M-Files-julkaisutyökalun kehitys

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
kevät, 2022

Svante Laine

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus

Tekijä Svante Laine **Vuosi** 2022

Työn nimi M-Files-julkaisutyökalun kehitys

Työn ohjaaja/t Tommi Lahti (HAMK), Ville Linnalehto (M-Files Oy)

TIIVISTELMÄ

Opinnäytetyössä suunniteltiin ja ohjelmoitiin ominaisuuspäivitys M-Files Oy:n C#-pohjaiseen Vientityökaluksi ristittyyn apuohjelmaan. Vientityökalu on M-Filesin sisäisesti ohjelmoima, ja sisäisesti käyttämä työkalu M-Files-ympäristöjen väliseen asetusten siirtoon. Toimeksiantajana toimi M-Files Oy, joka tuotti vaatimusmäärittelyn päivitykselle. Vientityökalun käyttö rajoittui palvelinympäristöihin, joiden välillä on internetyhteys. Työkalua on kuitenkin tarve käyttää myös paikoissa, joissa internetiä ei ole saatavilla.

Teoriaosuudessa tutustuttiin ohjelman yleiseen toimintaperiaatteeseen, ja suunniteltiin, kuinka toimeksiannon täyttävät muutokset saadaan sulautettua ohjelmaan tehden mahdollisimman pieniä muutoksia sen toimintalogiikkaan, samalla säilyttäen mahdollisimman yksinkertainen käyttäjäkokemus.

Käytännön osuudessa ohjelmoitiin suunnitelman mukaiset lisäykset sekä WPF-pohjaiseen käyttöliittymään, että sen taustalla olevaan C#-pohjaiseen toimintalogiikkaan. Työ onnistui korkealla tasolla hyvin, ja ohjelmaan tehdyt muutokset ajavat asiansa, joskin yksi toimeksiannon ominaisuuksista jäi puuttumaan.

Avainsanat C#, WPF, Ohjelmistosuunnittelu, Ohjelmistokehitys

Sivut 26 sivua, joista liitteitä 0 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology

Author Svante Laine **Year** 2022

Subject M-Files deployment tool development

Supervisors Tommi Lahti (HAMK), Ville Linnalehto (M-Files Oy)

ABSTRACT

In the thesis, new features were designed and programmed into M-Files Oy's helper program named "Deployment tool". Deployment tool is a C#-based program that is internally developed and used by M-Files Oy to assist in moving M-Files configurations from one server environment to another. The thesis was commissioned by M-Files Oy, and the requirements for the update were also provided by M-Files Oy. The tool's usage was limited to internet connected server environments, but new use cases required it to function in offline environments as well.

The theory section explores the overall working principles of the tool and identifies where and how to modify the program to achieve the new functionality. All design choices have the aim of keeping changes to the program's logic as small as possible and keeping user experience as streamlined as possible.

The planned changes were implemented in the practical section. Changes were made to both the program's WPF-based graphical user interface, as well as the C#-based background logic. The overall quality of both the design and implementations was satisfactory, but one of the features described in the requirements was not implemented.

Keywords C#, WPF, Software design, Software engineering

Pages 26 pages including appendices 0 pages

SISÄLLYS

LIITTEET

1 JOHDANTO

M-Files Oy on yhtiö, joka kehittää, myy, ja asentaa M-Files-nimistä tiedonhallintaohjelmistoa. Erittäin korkealla tasolla ohjelman tarkoitus on parantaa sähköisten asiakirjojen löydettävyyttä, ja automatisoida niiden hallinnointiin liittyviä toimia.

Ennen kuin ohjelmisto otetaan käyttöön asiakkaan tuotantoympäristössä, M-Files Oy kehittää asiakkaan tarpeita vastaavaa M-Files-konfiguraatiota erillisessä kehitysympäristössä. Kun konfiguraatio kelpuutetaan tuotantokäyttöön, räätälöity M-Files konfiguraatio kopioidaan internetin välityksellä asiakkaan tuotantopalvelimelle.

Siirtoon käytetään yleensä niin kutsuttua Julkaisutyökalua, joka on M-Files-ohjelmistosta erillinen apuohjelma. Julkaisutyökalu luo yhteyden kahteen erillään olevaan M-Files-palvelimeen, ja siirtää niiden välillä asetuksia. Kun yhden tai molempien ympäristöjen välille ei saada internetyhteyttä, Julkaisutyökalua ei voida käyttää, sillä se on alun perin suunniteltu toimimaan vain internetin kautta. Näissä tapauksissa järjestelmäasiantuntijan täytyy kopioida konfiguraatio kehitysympäristöstä massamuistilaitteelle ja siirtyä fyysisesti asiakkaan toimitiloihin suorittamaan asennus. Kopiointi ja vienti tuotantoon tapahtuu tällöin kömpelön manuaalisen prosessin kautta, joka vie turhaan aikaa ja jättää sijaa huolimattomuusvirheille.

M-Files Oy on tuottanut vaatimusmäärittelyn tiettyjen manuaalisen siirron vaiheiden automatisoinnille julkaisutyökalun avulla. Opinnäytetyön tavoite on suunnitella ja toteuttaa vaatimusmäärittelyn mukaiset ominaisuudet julkaisutyökaluun.
Opinnäytetyössä pyritään vastaamaan seuraaviin kysymyksiin:

- Millaisessa muodossa operaatioiden lopputulokset kannattaa säilyttää offline-tuen saavuttamiseksi?
- Miten säilytetään suoraviivainen käyttäjäkokemus, kun tuki uusille käyttötapauksille lisätään ohjelmaan?
- Miten M-Filesin COM APIa voidaan hyödyntää uusien ominaisuuksien toteuttamisessa?

2 M-FILES-OHJELMISTO

Tiedonhallintajärjestelmänä M-Files on suunniteltu vastaamaan sähköistetyn liiketoiminnan tarpeisiin. Nykyaikaisten yrityksien ja yhtiöiden dokumentit (sähköiset asiakirjat, kuvat, videot, ynnä muut liiketoimintaan liittyvät tiedostot) säilytetään yleensä sirpaloituneena monessa eri järjestelmässä, seurauksena liiketoiminnan tukena käytettyjen ohjelmien kirjosta.
(M-Files Oy, n.d.a)

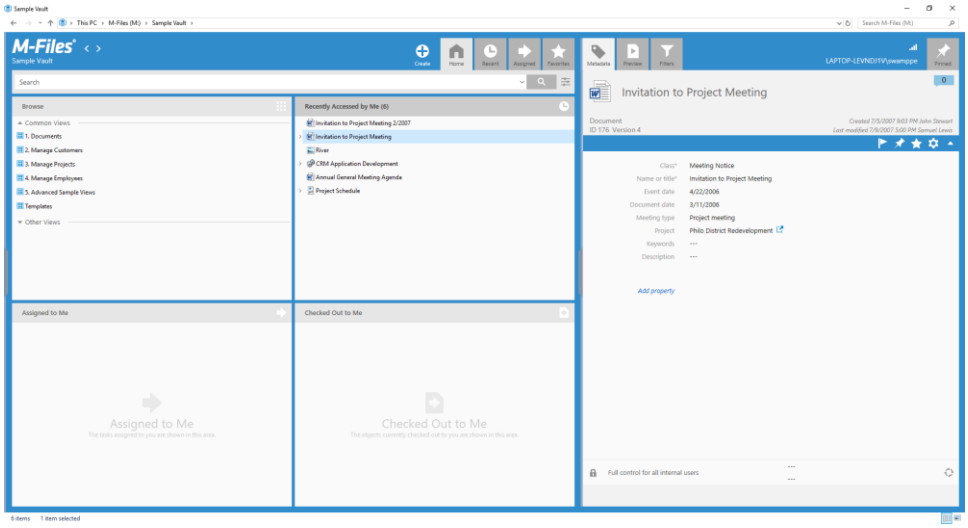
M-Files-ohjelmiston toimintaperiaate on ulkopuolisissa järjestelmissä olevien dokumenttien jatkuva luettelointi ja tarkkaileminen. M-Filesin tarkkailemia järjestelmiä, vaikkapa verkkolevyä tai CRM:ää kutsutaan tietolähteeksi. Kun käyttäjä tekee haun M-Filesin kautta, ohjelmisto etsii hakuehtoja vastaavia dokumentteja kaikista luetteloistaan, ja antaa käyttäjälle niistä koosteen.
(M-Files Oy, n.d.a)

Luetteloituihin dokumentteihin liitetään metatietoa. Metatiedolla tarkoitetaan tietoa, joka kuvailee tiedostoja ja niiden sisältöjä. Yleisesti käytettyä metatietoa on mm. tiedoston tyyppi; muistio, agenda, sopimus, jne., tai projekti, johon dokumentti liittyy. Käyttäjä voi vapaavalintaisesti liittää tiedostoihin uutta metatietoa, ja muokata jo liitettyä metatietoa.
(M-Files Oy, n.d.b)

M-Filesin seuraamia dokumentteja voi etsiä, suodattaa, ja järjestellä metatiedon perusteella, ja ohjelmiston tehokkuus tiedon järjestelyssä on suoraan riippuvainen käytetyn metatiedon tarkkuudesta. Metatietoa käytetään myös dokumenttien ”elinkaarien” tallettamiseen; M-Files panee merkille, kuka on tarkastellut tai muokannut dokumentteja, ja mitä muutoksia on tehty. Lisäksi M-Files voi säätää automaattisesti dokumenttien käyttöoikeuksia metatietojen perusteella.
(M-Files Oy, n.d.b)

M-Files toimii palvelin-käyttäjä-mallin kautta. Palvelintietokoneelle asennetaan M-Filesin palvelinympäristö, joka luetteloi tietolähteistä löytyvät dokumentit, muistaa dokumentteihin liitetyt metatiedot, ja määrittelee rakenteen, jonka mukaan käyttäjälle näytetään luetteloidut dokumentit. Käyttäjä tai käyttäjät selaavat palvelimen tietoja M-Filesin pääteohjelman kautta.
(M-Files Oy, n.d.c)

Kuva 1 M-Files Desktop-käyttöliittymän etusivu.



2.1 **Varastot**

Varastot (englanniksi ”Vault”) ovat M-Files-palvelimen sisäinen menetelmä tiedon jäsentelyyn. Varasto sisältää dokumentteja, ja niiden näyttämiseen liittyvää tietoa, kuten metatietotyyppejä ja muita asetuksia. Jokaiselle varastolle asetetaan yksi tai useampi tietolähde, kuten paikallinen hakemisto, verkkolevy, tai muu ulkoinen järjestelmä. M-Files-palvelimella voi olla myös yksi tai useampi varasto, joista kaikilla on omat tietolähteensä.
(M-Files Oy, n.d.d)

Käyttöoikeuksien hallinta M-Filesissa on kaksitasoista; ensimmäinen taso määrää, onko käyttäjällä ylipäänsä pääsyä tiettyyn varastoon. Pääsyn saamiseksi käyttäjällä täytyy olla sekä varastotason käyttäjätili, että palvelintason tunnus, joka on sidottu kuhunkin varaston käyttäjään.
(M-Files Oy, n.d.e)

Pääsy varastoon ei automaattisesti tarkoita, että käyttäjällä on oikeus nähdä tai muokata kaikkea varaston sisältöä. Käyttöoikeudet yksittäiseen dokumenttiin voivat vaihdella käyttäjätilin ryhmäjäsenyyden (AD- tai M-Files-ryhmä) tai kohteen metatietojen perusteella. Esim. kuvitteellisen organisaation markkinointiosastolla voisi olla oma AD-ryhmä. AD-ryhmän dokumenteiksi merkatut dokumentit voitaisiin piilottaa kaikilta *käyttäjätileiltä*, jotka eivät ryhmään kuulu.
(M-Files Oy, n.d.f)

2.2 **Tiedonhaku M-Filesista**

M-Filesissa on kaksi pääasiallista tapaa etsiä tietoa. Ensimmäinen tapa on haun tekeminen; käyttäjä voi tehdä hakuja joko vapaalla haulla, tai tiettyjen metatietojen arvojen perusteella. Vapaa vaku etsii hakusanaa kaikkien dokumenttien metatiedoista ja tiedostojen sisällöistä. Metatietopohjainen haku etsii dokumentteja käyttäjän määräämien metatietoyhdistelmien perusteella.
(M-Files Oy, n.d.g)

Toinen menetelmä on näkymien käyttö. Käytännössä näkymä on kuin perinteinen tietokoneen hakemisto, joka avataan tuplaklikkauksella. Näkymän sisältä löytyy otanta järjestelmän dokumenteista. Teknisesti näkymän voi mieltää tallennetuksi hakuehdoksi - kun käyttäjä avaa näkymän, M-Files etsii näkymän metatietosuodatusta vastaavat dokumentit varastosta, ja näyttää ne käyttäjälle.
(M-Files Oy, n.d.h)

3 TARVITTAVAT OHJELMOINTITEKNIIKAT JA KIRJASTOT

Julkaisutyökalu on .NET Framework-pohjainen ohjelma. .NET Framework on Microsoftin ylläpitämä avoimen lähdekoodin "ohjelmointikehikko", joka tarjoaa valmiin toteutuksen monelle hyötyohjelmissa yleisesti tarvittavalle toiminnolle. .NET Framework-ohjelmia voidaan kirjoittaa millä vain ohjelmointikielellä, joka kääntyy Common Intermediate Languageksi (CIL). Näihin kuuluvat mm. C#, Visual Basic, F# ja C++.

Julkaisutyökalun toimintalogiikka on kirjoitettu C#:lla. C# on Microsoftin kehittämä objektipohjainen, yleiskäyttöinen ohjelmointikieli, jota käytetään mm. silloin, kun ohjelmistokehityksessä halutaan hyödyntää jotain osaa .NET Frameworkista tai sitä ympäröivästä ekosysteemistä. Julkaisutyökalun käyttöliittymä on toteutettu Windows Presentation Foundationilla (WPF), joka on .NET Framework-ohjelmien kanssa yhteensopiva järjestelmä käyttöliittymien määrittelyyn. WPF:ssä käyttöliittymä määritellään XAML-merkintäkielellä, ja logiikka kirjoitetaan millä vain .NET Frameworkia tukevalla ohjelmointikielellä.

Kehitettävä työkalu hyödyntää laajasti M-Files-palvelinten tarjoilemaa Component Object Model- rajapintaa (COM API). Component object model on Microsoftin tuottama standardi ohjelmien välisten kommunikaatiorajapintojen määrittelyyn. Rajapinta kuvailee mm. M-Files-palvelimen käyttämät tietorakenteet ja tarjoaa mahdollisuuden käskyttää palvelinta julkaisutyökalun kautta.

Lisäksi Julkaisutyökalussa hyödynnetään Vault Application Framework-kirjastoa (VAF). VAF on M-Filesin tuottama C#-luokkakirjasto, joka helpottaa useiden COM-rajapinnan käyttöä, tarjoamalla helppokäyttöisempiä abstraktioita COM-rajapinnan matalatasoisten toimintojen päälle.

Koodieditoriksi muutosten tekemiseen valitsin Visual Studio Community 2017:n. Visual Studio tarjoaa lukuisia helppokäyttötoimintoja .NET Framework-ohjelmien toteuttamiselle.

4 TEHTÄVÄNANTO JA TOTEUTUSSUUNNITELMA

M-Files Oy:n tuottamassa vaatimusmäärittelyssä on kaksi kokonaisuutta. Ensimmäinen kokonaisuus on rakenteellisen tuen lisääminen verkkoyhteydettömille operaatioille, kuitenkaan rikkomatta julkaisutyökalun nykyisiä operaatioita. Julkaisutyökalun alkuperäinen toteutus on tehty sillä olettamuksella, että kaikki operaatiot käyttävät verkkoyhteyttä. Offline-tuen osalta on toivottavaa, että käyttäjältä vaaditut manuaaliset toimenpiteet pysyisivät yhtä vähäisinä kuin online-operaatioissakin.

Toinen kokonaisuus on tiettyjen "offline-operaatioiden" toteuttaminen. Nämä operaatiot ovat varastoreplikan siirto (eng. Replica Transfer), varastoapplikaation asennus (eng. Application Install) ja nimettyjen arvojen siirto (eng. Named Value Transfer). Nämä operaatiot käydään tarkemmin läpi kappaleessa 4.4.

Sain ohjaajaltani ehdotuksia mahdollisiin toteutustapoihin, sekä taustoja käyttötapauksien muodossa, mutta lopulliset valinnat jätettiin minun tehtäväkseni. Ohjelman päivityksessä ei edellytetä minkään tyylioppaan noudattamista, mutta tavoitteena on samankaltainen sävy, jota muualla ohjelmassa on käytetty.

4.1 Julkaisutyökalu

Julkaisutyökalu on C#:lla kirjoitettu ohjelma, jota voidaan käyttää komentorivin kautta, tai WPF-pohjaisen käyttöliittymän avulla.

Julkaisutyökalun toiminta perustuu ns. ”operaatioiden” käyttöön. Jokainen operaatio kohdistuu yhteen varastopariin. Varastoparissa on lähde- ja kohdevarasto. Operaatio hakee tiettyä dataa lähdevarastosta, ja siirtää sen kohdevarastoon. Tyypillisessä käyttötapauksessa julkaisutyökalua käytetään testivarastoon tehtyjen muutosten siirtämiseksi tuotantovarastoon. Tällaisessa skenaariossa testivarasto on lähdevarasto, ja

tuotantovarasto on kohdevarasto. Varastot sijaitsevat tyypillisesti eri palvelimilla, ja siirto tapahtuu julkaisutyökalun varastoihin ottamien internet-yhteyksien avulla.

Ohjelmatasolla operaatio on luokka, joka on periytetty julkaisutyökalulle keskeisestä VaultOperation-luokasta. Jokainen operaatio toteuttaa VaultOperationilta perityn Execute-metodin, jonka sisällä määritellään kunkin operaation toimintalogiikka. Ajon aikana julkaisutyökalua hallinnoiva RunManager-luokka suorituttaa jonossa käyttäjän valitsemien operaatioiden Execute-metodit. Toimintalogiikan lisäksi operaation instanssi pitää sisällään kaiken oheistiedon, jota operaation suorittamiseen tarvitaan.

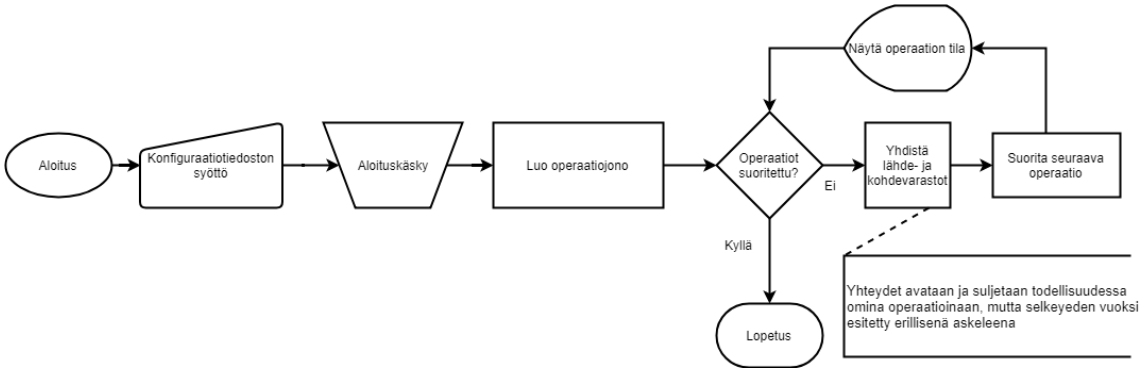
Operaatiot koostavat M-Files-palvelimen COM-rajapinnan toimintoja siten, että jokainen operaatio suorittaa tiettyyn varaston elementtiin kohdistuvan vientityön, ja kohdevarastossa vastaavan tuontityön. Yleensä julkaisutyökalulla ei muokata dataa, vaan se toimii ainoastaan välikätenä datan siirrossa. Jos operaation vientivaiheessa tuotetaan suuri määrä dataa, kuten esim. varaston kokonaisvaltaisessa replikoinnissa, julkaisutyökalu konfiguroidaan tyypillisesti siirtämään data tilapäiseen säilöön verkkolevylle, johon sekä lähde- että kohdevarastolla on näkyvyys. Pienemmän datamäärän, kuten yksittäisten avain-arvoparien kanssa, julkaisutyökalu säilyttää data omassa välimuistissaan, kunnes siirto kohdevarastoon on tehty.

Ennen operaatioiden ajamista, operaatiot konfiguroidaan, eli niille annetaan suorittamisen kannalta oleelliset ennakkotiedot. Esimerkiksi lähde- ja kohdevaraston internet-osoitteet täytyy määritellä ennen minkään operaation suorittamista. Operaatioilla saattaa olla myös muunlaisia parametrejä. Vaikkapa OperationReplicaTransfer, eli jonkin varaston rakenteen kopiointi ja vienti toiseen varastoon, vaatii että lähde- ja kohdevarastoissa on olemassa vienti- ja tuontireplikatyöt, jotka julkaisutyökalulle nimetään. Replikatyöt itsessään määrittelevät mitä tietoa varastosta viedään tai varastoon tuodaan, ja julkaisutyökalu komentaa M-Files-palvelimia suorittamaan työt järjestyksessä.

Konfiguroidut operaatiot voidaan tallentaa ns. konfiguraatiotiedostoon, johon kirjataan operaatioiden asetukset, varastoparit, ja kaikki muu operaatioiden toistamiseen

tarvittava tieto. Konfiguraatioilla varmistutaan siitä, tiedonsiirto kehitysympäristöstä tuotantoympäristöön tapahtuu aina määrätyssä järjestyksessä ja saman prosessin mukaan.

Kuva 2 Kuvaus Julkaisutyökalun toiminnasta, konfiguraation syöttö ja operaatioiden suoritus.



4.2 **Internetittömän käytön edellytykset**

Konseptuaalisesti online- ja offline-operaatioiden toiminta on jaettu kahteen vaiheeseen; lähdevarastoon suunnatut toimet ja kohdevarastoon suunnatut toimet. Online-operaatiot suorittavat peräkkäin molemmat vaiheet, koska niillä on yhteys varastoparin molempiin päihin (Kuva 2). Vaiheita ei rakenteellisesti ole eritelty toisistaan, vaan operaation epäonnistuessa operaatio joudutaan aloittamaan alusta, riippumatta siitä, saatiinko vientivaihe onnistuneesti suoritettua.

Offline-operaatioissa vaiheiden välissä on päätelaitteen vaihto, joten julkaisutyökalulle täytyy jollain tavalla viestiä, kumpi vaihe on suoritettavana. Ohjaajan kanssa keskusteltuani kävi ilmi, että samassa operaatiosarjassa ei tarvitse olla sekaisin vienti- ja tuontivaiheen operaatioita, vaan voidaan olettaa, että kaikki operaatiot kulkevat samassa vaiheessa. Voidaan myös olettaa, että internetiä käyttäviä operaatioita ei ajeta samanaikaisesti internetittömien kanssa.

Verkkokäyttöä silmällä pitäen julkaisutyökalu on suunniteltu niin, että se säilyttää kaiken yksittäisen operaation suorittamiseen tarvittavan tiedon VaultOperation-luokan instanssissa, eli toisin sanoen julkaisutyökalun välimuistissa. Tieto unohdetaan välittömästi, kun operaatio on suoritettu. Jotta operaatiot voidaan suorittaa ilman

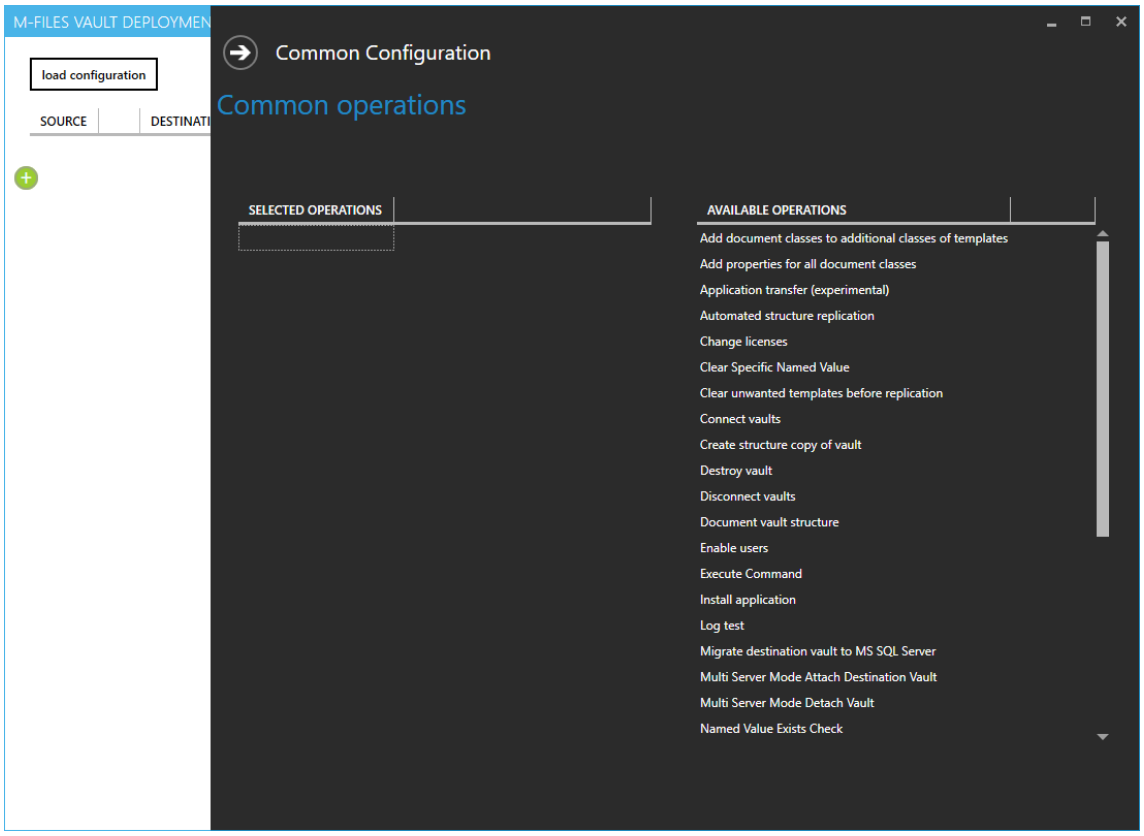
verkkoyhteyttä, täytyy operaation yksittäisen vaiheen aikana tuotettu data tallentaa määräisessä muodossa levypinnalle, jotta data voidaan siirtää fyysisesti eri sijaintiin ja operaatioita jatkaa siitä pisteestä, mihin lähdevarastossa jäätiin.

4.3 **Muutosten sovittaminen ohjelman rakenteeseen**

Ohjaajani ehdotti lähestymistavaksi, että niihin operaatioihin, joihin halutaan offline-tuki, lisättäisiin uusi ns. offline-ajometodi, joka ajettaisiin nykyisen ”Execute”- metodin sijasta silloin, kun julkaisutyökalua käytetään ilman internetiä. Päädyin kuitenkin erilaiseen ratkaisuun, jossa luon täysin uudet luokat niille operaatioille, joille halutaan offline-toimintoja. Valinnan syynä on se, että julkaisutyökalun operaatiot suorittavat hyvin tarkkaan rajattuja toimintoja, ja offline-ominaisuuksien lisäämisellä olemassa oleviin operaatioihin menetettäisiin niiden hienojakoisuus. Lähdekoodin luettavuus pysyy myös selkeämpänä, kun offline-operaatiot on jaoteltu selkeästi omiin luokkiinsa ja vain pitävät sisällään vain omalle toiminnalleen tarpeelliset tiedot.

Julkaisutyökalun graafisessa käyttöliittymässä esitellään käytettävissä olevat operaatiot listamuodossa. Nykyisellään julkaisutyökalu näyttää kaikki operaatiot kerralla, eli mahdolliset offline-operaatiot sekoittuisivat online-operaatioiden kanssa (Kuva 3).

Kuva 3 Näkymä julkaisutyökalun operaatioiden valintaan graafisella käyttöliittymällä.



Ohjaajani selvensi, että online- ja offline-operaatioita ei haluta käyttää sekaisin. Arvioin sen vuoksi, että käyttäjäkokemuksen kannalta selkeintä olisi listata vain yhtä operaatiotyyppiä kerrallaan. Operaatiotyypin vaihto tapahtuu käyttöliittymään lisättävän valintaruutukontrollin kautta. Oletuksena käytössä on online-tila, jolloin operaatiolistasta suodatetaan pois offline-operaatiot, ja muilta osin ohjelma toimii samalla tavalla entiseen nähden. Kun ohjelma vaihdetaan offline-tilaan, operaatiolistaan tuodaan näkyville vain offline-operaatiot.

Kaikki operaatiot jaottelevat toimintansa konseptuaalisesti vienti- ja tuontivaiheisiin. Online-operaatioissa julkaisutyökalu pystyy suorittamaan operaation vaiheet peräkkäin, koska sillä on yhteys molempiin päätelaitteisiin samaan aikaan. Offline-tilassa operaatiot luonnollisesti keskeytyvät vientivaiheen jälkeen, ja ne pitää voida aloittaa uudelleen toisella päätelaitteella siitä pisteestä johon vientivaiheessa jäätiin. Tämä saavutetaan niin, että konfiguraatiotiedostoon kirjataan offline-operaatioiden vaihe. Operaatiot tietävät kumpi vaihe tulee suorittaa seuraamalla tätä arvoa.

Kun käyttäjän luoma vientivaiheen offline-operaatiojono ajetaan, ohjelma luo uuden projektihakemiston joko käyttäjän asettamaan polkuun, tai suhteessa julkaisutyökalun sijaintiin. Projektihakemistoon tuotetaan automaattisesti uusi konfiguraatiotiedosto, johon tallennetaan tiedot suoritetuista operaatioista. Operaatiot tallentavat omat tiedostonsa projektihakemiston alahakemistoihin, ja merkkaavat luotujen hakemistojen nimet konfiguraatiotiedostoon. Kun vientioperaatiot on suoritettu, konfiguraatioon merkataan vaiheeksi import. Vientivaiheen tuloksena on siis hakemisto, joka pitää sisällään kaiken viennissä tuotetun datan, sekä konfiguraatiotiedoston, joka kuvailee vientivaiheessa suoritetut operaatiot ja ohjaa julkaisutyökalun oikeille jäljille, kun konfiguraatio ladataan tuontivaiheessa.

Tuontivaihe alkaa siitä, kun projektihakemisto on kuljetettu massamuistilaitteella seuraavalle päätelaitteelle, ja vientityön tuloksena tuotettu konfiguraatio luettu julkaisutyökalulla. Konfiguraatiosta julkaisutyökalu saa selville edellisessä vaiheessa suoritetut operaatiot, sekä niihin kuuluvan datan tallennuspaikat. Ennen töiden aloittamista, on vielä mahdollista hienosäätää operaatioiden asetuksia, eli jos käyttäjä ei vaikkapa tiennyt etukäteen päätepalvelimen varastojen nimiä, ne voidaan täyttää tässäkin vaiheessa. Tämän ominaisuuden tekemiseen ei sinänsä vaadita lisätyötä, vaan julkaisutyökalun alkuperäinen toimintatapa mahdollistaa sen.

Vientitulokset tallennetaan aina projektihakemistoon yhdessä konfiguraation kanssa, sen sijaan että käytettäisiin ns. absoluuttista polkua. Käyttäjä joutuu aina syöttämään käsin konfiguraatiotiedoston, ja kun vientidata sijaitsee ennalta-arvattavassa sijainnissa konfiguraatioon nähden, ohjelma pystyy itse päättelemään missä asiaan kuuluvat tiedot sijaitsevat, eikä käyttäjän tarvitse määritellä tätä tietoa itse.

4.4 **Suunnitelmat uusille operaatioille**

Suunnitelma applikaation asennusoperaation offline-versiolle muodostui ehkä kaikista helpoiten. Operaatio poikkeaa muista siten, että sen online-versiokaan ei käytä tiedonsiirtoon suoraa verkkoyhteyttä kahden varaston välillä. Sen sijaan operaatio edellyttää, että sekä lähde- että kohdepalvelimilla on pääsy samaan jaettuun verkkokansioon. Julkaisutyökalu kopioi lähdevarastosta applikaation, ja siirtää sen jaettuun kansioon. Sitten

julkaisutyökalu ottaa yhteyden kohdevarastoon, ja komentaa sen asentamaan applikaation jaetusta kansiosta. Voidaan siis sanoa, että operaatiolla oli jo valmiiksi ”vientivaihe”, jossa haluttu data etsitään lähdelaitteen massamuistista, siirretään ennalta päätettyyn sijaintiin, ja myöhemmin haetaan samasta sijainnista kohdelaitteella.

Internetittömän asennuksen vientivaiheessa siispä riittää, että applikaatio kopioidaan projektihakemiston alakansioon, ja alakansion nimi talletetaan vientivaiheen konfiguraatiotiedostoon. Offline-applikaatioasennuksen tuontivaihe taas muodostuu siten, että julkaisutyökalu lukee ajon aikana applikaation sijainnin konfiguraatiotiedostosta. Kun käyttäjä on määritellyt julkaisutyökalun kautta varastot, joita vasten asennusoperaatio ajetaan, operaatio käyttäytyy identtisesti online-varianttinsa kanssa, eli asentaa jokaisen applikaation, jonka se löytää alakansiosta.

Nimetyn arvon siirto-operaatio, eli Operation Named Value Transfer käsittelee jokaiselle varastolle ominaista named value storagea (NVS). NVS on varaston metatietorakenteesta erillinen tallennuspaikka. NVS:ään tallennetaan tyypillisesti varaston ominaisuuksiin liittyviä asetuksia. Lisäksi varastossa toimivat laajennusapplikaatiot voivat tallentaa NVS:ään tietoja, joita ei haluta säilyttää applikaation välimuistissa. NVS rakentuu yhdestä tai useammasta nimiavaruudesta. Nimiavaruudet ovat tunnisteita avain-arvoparelle. Nimiavaruuksien ja nimiavaruuden sisällä olevien avaimien täytyy olla uniikkeja, mutta eri nimiavaruuksissa voi olla samannimisiä avaimia.

Operaation tavoite on kopioida lähdevaraston tietystä nimiavaruudesta tietty avain-arvopari, ja tallentaa se kohdevarastossa samaan sijaintiin. Suunnitelmani on tallentaa operaation vientivaiheen tulokset JavaScript Object Notation-formaattiin (JSON). Julkaisutyökalussa on jo käytössä kirjasto JSONin käsittelyyn, joten tällä menettelyllä projektiin ei tarvitse lisätä uusia riippuvuuksia. Tulokset talletetaan projektihakemiston alakansioon, siten että tiedoston nimeksi tulee nimiavaruuden nimi, JSONin juurielementiksi avain-arvoparin avain, ja juuriobjektin arvoksi avain-arvoparin arvo. Tuloksien kääntäminen takaisin nimiavaruus -> avain -> arvo- muottiin kohdelaitteella on tällä kaavalla yksinkertaista. Jos konfiguraatiossa on useampia nimetyn arvon siirto-operaatioita, jokainen operaatio tallettaa tuloksensa eri alihakemistoon.

Varaston replikointioperaation online-variantti pohjaa toimintansa M-Files-palvelimelle etukäteen luotuihin replikaatiotöihin. Julkaisutyökalulle annettavassa operaatioiden konfiguraatiotiedostossa määritetään lähdevarastossa ajettavat replikan vientityöt ja kohdevarastossa ajettavat replikan tuontityöt. Kehitys- ja tuotantoympäristöt ovat etukäteen säädetty niin, että niillä on pääsy yhteiseen verkkosijaintiin. Julkaisutyökalu käskee lähdepalvelinta suorittamaan omat vientityönsä, joiden tulokset tallentuvat verkkolevylle. Kun lähteessä tehtävät replikat on suoritettu, ajetaan kohdevarastossa ennalta valitut replikan tuontioperaatiot, jotka kohdistuvat jälleen samaan verkkosijaintiin.

Julkaisutyökalu ei ole itse vastuussa siirron tekemisestä. Työkalu tarkistaa nimen perusteella, että määrätyt replikointityöt ovat olemassa lähde- ja kohdevarastoissa, jonka jälkeen palvelimia käskytetään aloittamaan replikointityönsä järjestyksessä, ja palvelimet suorittavat tehtävät siitä eteenpäin itsenäisesti. Jos replikointioperaatioissa ilmenee virhe, julkaisutyökalu keskeyttää replikointiin liittyvät työt kokonaan, ja ilmoittaa asiasta omissa lokitiedoissaan.

Ensimmäinen mieleeni tullut tapa toteuttaa replikoiden vienti ja tuonti offline-tilassa jäljittelee online-version menetelmää. Vientipäähän luodaan replikan vientityö, jonka tallennuskohteena on massamuistilaite, jolla replika on tarkoitus kuljettaa tuontipäähän. Tuontipäähän luodaan replikan tuontityö, jolle annetaan lähteeksi hakemisto massamuistilaitteella. Tässä lähestymistavassa on ilmeisiä ongelmia; massamuistilaitteen levykirjain on mahdoton tietää ennen tietokoneeseen laittamista, joten kunkin palvelimen replikatöitä täytyisi muokata ennen julkaisutyökalun käyttöä niin, että ne vastaavat massamuistilaitteen nykyistä levykirjainta. Replikatöiden muokkaaminen ohjelmallisesti COM APIn kautta ennen ajoa ei myöskään ole mahdollista.

Vientivaiheen ongelmaan löytyi pienen pohdinnan jälkeen yksinkertainen ratkaisu; Replikatyön vientivaihe säädetään M-Filesissä osoittamaan tiettyyn hakemistoon paikallisen tietokoneen levypinnalla. Julkaisutyökalulle kerrotaan replikointiin käytettävä polku, ja työkalu kopioi replikoinnin jälkeen tulokset projektihakemistoonsa. Jos projektihakemisto on jo valmiiksi massamuistilaitteella, käyttäjältä ei vaadita muita toimia. Jos

projektihakemistokin on paikallisen tietokoneen levypinnalla, käyttäjän täytyy lopuksi siirtää koko kansio massamuistilaitteellensa. Näin saadaan joka tapauksessa pidettyä käsin tehtävä osuus mahdollisimman lyhyenä.

Tuontivaihe noudattaa samoja askelia käänteisessä järjestyksessä; kohdelaitteen M-Files-iin säädetään replikoiden tuontityö, joka käyttää lähteenään tiettyä polkua palvelimen omalla levypinnalla. Sama polku kerrotaan julkaisutyökalulle. Julkaisutyökalu etsii projektihakemistosta tuontityötä vastaavan replikapaketin, ja kopioi sen tuonnissa käytettävään polkuun palvelimella. Kun kopiointi on tehty, julkaisutyökalu käskyttää kohdevarastoa suorittamaan replikan tuontityön. Tällä menetelmällä siirtotyö pelkistyy pienimmillään neljään käyttäjän suorittamaan vaiheeseen:

- Vientikonfiguraation syöttö
- Vientivaiheen suoritus
- Päätelaitteen vaihto
- Tuontikonfiguraation syöttö
- Tuontikonfiguraation suoritus

5 TOTEUTUKSET

Päätin aloittaa toteutuksen ohjelman rakennemuutoksista, joita offline-operaatioiden ajamiseksi yleisesti tarvitaan. Tähän kategoriaan kuuluvat graafisen käyttöliittymän muutokset, konfiguraatiotiedostojen lukeminen ja kirjoitus, sekä Julkaisutyökalulle keskeisen VaultOperation-luokan muutokset. Ensiksi lähdin muokkaamaan VaultOperationia. Kaikki ohjelman operaatiot periytyvät tästä luokasta. Se määrittelee kaikille operaatioille yhteiset kuvaavat tiedot, kuten operaation nimen ja kuvauksen, jotka näytetään käyttöliittymässä. Lisäksi luokka määrittelee asetuksia, jotka vaikuttavat operaation ajonaikaiseen käyttäytymiseen. Kun offline-toimintaa kuvaavat ominaisuudet lisätään tähän sijaintiin, ohjelman ajonaikaista käyttäytymistä ohjaavat luokat pääsevät siihen helposti käsiksi.

Ensimmäisenä lisäsin isOffline-nimisen Boolen arvon, joka kertoo voiko operaatiota käyttää offline-tilassa vai ei (Kuva 4). Operaatiot, joilla ominaisuus on tosi, ovat käytettävissä offline-tilassa, muut online-tilassa. Vakioarvo on epätosi, jotta toiminnallisuuden lisäys ei häiritse olemassa olevien online-operaatioiden käyttöä. Käytännössä siis kaikki operaatiot ovat oletuksena online-operaatioita, kunnes niiden isOffline-ominaisuus on vaihdettu arvoon true.

Kuva 4 VaultOperation-luokan offline-käytöstä määräävä ominaisuus.

```
/// <summary>
/// Represents whether or not operation is used in offline mode
/// </summary>
6 references | 0 changes | 0 authors, 0 changes
public bool IsOffline { get => isOffline; set => isOffline = value; }
private bool isOffline = false;
```

Tämän jälkeen lisäsin VaultOperationiin uuden arvojoukon OfflinePhases (suom. offline vaiheet), joka listaa käytettävissä olevat offline-operaation vaiheet, export ja import (suom. vienti ja tuonti) (Kuva 5). Tämän ohessa lisäsin myös ominaisuuden Phase (suom.

vaihe), joka kertoo mikä operaation vaihe tulee suorittaa seuraavaksi. Näitä ominaisuuksia käyttämällä ohjataan Operaation ajonaikaista toimintaa.

Kuva 5 Offline-operaatioiden toimintaa ohjaavat ominaisuudet.

```
/// <summary>
/// Used for offline operation flow control, determines which part of <see cref="Execute"/> should be ran.
/// </summary>
6 references | 0 changes | 0 authors, 0 changes
public enum OfflinePhases
{
    Export,
    Import
}

/// <summary>
/// Offline operation's current phase. Available phases listed in <see cref="OfflinePhases"/> enum.
/// </summary>
6 references | 0 changes | 0 authors, 0 changes
public string Phase { get => phase; set => phase = value; }
private string phase = null;
```

Päädyin tekemään Phase-ominaisuudesta merkkijonotyyppisen, sen sijaan että se olisi valinta OfflinePhases-joukosta. Tein päätöksen siksi, että merkkijono, toisin kuin enum, sallii käytettävän tyhjää arvoa. Tällöin online-operaatiot, jotka eivät hyödynnä vaihetietoa mitenkään, voivat jättää ominaisuuden tyhjäksi. Ajattelin aluksi tämän olevan selkeämpi ratkaisu tyylillisesti, mutta jälkikäteen valinta osoittautui epäkäytännölliseksi, sillä erityyppisten ominaisuuksien arvoja täytyi kääntää ristiin (enumista merkkijonoksi, merkkijonosta enumiksi) käytön aikana. Jonkun vakioarvon käyttö phasessa online-operaatioilla ei olisi tuottanut ongelmia, sillä tietoa ei ylipäänsä hyödynnetä online-käytössä millään tavalla.

DataManager-luokka tuottaa käyttöliittymällä esitettävän listauksen operaatioista. DataManager etsii reflektion avulla kaikki luokat, jotka perivät VaultOperation-luokan, tekee yhden ilmentymän jokaisesta, ja säilyttää ilmentymiä Dictionary-tietorakenteessa ajon aikana. Kun operaatioita esitetään käyttöliittymällä, tiedot haetaan tästä tietorakenteesta.

Nimesin Operations-listan uudelleen OnlineOperationsiksi, ja lisäsin toisen Dictionary-tyyppisen ominaisuuden nimeltä OfflineOperations. Muutin DataManagerin listausten täyttölogiikkaa niin, että se erittelee operaatiot kokoelmiin riippuen operaation isOffline-lipun asennosta (Kuva 6).

Kuva 6 Operaatiotyyppien jaottelu tyypin perusteella, LINQ-kyselykieltä käyttäen.

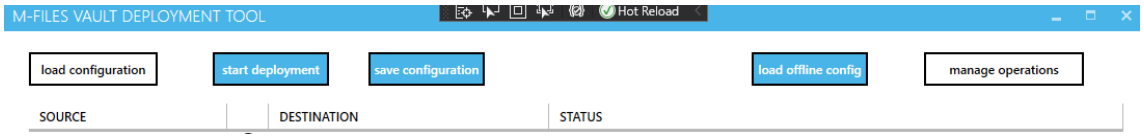
```
AllOperations = AllOperationList
    .Where(type => !type.IsAbstract)
    .Select(type => (VaultOperation)Activator.CreateInstance(type))
    .Where(operation => !string.IsNullOrEmpty(operation.Identifier))
    .OrderBy(operation => operation.Name)
    .ToDictionary(operation => operation.Identifier, operation => operation);

// Populate collection of online operations
OnlineOperations = AllOperations
    .Where(operation => !operation.Value.IsOffline)
    .OrderBy(operation => operation.Value.Name)
    .ToDictionary(operation => operation.Value.Identifier, operation => operation.Value);

// Populate collection of offline operations
OfflineOperations = AllOperations
    .Where(operation => operation.Value.IsOffline)
    .OrderBy(operation => operation.Value.Name)
    .ToDictionary(operation => operation.Value.Identifier, operation => operation.Value);
```

Lisäsin käyttöliittymään uuden napin, "load offline config", jonka kautta offline-asetustiedosto luetaan (Kuva 7). Nappi jäljittelee alkuperäisen "load configuration" napin toimintaa, eli se avaa Windowsin tiedostovalitsimen, jonka kautta käyttäjä syöttää haluamansa konfiguraation ohjelmalle. Lisäys tapahtui applikaation MainWindow.xaml-tiedostoon, jossa muukin käyttöliittymä on määritelty. Napin painallukseen liitin funktion, joka tiedottaa DataManagerille, että tuleva konfiguraatio tulee käsitellä eri tavalla kuin normaalisti. Suunnitelmasta poiketen valitsin valintaruudun sijasta napin tätä varten, sillä toiminnallisuuden ohjelmointi on huomattavasti yksinkertaisempaa, kun ei tarvitse ottaa huomioon tilanteita, joissa käyttäjä vaihtaa toimintatilaa lennossa. Ohjelman kannalta voidaan aloittaa aina puhtaalta pöydältä uutta konfiguraatiotiedostoa luettaessa.

Kuva 7 Käyttöliittymään lisätty painike "load offline config".



Riippuen kummasta napista konfiguraatio ladataan, ohjelma näyttää käyttöliittymällä joko online- tai offline-operaatiot. operaatiot. Jos käyttäjä lataa uudet asetukset "load configuration"-napista (alkuperäinen konfiguraation latausnappi), listaus vaihtuu jälleen näyttämään online-operaatiot (Kuva 7).

5.1 **Offline-konfiguraation lukeminen ja tallentaminen**

Vientityökalun konfiguraatiotiedosto koostuu serialisoiduista operaatioista, ynnä vähemmän erikoistetuista yhteiskonfiguraatioista. Serialisointi tarkoittaa luokan ilmentymän esittämistä tekstimuodossa. Käyttöliittymässä säädetyt operaatiot ja asetukset muutetaan tekstiksi, ja tallennetaan konfiguraatiotiedoston. Käänteisesti, kun tiedosto ladataan, tekstimuotoiset oliot käännetään vastaaviksi olioiksi ohjelman käyttöön.

Serialisointilogiikka ei tarvitse muutoksia - olemassa olevien rakenteiden järkevä käyttö riittää offline-operaatioiden tietojen säilömiseen. Ohjelma sisältää korkean tason yhteiskonfiguraatio-objektin, jossa olevat tiedot vaikuttavat kaikkien operaatioiden ajoon. Lisäksi jokainen operaatio listaa itsensä kannalta oleelliset asetukset. Serialisaattori käy sekä yhteiskonfiguraation, että kunkin operaation omat asetukset läpi, ja tallettaa ne tekstitiedostoon. Operaatiot voivat siispä listata itse tarvitsemansa asetukset, ja ne kirjoitetaan automaattisesti konfiguraatiotiedostoon serialisoinnin yhteydessä. Konfiguraatioita voidaan käyttää myös ajon aikana tuotetun tiedon säilömiseen - tieto lisätään konfiguraatio- tai operaatio-objektiin ennen serialisointia. Näin voidaan parhaiten hyödyntää ohjelman valmiita rakenteita.

Offline-konfiguraatioiden eriyttäminen tavallisista konfiguraatioista alkoi DataManager-luokan Initialize-metodista. DataManager on keskeinen luokka, joka pitää hallussaan ladatun konfiguraation ja käyttäjän asettamat operaatiot. Kiteytettynä se on luokka, joka hallinnoi tietoa, jonka ohjelma tarvitsee ennen operaatioiden ajoa.

Initialize-metodin alkuperäisessä toteutuksessa ainoana parametrina vaadittiin configPath-merkkijono, eli konfiguraatiotiedoston sijainti. Lisäsin parametrin *online*, joka kertoo metodille, miten sisään tuleva tiedosto tulee käsitellä. Oletusarvoisesti arvo on tosi (Kuva 8). Tällöin ohjelma käyttäytyy täsmälleen kuin aiemminkin. Kun arvo on epätosi, ohjelman käytökseen tulee tiettyjä poikkeuksia. Initializea kutsutaan aina kun yhtä konfiguraation latausnapeista painetaan, ja kun offline-latausnappia on käytetty, online-parametrin arvoksi annetaan false.

Kuva 8 Initialize-metodin parametrien muutos.

```
/// <summary>
/// Reads the given configuration
/// </summary>
/// <param name="configPath"></param>
3 references | 0 changes | 0 authors, 0 changes
public static void Initialize(string configPath, bool online = true)
{
```

Serialisointi ja deserialisointi tehdään Newtonsoft.JSON-kirjastolla. Kirjasto kääntää konfiguraatiotiedoston tekstisisällön yhdeksi JOBject-tyypin olioksi muuttujaan *config*. Config sisältää kaikki operaatiot asetuksineen, sekä ohjelman yleisasetukset. Tämä on ohjelman normaalia toimintaa, eikä siihen tarvitse puuttua, sillä ohjelma pystyy käsittelemään tällä mekanismilla myös offline-operaatioita.

Ainoa tarvittava lisäys on offline-operaation vaiheen tulkinta. Jos konfiguraatiota käsitellään offline-konfiguraationa, tarkistetaan, onko "Offline phase"-asetukselle annettu jokin arvo. Jos arvoa ei ole, oletetaan että kyseessä on vientivaihe. Kun arvo on asetettu, sitä käytetään sellaisenaan. Offline phasen arvo kirjoitetaan staattisen InstallationConfiguration-luokan OfflinePhase-ominaisuuteen. Operaatiot lukevat sieltä ajon aikana vaihetiedon (Kuva 9).

Kuva 9 Offline-konfiguraatioiden luku ja poikkeuskäsittely.

```
// Read configuration file
JObject config = JObject.Parse(File.ReadAllText(configPath));

if (!online)
{
    // set offline config path
    OfflineConfigFileLocation = configPath;
    // check if config contains import phase key
    if (config.ContainsKey("Offline Phase"))
    {
        // phase is set, use value from config
        InstallationConfiguration.OfflinePhase = config["Offline Phase"].ToString();
    }
    else
    {
        // no phase, assume export phase
        InstallationConfiguration.OfflinePhase = VaultOperation.OfflinePhases.Export.ToString();
    }
}
```

Kun operaatiot on ajettu, RunManager päättelee DataManagerin Online-ominaisuudesta, olivatko juuri ajetut operaatiot offline-operaatioita. Jos olivat, Export-vaiheen jälkeen RunManager kutsuu DataManagerille luotua WriteImportConfig-metodia. WriteImportConfig kirjoittaa samanlaisen konfiguraatiotiedoston kuin Julkaisutyökalu normaalistikin kirjoittaa, mutta tekee sen uuteen tiedostoon alkuperäisen konfiguraation rinnalle. Konfiguraatiotiedostoon on listattu operaatioiden tuotokset. Uusi konfiguraatiotiedosto ohjaa toimintaa vientivaiheessa. Jos taas vaihe on Import, operaatioiden tiedot vain luetaan ja suoritetaan - kaikki tarvittava tieto on Julkaisutyökalun saatavilla.

5.2 **Operaatiot**

Aikeeni oli lähteä toteuttamaan ensiksi applikaation siirto-operaatiota, jonka olin arvioinut suunnitteluvaiheessa yksinkertaisimmaksi. Pian työn alettua huomasin kuitenkin, että vaatimusmäärittelystä oli jäänyt kokonaan huomioimatta ehkä tärkein operaatio, eli varastojen yhdistämisoperaatio. Alkuperäinen julkaisutyökalu sisältää operaation nimeltä OperationConnectVaults, joka avaa yhteydet lähde- ja kohdevarastoon. Myöhemmät operaatiot hyödyntävät avattuja yhteyksiä tarpeen mukaan.

Offline-toiminnassa tämä ei luonnollisesti toimi, sillä yhteys on avattavissa vain yhteen varastoon kerrallaan. Ongelman ratkaisemiseksi loin neljännen uuden offline-operaation, "OperationConnectVaultsOfflinen". Operaatio on muilta osin identtinen online-variantin kanssa, mutta ottaa yhteyden vain varastoparin yhteen päähän kerrallaan, riippuen vaiheesta. Näin ollen myöskään molempien varastojen tietoja ei tarvitse tietää etukäteen; riittää että vain nykyisen siirtovaiheen varaston tiedot ovat oikeelliset. Varastoparin toinen puoli täytetään "tyhjällä varastolla" joka täyttää ohjelman vaatimat määritykset, mutta ei ole käytännössä toiminnallinen.

Ongelmasta selvittiin melko pienellä korjausliikkeellä, kiitos ohjaajan tarkennuksen, että voidaan olettaa tehtävän vain yhden siirtovaiheen operaatioita kerrallaan. Muutoin järjestelmään olisi tarvinnut luoda tuki myös useiden varastoyhteyksien ja operaatiovaiheiden ristikäytölle, laajentaen rakenteellisten muutosten määrää huomattavasti.

Tämän jälkeen palasin applikaation siirto-operaatioon. Operaatio tarvitsee kaksi parametria; kansion, josta vientivaiheessa kopioidaan varastoapplikaatiot, sekä kansion projektihakemistossa, johon applikaatiot säilötään siirtoa varten (Kuva 10). Lisäsin k.o. konfiguraatiot käyttäen VaultOperation-luokan ominaisuutta Configuration, joka listaa kaikki operaation tarvitsemat asetukset. Käyttäjä voi antaa asetuksille arvon työkalun käyttöliittymästä, ja ne päätyvät automattisesti konfiguraatiotiedostoon ilman muita toimenpiteitä. Ohjelma ymmärtää kuinka tässä formaatissa tehdyt konfiguraatiot tulisi esittää käyttöliittymällä, ja kuinka niiden arvot tulee kirjoittaa konfiguraatiotiedostoon.

Kuva 10 OperationApplicationTransferOfflinen asetukset.

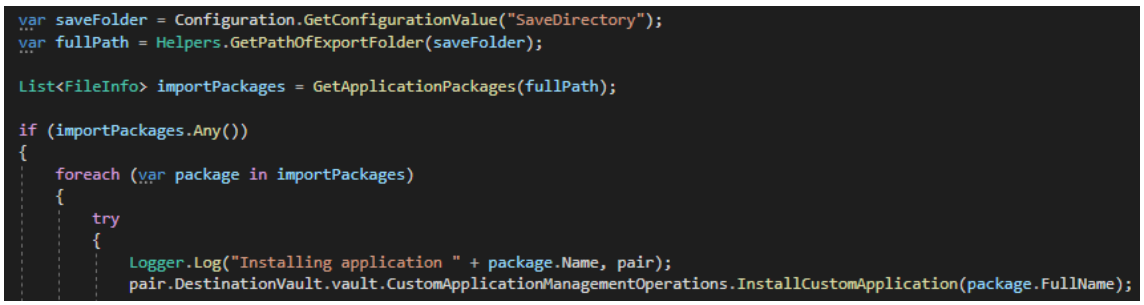
```
// Find application packages from this path
Configuration.AddConfiguration("VaultApplicationsPath", "C:\\Applications", true);

// THe location for copied applications is created by the tool in import phase
Configuration.AddConfiguration("SaveDirectory", "Autofilled when operation starts");
```

Vientivaiheessa applikaatio luo uuden kansion projektihakemistoon. Hakemiston nimi asetetaan SaveDirectoryn arvoksi. Hakemiston nimi on uusi GUID (Globally Unique Identifier), eli satunnainen tunniste, joka koostuu alfanumeerisista merkeistä. GUIDeja käyttämällä vältytään nimeämislogiikan keksimiseltä siinä tapauksessa, että operaatiojonossa on useita applikaation kopiointioperaatioita. Esim. silloin, kun operaatiojonossa otetaan sarjassa useisiin varastoihin yhteyksiä, ja jokaiseen kopioidaan eri applikaatiot, applikaatioita ei voida tallettaa samaan hakemistoon. Kaikki zip- ja mfappx-päätteiset tiedostot (molemmat käypiä varastoapplikaation formaatteja) kopioidaan VaultApplicationsPathista SaveDirectoryyn, ja vientivaihe on valmis.

Tuontivaiheen alussa luetaan konfiguraatiotiedostosta SaveDirectoryn arvo. Se lisätään konfiguraatiotiedoston sijaintiin (projektihakemisto), ja näin saadaan täysmittainen polku hakemistoon, jossa tämän operaation vientivaiheen applikaatiot olevat. Hakemistosta etsitään jälleen kaikki zip- ja mfappx päätteiset tiedostot, ja ne syötetään nykyisen varastoparin kohdevarastolle. Varaston COM-rajapinnan kutsua InstallCustomApplication käytetään kunkin paketin asentamiseksi varastoon. Jos virheitä ilmenee asennuksessa, ne ilmaistaan käyttäjälle julkaisutyökalun olemassa olevaa lokinpitojärjestelmää, joka näyttää mahdolliset virheilmoitukset ohjelman käyttöliittymällä.

Kuva 11 OperationApplicationTransferOfflinen tuontivaihe.



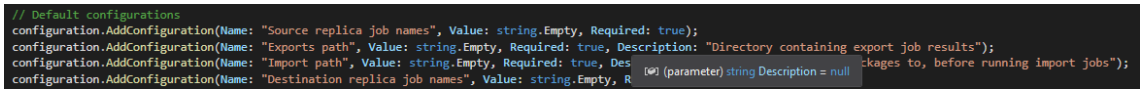
```
var saveFolder = Configuration.GetConfigurationValue("SaveDirectory");
var fullPath = Helpers.GetPathOfExportFolder(saveFolder);

List<FileInfo> importPackages = GetApplicationPackages(fullPath);

if (importPackages.Any())
{
    foreach (var package in importPackages)
    {
        try
        {
            Logger.Log("Installing application " + package.Name, pair);
            pair.DestinationVault.vault.CustomApplicationManagementOperations.InstallCustomApplication(package.FullName);
```

Replikan siirto-operaatiota varten loin lähdekoodiin uuden luokan, "OperationReplicaTransferOffline". Luokan konfiguraatiot ovat suunnitelman mukaiset; lähdevarastossa ajettavien replikan vientitöiden nimet puolipisteellä erotettuna, juurihakemisto, johon kaikki replikatyöt tuottavat replikat, kohdevarastossa ajettavien tuontitöiden nimet puolipisteellä erotettuna, sekä polku johon vientivaiheen replikapaketit tulee siirtää ennen tuontitöiden ajamista (Kuva 12).

Kuva 12 OperationReplicaTransferille annetut asetukset.



```
// Default configurations
configuration.AddConfiguration(Name: "Source replica job names", Value: string.Empty, Required: true);
configuration.AddConfiguration(Name: "Exports path", Value: string.Empty, Required: true, Description: "Directory containing export job results");
configuration.AddConfiguration(Name: "Import path", Value: string.Empty, Required: true, Des  (parameter) string Description = null  ckages to, before running import jobs");
configuration.AddConfiguration(Name: "Destination replica job names", Value: string.Empty, R
```

Vientivaiheessa vientivarastoa käskytetään ajamaan käyttäjän määrittelemät replikointityöt. Tämä toiminnallisuus löytyi jo OperationReplicaTransferiin, eli tämän operaation online-varianttiin, RunReplicaJobs-metodista. Kopioin kyseisen funktion myös uuteen operaatioon. Funktion olisi myös voinut muuttaa staattiseksi, jolloin sitä ei olisi tarvinnut kahdentaa lähdekoodissa, mutta päädyin tähän ratkaisuun olettamuksella, että funktiota saattaa joutua muokkaamaan offline-toimintaa varten, jolloin on järkevämpää, että funktion toiminta määritellään yksilöllisesti kullekin luokalle, joka sitä käyttää.

Kun replikointityöt on suoritettu, julkaisutyökalu kopioi replikoinnin juurihakemiston paikalliselta tietokoneelta sijainnista, jonka käyttäjä konfiguroi "Exports path"-konfiguraatioon. Hakemisto kopioidaan projektihakemiston alle uuteen alihakemistoon, jonka nimeksi tulee uusi GUID. Tällöin käyttäjän ei tarvitse itse varmistaa, että replikoinnin juurihakemistot eri replikansiirto-operaatioiden välillä eivät ole samannimisiä, eikä

ohjelman tarvitse ottaa kantaa siihen, mitä tapahtuu, jos kaksi hakemistoa ovatkin samannimisiä (Kuva 13).

Kuva 13 Replikan siirto-operaation tulosten tallennus

```
// get configured replication path
var contentPackageDir = configuration.GetConfigurationValue("Exports path");
// create path for new sub-directory in project directory to store replication content packages
// use guid instead of the bare name of the root folder to avoid name conflicts
var destinationDir = Path.Combine(DataManager.OfflineConfigFileLocation, new Guid().ToString());
Directory.CreateDirectory(destinationDir);
FileSystem.CopyDirectory(sourceDirectoryName: contentPackageDir, destinationDirectoryName: destinationDir);

// inject new configuration item to config for the results of this replica run
configuration.AddConfiguration(Name: "StorageDirectory", Value: destinationDir);
```

Tuontivaihe aloitetaan tarkistamalla, että replikointisisältö löytyy odotetusta sijainnista projektihakemistosta, eli vientivaiheessa luodun talletushakemiston sisältä. Tässä vaiheessa käyttäjä on siirtynyt toiselle tietokoneelle, joten ladatun konfiguraatiotiedoston sijainti luetaan DataManagerilta, ja tarkistetaan, että vientivaiheen GUIDilla nimetty kansio löytyy konfiguraatiotiedoston vierestä. Jos kansiota ei löydy, lopetetaan operaation ajo, mutta jätetään julkaisutyökalun lokinpitäjään merkintä poikkeustilanteesta. Kansion löytyessä tarkistetaan vielä, että palvelimelta löytyy konfiguroidut replikan tuontityöt. Jos niitä ei löydy, operaatio katkaistaan ja lokiin jätetään maininta poikkeustilanteesta (Kuva 14).

Kuva 14 Replikasiirron tuontivaiheen ennakkotarkastukset.

```
// check project folder for the content package dir
var storageDir = configuration.GetConfigurationValue("StorageDirectory");
var pathInProjectDir = Path.Combine(DataManager.OfflineConfigFileLocation, storageDir);
if (!Directory.Exists(pathInProjectDir))
{
    Logger.Log("No content packages found in project directory", pair);
    return;
}

// check that configured replication jobs exist
var replicaJobNames = Configuration.GetConfigurationValue("Destination replica job names").Split(';');
var replicaJobIDs = ConvertReplicationJobNamesToIDs(replicaJobNames, pair.DestinationVault.vault);
if (!replicaJobIDs.Any())
{
    Logger.Log("No import replica jobs configured", pair);
    return;
}
```

Ennakkotarkastusten onnistuttua, replikoinnin juurihakemisto kopioidaan konfiguroituun Import path-sijaintiin, josta vastaanottavan palvelimen replikoinnin tuontityöt osaavat etsiä sisältöpaketteja. Jos import pathia ei ole olemassa palvelimella, työkalu luo sen ennen siirtoa. Kopioinnin jälkeen vastaanottava palvelin käskytetään ajamaan

replikapakettien tuontityöt. Töiden ajamiseen käytetään julkaisutyökalussa jo ollutta RunReplicaJobs-funktiota. RunReplicaJobsille annetaan suoritettavien replikatöiden numeeriset tunnisteet, jotka tässä tapauksessa saatiin selville, kun tuontitöiden olemassaolo tarkistettiin (Kuva 15).

Kuva 15 Tuontitöiden ennakkotarkistukset ja tuontitöiden suorituttaminen.

```
var destinationDir = configuration.GetConfigurationValue("Import path");
// make sure import pah exists on server
if (!Directory.Exists(destinationDir))
{
    _ = Directory.CreateDirectory(destinationDir);
}
// copy content packages to import path
FileSystem.CopyDirectory(sourceDirectoryName: pathInProjectDir, destinationDirectoryName: destinationDir);

Logger.Log("Starting destination Replica jobs", pair);
status = "Transferring replica packages to destination";

// execute import jobs on server
RunReplicaJobs(pair.DestinationVault.vault, replicaJobIDs);
Logger.Log("Replica jobs finished", pair);
```

Viimeiseksi lähdin toteuttamaan OperationNamedValueTransferOfflinea. Toteutusvaiheen alussa minulla oli heikoin käsitys tämän operaation toiminnasta, ja toteutus jäikin vaiheeseen; isoimpana hidasteena oli COM-rajapinnan epäselkeys Named Value Storagen (NVS) ympärillä. Törmäsin puutteisiin sekä COM-rajapinnan dokumentaatiossa, että VAFin NVS-abstraktioiden dokumentaatiossa. En saanut riittävää kokonaiskuvaa NVS:n toiminnasta, että olisin saanut operaation valmiiksi jäljellä olevan ajan puitteissa. Kävimme ohjaajani kanssa muutamia keskusteluja aiheesta, mutta en päässyt niidenkään avulla riittävästi eteenpäin.

Osasyynä oli ehdottomasti myös jääräpäisyys omien oppimistavoitteitteni kanssa; en halunnut käyttää pohjana alkuperäistä OperationNVSTransferia ymmärtämättä sen toimintalogiikkaa. Jos olisin taipunut tekemään ns. kopioi-liitä ohjelmointia, luulen että olisin saanut toteutettua kaikki halutut ominaisuudet. Tein kuitenkin applikaatioon OperationNVSTransferOffline-luokan, jota voidaan käyttää pohjana jatkokehitykselle.

6 YHTEENVETO

Suunnitteluvaiheessa tehdyt ratkaisut olivat toimivia ja päätyivät Julkaisutyökaluun suurelta osin ilman muutoksia. Toteutusvaiheessa muutettiin tiettyjä yksityiskohtia paremman käyttäjäkokemuksen saavuttamiseksi, mutta suuret linjat pitivät kutinsa. Tästä esimerkkinä mm. online- ja offline-tilojen vaihto konfiguraation avausnappia käyttämällä, sen sijaan että vaihto tapahtuisi rasti ruutuun-valinnalla, kuten alun perin oli suunniteltu.

Työ onnistui mielestäni hyvin, vaikka kaikkiin tavoitteisiin ei päästykään. Sain kehitettyä Julkaisutyökaluun toimivat raamit offline-operaatioiden ohjelmoinnille ja ajamiselle. Operaatioita ohjelmoidessa sain laajaa tuntemusta M-Filesin COM-rajapinnasta, sekä hyvistä että huonoista puolista. Kesken jääneen operaation viimeistelyn pitäisi olla helpohko työ sellaiselle, joka tuntee rajapinnan NVS:n osalta paremmin kuin minä. Muiden operaatioiden kohdalla jouduin ratkomaan haastavia mutta mielekkäitä ongelmia yksinkertaisen käyttäjäkokemuksen säilyttämiseksi. Koen onnistuneeni etenkin replikan siirto-operaation kanssa. Operaatiossa oli lähtökohtaisesti suhteellisen tiukat rajoitteet, koska palvelimen replikointitöiden säätöihin ei pystytä vaikuttamaan ajon aikana, ja rajoitteiden puitteissa suunnitteleminen kannusti etsimään luovia ratkaisuja.

Työ opetti myös hyvän dokumentaation arvon - rehellisesti M-Filesin COM-rajapinnan dokumentaatio on melko karu, eikä yleensä auta ohjelmoijaa tekemään hyviä ratkaisuja. Se on erittäin matalan tason kuvaus siitä, mitä rajapinnassa on, mutta ei siitä miten sitä pitäisi käyttää. Yleensä rajapinnan dokumentaatiota täytyi "täydentää" käyttämällä sivussa M-Filesiä ja tutustumalla graafisen käyttöliittymän kautta niihin ominaisuuksiin, joita kulloinkin rajapinnan kautta halusi käyttää. Käyttöliittymä on ohjelmoitu pitkälti saman COM-rajapinnan päälle, muutamaa poikkeusta lukuun ottamatta. Lähestymistapa toimi niin kauan, kun graafinen käyttöliittymä oli saatavilla - named value storagen käyttöön ei ole käyttöliittymää, eikä siten kunnon esimerkkiä. Tässä kohtaa pelkistetty

dokumentaatio ja esimerkkien puute aiheutti väärinkäsityksiä, ajan hukkaa ja turhautumista.

Vastoinkäymisistä huolimatta opinnäytteen tekeminen oli palkitseva kokemus. Applikaation toimintaan tutustuessa opin paljon uusia ohjelmointitapoja ja niksejä. Toisen ihmisen suunnittelemat ohjelmat ovat ihan erilaisia kuin omat, ja niihin perehtyminen tuo uusia näkökulmia omaan tekemiseen. Työssä oli vaihtelevasti haastavia ja helppoja vaiheita.

LÄHTEET

M-Files Oy. (n.d.a) *Esittelysivusto*. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/en/intelligent-information-management

M-Files Oy. (n.d.a) *Esittelysivusto*. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/en/intelligent-information-management

M-Files Oy. (n.d.b) *What is metadata-white paper*. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/Content/documents/en/res/Metadata_White_Paper.pdf

M-Files Oy. (n.d.b) *What is metadata-white paper*. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/Content/documents/en/res/Metadata_White_Paper.pdf

M-Files Oy. (n.d.c) *System overview*. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/user-guide/2018/eng/system_overview.html

M-Files Oy. (n.d.d) *Getting started*. Haettu 17.2.2020 osoitteesta
https://developer.m-files.com/Getting-Started/

M-Files Oy. (n.d.e) What is the difference between a user and a login account. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/user-guide/2018/eng/faq_the_difference_between_a_user_and_a_login_account.html

M-Files Oy. (n.d.f) *Difference between a nacl and a user group*. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/user-guide/2018/eng/faq_difference_between_a_nacl_and_a_user_group.html

M-Files Oy. (n.d.g) *Search functions*. Haettu 17.2.2020 osoitteesta
https://www.m-files.com/user-guide/2018/eng/Search_functions.html

M-Files Oy. (n.d.h) *New view*. Haettu 17.2.2020 osoitteesta https://www.m-files.com/user-guide/2018/eng/New_view.html

Liite 1: Aineistonhallintasuunnitelma

Julkaisutyökalun lähdekoodiin tutustutaan, ja sen pohjalta tehdään havaintoja sekä kartoitusta ohjelman toiminnasta. Havainnot kirjataan opinnäytetyön suunnitteluosuuteen. Suunnitelmien pohjalta tuotetaan uutta lähdekoodia. Tuotettu lähdekoodi raportoidaan sekä opinnäytetyössä, että git-versiohallintaohjelman commiteissa.
Lisäksi opinnäytetyötä varten käydään lyhyitä ohjaus- ja palautekeskusteluja toimeksiantajan ohjaajan kanssa, mutta niitä ei taltioida. Keskustelujen lopputulokset, siltä osin, kun ne vaikuttavat kehityksen suuntaan, raportoidaan opinnäytetyössä.

Ennen opinnäytetyön alkamista, sovimme toimeksiantajan ohjaajan kanssa, kuinka työkalun lähdekoodia käsiteltäisiin. Lähdekoodi laitetaan opinnäytteen tekemistä varten saataville toimeksiantajan hallitsemaan Azure DevOps-repositorioon. Opinnäytteen vaatimat muutokset lähdekoodiin tehdään git-työkalulla mainittuun repositorioon. Toimeksiantaja hallitsee näin ollen pääsyjä repositorioon, ja voi halutessaan rajata tai poistaa pääsyn lähdekoodiin kokonaan tai osittain. Ohjelmistokehityksessä luonnollisesti yksi kopio lähdekoodista löytyy myös kehittäjän tietokoneelta. Kovalevy, jolla lähdekoodia säilytetään, on kryptattu, ja näin ollen lähdekoodi on suhteellisen hyvässä turvassa esimerkiksi jos kehittämiseen käytetty laite anastetaan. Opinnäytetyön tultua valmiiksi, työkalun lähdekoodi poistetaan tekijän tietokoneelta. Azure DevOps repositorio on toimeksiantajan vastuulla opinnäytyön aikana ja sen jälkeen. Opinnäytetyössä tuotettu lähdekoodi on toimeksiantajan omaisuutta.

Opinnäytteessä annetut kuvaukset ohjelman toiminnasta ovat mahdollisimman suurpiirteisiä, ja kun on tarve kuvailla tarkkoja yksityiskohtia, niiden tukemiseksi esitetään vain niin pieni osa lähdekoodista, kuin on tarpeellista.