituista vaihtoehtoista kannattaisi lähteä tekemään konseptia. Avalonia UI vaikuttaa paperilla ominaisuuksiensa puolesta parhaalta valinnalta Endeavourille. Avalonia UI suorituu natiivisti Linux-käyttöjärjestelmillä, joten sen pitäisi olla hieman tehokkaampi kuin WINE-käännöskerroksen kautta suorittaminen, ohjelmien suorittaminen natiivisti ilman erillisiä työkaluja on myös usein vakaampaa kuin käännöskerroksen kautta. Kuitenkin, kuten luvussa 2.2 mainittiin, Endeavourin käyttöliittymä pohjaa sen yläohjelman käyttöliittymään, eikä Endeavour ole itsenäinen ohjelmisto. Tämä aiheuttaa sen, että Avalonia UI:n käyttöönotto vaatisi koko ohjelmiston käyttöliittymän refraktoroinnin, jonka työmääräksi arvoitiin hyvin suureksi. Täten Avalonia UI jää liian työlääksi kokeiluksi tämän tutkimuksen kannalta. Avalonia XPF lupaa lähes suoraa käännöstä WPF-pohjaisille ohjelmistoille. Kuitenkin Endeavour käyttää WPF-kirjastoa vain testitapausten piirtämiseen ja muuten hyödyntää WinForms-kirjastoa.

Konseptin tekoa varten valittiin siis WINE. Koska WINE on käännöskerros, sillä Endeavourin suorittaminen pitäisi toimia ilman suurempia koodimuutoksia.

Ennen WINEn käyttöä Endeavourin käyttämä .NET Framework 4.7.2 täytyy päivittää joksikin .NET-versioksi, sillä .NET Frameworkin lisenssi kieltää ohjelmistokehyksellä koottujen ohjelmistojen suorittamisen muilla kuin aktiivisen Windows-lisenssin omaavilla laitteilla [16], .NET on julkaistu MIT-lisenssin alla, joka ei sisällä tällaisia rajoitteita. .NET on Microsoftin avoimen lähdekoodin implementaatio .NET Frameworkistä, sekä toisin kuin edeltäjänsä, .NET on saatavilla myös muille alustoille kuin Windows. Endeavour-WINE-ajoa varten päädyttiin valitsemaan .NET-versio 7, sillä Endeavourin yläohjelmaan oli jo aikaisemmin aloitettu tähän käännöstyö.

4.2 Työkalut

Endeavour-työkalun kehitystä tehtiin käyttämällä Microsoft *Visual Studio Professional 2022* -ohjelmistoympäristöä. Visual Studio on erityisesti C#-kehitykseen tehokas työkalu, sillä se sisältää useita työkaluja, kuten ajonaikasen debugger-työkalun, jolla voi keskeyttää ohjelman ajon ja tutkia sen sisäistä tilaa keskeytyksen aikana. Myöskin Visual Studio kykenee profiloimaan sen kautta suoritetun C#-ohjelman prosessori sekä Hajasaantimuistin (Random Access Memory) (RAM) käyttöä ajon aikana. Lisäksi Visual Studio kykenee pitämään kirjaa kaikkien luokkien ja metodien kutsumääristä ja paikoista, joka helpottaa uuteen koodipohjaan tutustumista suunnattomasti.

Versionhallintaan Endeavour käyttää Nokialla sisäisesti hostattua Gitlab-versionhallintaohjelmistoa. Endeavour säilötään samassa repositoriossa sen sisällyttävän yliohjelmansa kanssa. Gitlab repositorio sisältää jatkuvan koonnin (CI) ympäristön, joka sisältää myös koonnin sekä muutaman testin Endeavourille.

4.3 Prosessi

Endeavour-Linux-version tekeminen alkoi aiemmin mainitun .NET Framework 4.7.2 päivittämisellä versioon .NET 7. Tästä päivityksestä oli jo olemassa aikaisemmin aloitettu työ Endeavourin yliohjelmalle, josta pystyttiin alkamaan tutkia Linux-version mahdollisuutta. .NET-version päivitys sujui lähes ongelmitta lukuunottamatta muutamia laajempia refraktorointeja, joita täytyi tehdä, jotta Endeavour saatiin koottua ilman virheitä. Nämä ongelmat liittyivät käytettyihin .NET Framework -teknologioihin, joita ei enää ole olemassa .NET-kirjastossa.

CodeDOM ja Roslyn

Endeavour hyödyntää toiminnassaan ajon aikana koottuja datatyyppejä, joita käytetään esimerkiksi *Endeavour-server*-palvelimelta tulevien viestien datakenttien siirrosten laskemiseen. Näiden datatyyppien ajonaikaiseen kokoamiseen oli käytetty *CodeDOM*-kirjaston *CSharpCodeProvider*-luokkaa, joka ei ole tuettu .NET alla.

Vanha *CodeDOM CSharpCodeProvider*-luokkaa hyödyntävä koodi on jotakuinkin esimerkikoodin 4 mukainen. *CSharpCodeProvider*-luokalle annetaan koodipolku, jonne on kirjoitettu, joko ajon aikana tai ennen suoritusta C# koodia, joka halutaan koota. Endeavour-tapauksessa koodut luokat halutaan tallettaa taulukkoon myöhempää käyttöä varten, joten luotava *DLL-Assembly* luodaan vain muistiin. Muistiin luonti määritellään *parameters.GenerateInMemory*-parametrilla ja *parameters.GenerateExecutable = false* -parametrilla määritellään, että luodusta objektista ei tehdä suoritettavaa tiedostoa. Ennen koodin kokoamista parametreiksi annetaan myös koontiin tarvittavia kirjastoja. Esimerkin tapauksessa nämä ovat *System.dll* ja *System.Data.dll*.

```

1 public void CSharpProviderExample()
2 {
3     /* Koottava koodi on etukäteen luotu joko
4         ohjelmallisesti tai käsin ja kirjoitettu
5         tiedostoon */
6     string codepath = "/path/to/file";
7     CSharpCodeProvider csProvider = new CSharpCodeProvider();
8     CompilerParameters parameters = new CompilerParameters();
9     parameters.GenerateExecutable = false;
10    parameters.GenerateInMemory = true;
11    parameters.CompilerOptions = "/unsafe+ /checked- /optimize-";
12
13    parameters.ReferencedAssemblies.Add("System.dll");
14    parameters.ReferencedAssemblies.Add("System.Data.dll");
15
16    CompilerResults result = csProvider
17        .CompileAssemblyFromFile(parameters, codepath);
18    System.Reflection.Assembly assembly
19        = result.CompiledAssembly;
20    /* Ajon aikana koottava Assembly on nyt luotu */
21 }

```

Koodiesimerkki 4: Esimerkki CodeDOM CSharpCodeProvider -luokkaa käyttävästä ajon aikaisesta C#-Assemblyn luonnista.

Koodiesimerkki 5 näyttää vastaavan toteutuksen uudesta insinööriyön aikana tehdystä vanhaa CSharpCodeProvider-luokkaa käyttävän järjestelmän korvaajsta. Korvaavana C#-koodin koontiluokkana käytetään Roslyn-kirjaston CSharpCompilation-luokkaa. Roslyn, tai oikealta nimeltään .NET Compiler Platform, on MIT-lisenssin omaava avoimen lähdekoodin C#-koonti, ja analysointityökalu, joka toisin kuin CodeDOM, on tuettu .NET-versioilla. CSharpCompilation-luokka toimii samankaltaisesti kuin CSharpCodeProvider. Pääasiallinen ero kahden luokan välillä, syntaksin lisäksi, on luotavan Assemblyn käsittely, jonka CSharpCodeProvider pystyy tuottamaan suoraan *CompilerOptions*-objektista, Roslyn-kirjastolla tosin se täytyy kirjoittaa käyttäen *MemoryStream*-luokkaa.


```

1  private List<MetadataReference> References
2      = new List<MetadataReference>();
3
4  /* Tämä funktio ei ole pakollinen, mutta auttaa jos
5     koottavalle koodille täytyy lisätä kirjastoja,
6     jotka eivät kuulu C# SDK:hon */
7  private bool AddAssembly(string assemblyDll)
8  {
9      if (string.IsNullOrEmpty(assemblyDll)) return false;
10     var file = Path.GetFullPath(assemblyDll);
11
12     if (!File.Exists(file)) return false;
13
14     try
15     {
16         var reference = MetadataReference.CreateFromFile(file);
17         if (References.Contains(reference)) return true;
18         References.Add(reference);
19     }
20     catch
21     {
22         return false;
23     }
24     return true;
25 }
26
27 public void RoslynExample(string sourceString)
28 {
29     /* Roslyn käsittelee koottavia skriptejä suoraan
30        esimerkiksi string tyyppisten muuttujien kautta,
31        eikä sen tarvitse lukea tiedostosta */
32     AddAssembly("System.dll");
33     AddAssembly("System.Data.dll");
34
35     var options = new CSharpCompilationOptions(
36         OutputKind.DynamicallyLinkedLibrary,
37         reportSuppressedDiagnostics: true,
38         optimizationLevel: OptimizationLevel.Debug,
39         generalDiagnosticOption: ReportDiagnostic.Default,
40         allowUnsafe: true,
41         platform: Platform.X64
42     );
43
44     var compilation = CSharpCompilation.Create(
45         "CompiledAssembly.cs",
46         references: References,

```

```

47         syntaxTrees: new SyntaxTree[]
48         {
49             SyntaxFactory.ParseSyntaxTree(sourceString,
50             CSharpParseOptions.Default
51             .WithLanguageVersion(
52                 LanguageVersion.CSharp8),
53             encoding: Encoding.UTF8)
54         },
55         options: options);
56
57     Microsoft.CodeAnalysis.Emit.EmitResult compilationResult;
58     System.Reflection.Assembly assembly = null;
59
60     using (var ms = new MemoryStream())
61     {
62         compilationResult = compilation.Emit(ms);
63
64         if (compilationResult.Success)
65         {
66             ms.Seek(0, SeekOrigin.Begin);
67             assembly = System.Reflection.Assembly
68                 .Load(ms.ToArray());
69         }
70         /* Ajon aikana koottava Assembly on nyt luotu */
71     }
72 }

```

Koodiesimerkki 5: Esimerkki Roslyn-CSharpCompilation-luokaa käyttävästä ajon aikaisesta C#-Assemblyn luonnista.

APM ja TAP.

Endeavour hyödynsi tiedostojen lataamiseen vanhaa Asynchronous Programming Model (Asynkroninen ohjelmointimalli) (APM) -ohjelmointimallia. APM pohjaa *IAsyncResult*-nimiseen luokkaan, joka ilmaisee asynkronisen toiminnon tilaa. APM-suunnittelumallin mukaan asynkroninen operaatio aloitetaan kutsumalla **BeginOperaatioNimi**-funktiota, joka aloittaa asynkronisen operaation toisella säikeellä. Tämän jälkeen kutsunut säie voi jatkaa toimintaansa. Asynkroninen operaatio loppuu **EndOperaatioNimi**-funktion kutsuun, joka odottaa asynkronisen operaation loppua ja palauttaa tulokset *IAsyncResult*-luokan perivänä objektina. [17.] Koodiesimerkki 6 esittää samankaltaista tiedostonlukuoperaatiota, jota täytyi muokata Endeavourissa. Tiedoston luku on toteutettu hyödyntäen *delegate* tyyppi-

piä. Delegate on C#-tyyppi, joka viittaa metodiin, jolla on määritelty parametrilista ja palautustyyppi. Esimerkissä APM-mallin mukaiselle *BeginRefreshFromFile*-metodille annetaan *AsyncCallback*-takaisinkutsuobjekti, jolla voidaan lukuoperaatioiden jälkeen kutsua kästtelyfunktiota *doSomething*. *RefreshFromFileDelegate*-delegate-objektille annetaan lukumetodi *refreshFromFile* parametriksi, ja kutsutaan delegate-luokan metodia *BeginInvoke*, joka aloittaa *refreshFromFile*-metodin suorittamisen uudella säikeellä ja lopuksi kutsuu sille määriteltyä *AsyncCallback*-objektia, jos sellainen on annettu. *AsyncCallback*-objektina annettu metodi sitten kutsuu asynkronisen operaation loputtua *EndInvoke*-metodia, joka estää kutsuvan säikeen suorittamisen kunnes asynkroninen Delegate-operaatio on loppunut ja palauttaa operaation tuloksen *IAAsyncResult*-objektina. [18]

```

1  public void APMExample()
2  {
3      string filePath = "Path/To/File";
4      StatusObject status = new StatusObject();
5      AsyncCallback callback = new AsyncCallback(doSomething);
6      status.BeginRefreshFromFile(filePath, callback, status);
7  }
8  private void doSomething(IAAsyncResult result)
9  {
10     /* Tee jotain asynkronista */
11     StatusObject status = (StatusObject)result.AsyncState;
12
13     if (!status.EndRefreshFromFile(result))
14     {
15         /* Jotain meni vikaan */
16     }
17     /* Tee jotain luetulla tiedolla */
18 }
19
20 class StatusObject
21 {
22     private delegate
23         bool RefreshFromFileDelegate(string filePath);
24
25     public void BeginRefreshFromFile(string filePath,
26         AsyncCallback asyncCallback, Object state)
27     {
28         RefreshFromFileDelegate refreshDelegate =
29             new RefreshFromFileDelegate(this.refreshFromFile);
30         refreshDelegate

```

```

31         .BeginInvoke(filePath, asyncCallback, state);
32     }
33
34     public bool EndRefreshFromFile(IAsyncResult result)
35     {
36         RefreshFromFileDelegate refreshDelegate =
37             ((AsyncResult)result).AsyncDelegate
38             as RefreshFromFileDelegate;
39
40         try
41         {
42             bool status = refreshDelegate.EndInvoke(result);
43             return status;
44         }
45         catch
46         {
47             return false;
48         }
49         return false;
50     }
51
52
53     private bool refreshFromFile(string filePath)
54     {
55         /* Tänne tulisi tiedoston luvun toteutus */
56         return true;
57     }
58
59 }

```

Koodiesimerkki 6: APM-mallin mukaisesti tehty esimerkki tiedoston lukemisesta asynkronisesti.

Microsoft ei enää nykyisin suosittele APM-mallin käyttöä asynkronisiin operaatioihin [19]. Tämän lisäksi .NET ei sisällä Endeavourin implementaation käyttämää *IAsyncResult*-rajapinnan perivää *AsyncResult*-luokkaa, jolla päästiin käsiksi operaation tilaan. Täten päädyttiin muuttamaan vanhat asynkroniset toiminnot hyödyntämään uutta Task Parallel Library (Tehtävä-Rinnakkais-kirjasto) (TPL) -kirjastoa ja Task-based Asynchronous Pattern (Tehtäväpohjainen asynkroninen malli) (TAP) -mallia

```

1  public void TAPExample()
2  {
3      string filePath = "/Path/To/File";
4      TAPStatusObject status = new TAPStatusObject();
5      Action<Task<bool>, Object> taskCallback = doSomething;
6      status.BeginRefreshFromFile(filePath, taskCallback, status);
7  }
8
9  private void doSomething(Task<bool> result, Object state)
10 {
11     /* Tee jotain asynkronista */
12     TAPStatusObject status = (TAPStatusObject)state;
13
14     if (!status.EndRefreshFromFile(result))
15     {
16         /* Jotain meni vikaan */
17     }
18     /* Tee jotain luetulla tiedolla */
19 }
20
21 class TAPStatusObject
22 {
23     public async void BeginRefreshFromFile(string filePath,
24         Action<Task<bool>, Object> taskCallback, Object state)
25     {
26         Task<bool> refreshTask;
27         await (refreshTask = Task.Run(()
28             => refreshFromFile(filePath)));
29         await refreshTask.ContinueWith(result
30             => taskCallback(result, state));
31     }
32
33     public bool EndRefreshFromFile(Task<bool> task)
34     {
35         try
36         {
37             bool status = task.Result;
38             return status;
39         }
40         catch
41         {
42             return false;
43         }
44         return false;
45     }
46

```

```

47
48     private bool refreshFromFile(string filePath)
49     {
50         /* Tänne tulisi tiedoston luvun toteutus */
51         return true;
52     }
53
54 }

```

Koodiesimerkki 7: TAP-mallin mukaisesti tehty esimerkki tiedoston lukemisesta asynkronisesti.

TAP-mallin toiminta pohjaa TPL-kirjaston luokkaan *Task*. *Task*-luokka edustaa työtehtävää, joka ei palauta arvoa, kun taas *Task<T>*, jossa *T* on palautettavan arvon tyyppi, edustaa työtehtävää, joka palauttaa arvon. Koodiesimerkki 7 näyttää työn aikana tehtyä toteutusta vastaavan muutoksen esimerkkiin 6. Muutoksia tehdessä haluttiin joutua refraktoroimaan mahdollisimman vähän, joten lopputulos ei ole aivan tyylipuhdasta TAP-mallin mukaista koodia, vaan enemmänkin jotakin APM- ja TAP-mallien välimaastossa. Koodissa suurimpina muutoksina on *delegate*-objektin poistuminen sekä *BeginInvoke*- ja *EndInvoke*-kutsujen poistuminen. *AsyncCallback*-objekti on muuttunut *Action<Task<bool>, Object>*-objekiksi, joka on toiminnallisesti vastaava kuin vanha *AsyncCallback*-objekti. *Action*-takaisinkutsuobjektia kutsutaan *refreshTask.ContinueWith*-metodikutsulla, jota kutsutaan, kun aikaisempi *Task.Run*-kutsu on loppunut. Nämä kutsut on merkitty *await*-avainsanalla, joka ilmaisee, että näiden metodikutsujen loppumista tulee odottaa, ennen kuin siirrytään seuraavaan operaatioon, *await*-metodit eivät kuitenkaan estä senhetkisen säikeen suoritusta [20].

4.4 Testaus

Endeavour-WINE-versiota testattiin kahdella Linux-ympäristöllä. Käytössä oli *Virtualbox 7.0* -virtuaalikone, sekä *Thinkpad T440s* kannettava tietokone. Virtuaalikoneelle annettiin 6 GB RAM-muistia, sekä sen käytössä oli 4 säiettä virtuaalikonetta suorittavan Thinkpad T14 Intel 15-1145G7 -prosessorin 8 säikeestä. Prosessorin virtualisoinnin rajapintana käytettiin Microsoftin *Hyper-V*-virtualisointirajapintaa.

Thinkpad T440s kannettavassa tietokoneessa käytettävänä oli 8 GB RAM-muistia sekä Intel i7-4600U -prosessori.

Molemilla laitteilla oli asennettu Fedora Linux -versio 37. Fedora Linux on Red Hat Inc. -yrityksen tuottama Linux-distribuutio. Tämä versio valittiin, sillä Red Hat Inc. tuottaa Red Hat Enterprise Linux -distribuutiota, joka on yrityksille suunnattu Linux-käyttöjärjestelmä, jota käytetään myös Nokialla. Myös Fedora Linux on yksi suosituimmista Linux-distribuutioista. Virtualbox-virtuaalikoneella käytettiin LXQT-työpöytäympäristöä, ja Thinkpad T440s kannettavalla käytettiin Cinnamon-työpöytäympäristöä. Työpöytäympäristön valinnalla ei pitäisi olla erityisemmin väliä ohjelmistojen toimivuuden kannalta, sillä molemmat käyttävät X11-ikkunointijärjestelmää, joka hallinnoi ohjelmistoikkunoiden piirtämistä.

Virtuaalikoneella käytettiin WINE-versiota 8.1 (Staging), ja Thinkpad T440s käytti versioita 8.4 (Staging) sekä 8.5 (Staging). WINE Staging -versio on WINE-versio, joka sisältää tuoreimmat korjaukset, joita repositorioon on tehty, WINEsta on myös saatavilla Stable, eli vakaa versio, mutta se on huomattavasti ominaisuuksissaan jäljessä Staging-versiota. Staging-versio valittiin, sillä se on Fedora Linuxin oletusversio WINElle. Kaikki Endeavour WINE-ajon testaaminen toteutettiin käsin kokeilemalla läpi eri toimintoja ja prosesseja. .NET 7 -päivitykseen liittyvien muutosten testaaminen tapahtui pääasiallisesti Windows käyttöjärjestelmällä, ja gitlab-repositorion CI-putken avulla.

4.5 Ongelmat

Korjatut ongelmat

WINE-version suorittaminen ei sujunut täysin ongelmitta. Matkan varrella löytyi useampia ongelmia, joista osaa ei tämän insinööritöön aikana ehditty korjaamaan. Projektin kokeilu alkoi täysin virtuaalikonetta käyttämällä, jolloin huomattiin Endeavour-testiajojen suoritusnopeuden olevan huomattavasti hitaampi kuin windows versiolla. Kuitenkaan tällaista hitautta ei ollut nähtävissä myöhemmin kun, testikäyttöön otettiin Linux-kannettava. Mitä luultavimmin WINE ei käyttäydy hyvin

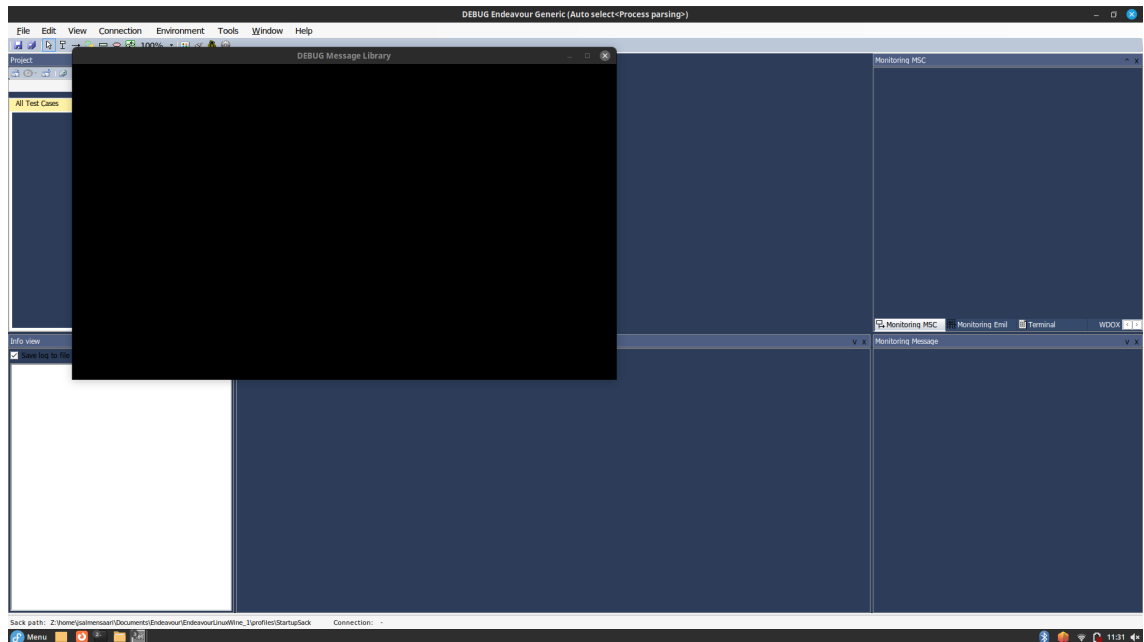
Virtualboxin virtualisoidun prosessorin kanssa, joka aiheutti hitautta.

Endeavour piirtää useita ikkunoita käynnistyessään, joista suurin osa piilotetaan. WINE ei osaa kääntää tätä toimintoa oikein X-ikkunointijärjestelmälle, jolloin piilotetut ikkunat jäivät näytölle näkyviin mustina reagoimattomina ikkunoina (katso kuva 8), jotka pystyttiin sulkemaan vasta, kun ikkuna manuaalisesti avattiin Endeavourin valikoiden kautta. Ongelma saatiin korjattua poistamalla rikkinäiset ikkunat Endeavour-käynnistyksessä avattavien ikkunoiden listasta, jos Endeavour koottiin WINE-käyttöä varten.

Korjaamatta jääneet ongelmat

WINE-ajon alla huomattiin muutama ongelma, joita ei saatu korjattua. Yksi ongelma, jota ei saatu korjattua, löytyi Endeavour-testitapauksen ikkunan kokoa muuttamalla. Testitapauksen koon muuttaminen kaatoi Endeavourin esimerkin 8 mukaisen virheen kanssa. Virhe voisi johtua WINEn tavasta kääntää WPF-kirjaston ikkunan koon muuttamiskäskyt, mutta yksi mahdollinen syy on myös kannettavan tietokoneen vanha näytönohjain. Ohjelmaa ajettaessa WINEn kautta tulee Mesa-grafiikkakirjastolta varoitus *MESA-INTEL: warning: Haswell Vulkan support is incomplete* varoittaa i7-4600U -prosessorin sisäänrakennetun näytönohjaimen ajureiden puutteellisuudesta.

.NET 7 -päivityksestä syntyi myös virhe, jota ei ehditty selvittää. C# sisältää *Type marshalling* -ominaisuuden, jolla voi käsitellä hallinnoimattomia datatyyppejä. .NET Framework ja .NET välillä marshal-ominaisuudessa on muuttunut jotakin, joka on hajoittanut sen käytön. Vanhalla .NET Framework -versiolla Type marshalling toimi normaalisti. .NET -puolella, kun yritetään selvittää datatyypin kentän sijaintia muistissa Type-marshallingin kautta, tulee virhe *Type 'Type' cannot be marshaled as an unmanaged structure; no meaningful size or offset can be computed*. eli marshal-järjestelmä ei pysty käsittelemään annettua datatyyppiä hallinnoimattomana rakenteena. Ongelmaa tutkiessa ei löydetty mitään eroa datatyypeistä, joita yritetään lukea, joten oletettavasti ongelma on itse marshal-ominaisuudessa.



Kuva 8: Endeavour-Linux-versiossa piirtyvä musta reagoimaton ikkuna.

```

1 X Error of failed request: BadWindow (invalid Window parameter)
2   Major opcode of failed request: 12 (X_ConfigureWindow)
3   Resource id in failed request: 0x40052a3
4   Serial number of failed request: 2314
5   Current serial number in output stream: 2315

```

Koodiesimerkki 8: X-ikkunointijärjestelmän antama virhe Endeavour-testitapauksen ikkunan koon muuttamisesta

Näiden korjaamatta jääneiden ongelmien takia työn aikana ei saatu tehtyä asennuspakettia Linux-käyttöjärjestelmille. Kuitenkin työssä saatiin osoitettua, että Endeavour-ajo WINEn kautta Linuxissa on mahdollista, ja kehitysresurssien sal-
liessa ongelmien korjaaminen olisi kyllä mahdollista.

5 Yhteenveto

Työssä tutkittiin vaihtoehtoja Endeavourin-Linux-versiolle. Ennen työn aloittamista oli päädytty kahteen todennäköiseen vaihtoehtoon Avalonia UI:hin sekä WINEin. Työssä tutkittiin näiden vaihtoehtojen hyviä ja huonoja puolia, ja loppujen lopuksi päädyttiin tekemään PoC-versiota Linux-käyttöliittymästä käyttämällä WINE-käännöskerrosta, sillä sen arvioitiin olevan työmäärältään pienempi.

Työtä aloittaessa toivottiin, että työn tuloksena saisi jonkinlaisen asennuspaketin Endeavour-Linux-versiolle. Tämä ei kuitenkaan työn aikarajoitusten sisällä ollut mahdollista. Täten työ ei pystynyt etenemään konkreetian puolelle vaan jäi konseptin tasolle. Työhön käytetyt tunnit eivät kuitenkaan menneet hukkaan, vaan saatiin tärkeätä tietoa käännökseen tarvittavasta työmäärästä ja sen mahdollisuudesta.

Pitkällä tähtäimellä katsoessa en näe WINE-version olevan paras vaihtoehto, sillä käännöskerroksen lisääminen ohjelmiston suorittamiseen lisää yhden kerroksen pinoon, joka voi murtua. Erilliselle Linux-versiolle tarvittaisiin myös tukihenkilöitä, jotka voivat auttaa käyttäjiä tarpeen tullen sekä korjata bugeja ja muita virheitä, joita todennäköisesti tulisi kehityksen edetessä. Näin siis loppujen lopuksi koko käyttöliittymän kääntäminen pohjakomponenteista saakka esimerkiksi Avalonia - UI:lle olisi viisain vaihtoehto, sillä sitten Linux-versiota ei tarvitsisi enää erikseen tukea, eikä ohjelmapinossa olisi erillistä käännöskerrosta.

Lähteet

- 1 evil', Gilles 'SO- stop being. 2010. Why won't Windows EXE files work on Linux? Verkkoaineisto. <<https://superuser.com/a/209736>>. Luettu 04. 05. 2023.
- 2 Kirk, Steven. 2013. Avalonia Initial commit. Verkkoaineisto. <<https://github.com/AvaloniaUI/Avalonia/commit/cd2b7530f5e3e0cea2c45c39bdc9a433149c2200>>. Luettu 17. 04. 2023.
- 3 Avalonia UI: About Us 2023. Verkkoaineisto. <<https://avaloniaui.net/About>>. Luettu 17. 04. 2023.
- 4 Kirk, Steven. 2014. Avalonia Intro.md. Verkkoaineisto. <<https://github.com/AvaloniaUI/Avalonia/blob/d6109d973be3d274c02e3f55f8f06e163a804bb9/Docs/intro.md>>. Luettu 17. 04. 2023.
- 5 Avalonia XPF Announcement 2023. Verkkoaineisto. <<https://twitter.com/AvaloniaUI/status/1625042505606615040?s=20>>. Luettu 18. 04. 2023.
- 6 Avalonia XPF 2023. Verkkoaineisto. <<https://avaloniaui.net/XPF>>. Luettu 18. 04. 2023.
- 7 Anderson, Tim. 2023. Interview: Avalonia XPF introduced for WPF on Mac, Linux and eventually mobile. Verkkoaineisto. <<https://devclass.com/2023/02/13/interview-avalonia-xpf-introduced-for-wpf-on-mac-linux-and-eventually-mobile/>>. Luettu 18. 04. 2023.
- 8 Wine History 2023. Verkkoaineisto. <https://wiki.winehq.org/Wine_History>. Luettu 19. 04. 2023.
- 9 About WINE 2023. Verkkoaineisto. <<https://www.winehq.org/about>>. Luettu 17. 04. 2023.
- 10 Wine Developer's Guide/Architecture Overview 2023. Verkkoaineisto. <https://wiki.winehq.org/Wine_Developer%27s_Guide/Architecture_Overview>. Luettu 19. 04. 2023.
- 11 Dynamic-link library 2023. Verkkoaineisto. <https://en.wikipedia.org/wiki/Dynamic-link_library>. Luettu 02. 05. 2023.
- 12 What is a DLL 2023. Verkkoaineisto. <<https://learn.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library>>. Luettu 02. 05. 2023.
- 13 Eikum, Andrew. 2019. Working on Wine Part 1 - The Wine Ecosystem. Verkkoaineisto. <<https://www.codeweavers.com/blog/aeikum/2019/1/3/working-on-wine-part-1-the-wine-ecosystem>>. Luettu 04. 05. 2023.

- 14 McKenzie, James. 2009. Legal issues. Verkkoaineisto. <<https://forum.winehq.org/viewtopic.php?p=37364&sid=068defede91a36ea192cdefd08dde32b#p37364>>. Luettu 19. 04. 2023.
- 15 Black-box testing 2023. Verkkoaineisto. <https://en.wikipedia.org/wiki/Black-box_testing>. Luettu 03. 05. 2023.
- 16 Microsoft .NET Framework Redistributable EULA 2006. Verkkoaineisto. <[https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/ms994405\(v=msdn.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/ms994405(v=msdn.10)?redirectedfrom=MSDN)>. Luettu 23. 02. 2023.
- 17 Asynchronous Programming Model (APM) 2021. Verkkoaineisto. <<https://learn.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/asynchronous-programming-model-apm>>. Luettu 27. 04. 2023.
- 18 Calling Synchronous Methods Asynchronously 2021. Verkkoaineisto. <<https://learn.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/calling-synchronous-methods-asynchronously>>. Luettu 27. 04. 2023.
- 19 Asynchronous programming patterns 2023. Verkkoaineisto. <<https://learn.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/>>. Luettu 27. 04. 2023.
- 20 await operator - asynchronously await for a task to complete 2023. Verkkoaineisto. <<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await>>. Luettu 28. 04. 2023.