Groundhog-sovellusprojekti

Iiro Iivanainen
Harri Linna
Jere Pakkanen
Riikka Vilavaara

Sovellusraportti

Versio: 1.0.0
Julkinen
27.10.2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Hyväksyjä	**Päivämäärä**	**Allekirjoitus**	**Nimenselvennys**
Projektipäällikkö	__.__.2021		
Tilaaja	__.__.2021		
Ohjaaja	__.__.2021		

Tietoa dokumentista

Tekijät:

Iiro Iivanainen (II)	iiro.h.iivanainen@student.jyu.fi
Harri Linna (HL)	harri.s.linna@student.jyu.fi
Jere Pakkanen (JP)	jejopakk@student.jyu.fi
Riikka Vilavaara (RV)	riikka.k.vilavaara@student.jyu.fi

Dokumentin nimi: Groundhog-projekti, Sovellusraportti
Sivumäärä: 33
Tiedosto: sovellusraportti.tex

Tiivistelmä: Groundhog-projektissa toteutetulla ohjelmistolla voidaan yhdistää, hallita ja visualisoida erityyppisiä geologian näytteiden tutkimusdatoja. Sovellusraportissa kuvataan sovelluksen taustoja, käyttöliittymää, rakennetta ja toteutusratkaisuja. Lisäksi sovellusraportissa kuvataan myös projektissa noudatettuja ohjelmointi- ja testauskäytänteitä, testaustuloksia ja vaatimusten täyttämistä. Raportissa on myös kuvailtu ideoita jatkokehitystä varten.
Avainsanat: C#, jatkokehitys, käyttöliittymä, ohjelmointikäytänteet, tietorakenne.

Muutoshistoria

Versio	Päivämäärä	Muutokset	Tekijät
0.0.1	10.6.2021	Sovellusraportin pohja luotiin. Kirjoitettiin alustavasti sovelluksen ohjelmointikäytänteistä.	RV
0.0.2	11.6.2021	Kirjoitettiin johdanto ja tiivistelmä sekä jatkettiin ohjelmointikäytänteiden kirjoittamista.	RV
0.0.3	15.6.2021	Kirjoitettiin käyttöliittymäkuvausta.	RV
0.0.4	17.6.2021	Jatkettiin käyttöliittymäkuvauksen kirjoittamista.	RV
0.0.5	20.6.2021	Kirjoitettiin käyttöliittymäkuvaus ja ohjelmointikäytänteet loppuun. Kirjoitettiin taustoista ja tarpeista.	RV
0.0.6	21.6.2021	Kirjoitettiin sovelluksen rakenteesta ja toteutusratkaisuista.	RV
0.0.7	22.6.2021	Kirjoitettiin sovelluksen toimintalogiikasta.	RV
0.1.0	23.6.2021	Julkistettiin ensimmäinen väliversio.	RV
0.1.1	28.6.2021	Tehtiin korjauksia ohjaajalta saadun palautteen mukaisesti.	RV
0.1.2	1.7.2021	Kirjoitettiin järjestelmätestauksen testauskertojen tuloksista.	RV
0.1.3	3.7.2021	Kirjoitettiin toteutusratkaisujen kehittymisestä.	HL
0.1.4	5.7.2021	Kirjoitettiin yksikkötestauksesta.	RV
0.1.5	6.7.2021	Asemoitiin kansilehti uudelleen.	HL
0.1.6	7.7.2021	Siirrettiin tietoa projektista dokumenttiluokkaan ja päivitettiin termilistaa.	HL
0.1.7	13.7.2021	Kirjoitettiin vaatimusten toteutumisesta.	RV
0.1.8	16.8.2021	Kirjoitettiin sovelluksen epätyydyttävistä toteutusratkaisuista.	RV
0.1.9	17.8.2021	Kirjoitettiin jatkokehitysideoista.	RV
0.1.10	21.8.2021	Kirjoitettiin lähdekoodin jatkokehitysideoista.	RV
0.1.11	23.8.2021	Kirjoitettiin yhteenvetolukua.	RV
0.2.0	26.8.2021	Julkistettiin väliversio palautetta varten.	RV
0.2.1	23.10.2021	Kirjoitettiin toteutusratkaisujen kehittymisestä.	HL
0.2.2	24.10.2021	Kirjoitettiin ideoita jatkokehitykseen.	HL
0.2.3	24.10.2021	Kirjoitettiin toteutusratkaisuista ja jatkokehitysideoista.	JP
0.3.0	25.10.2021	Kirjoitettiin yhteenvetoa. Tehtiin palautteessa annetut korjaukset.	JP
0.4.0	26.10.2021	Tehtiin palautteessa annetut korjaukset.	HL ja JP
1.0.0	27.10.2021	Hyväksytty versio julkistettiin.	JP

Tietoa projektista

Groundhog-projekti suunnitteli ja toteutti Geologian tutkimuskeskukselle (GTK) multimodaalisen tutkimusdatan hallintaohjelmiston kevätlukukaudella 2021.

Tekijät:

Iiro Iivanainen (II)	iiro.h.iivanainen@student.jyu.fi
Harri Linna (HL)	harri.s.linna@student.jyu.fi
Jere Pakkanen (JP)	jejopakk@student.jyu.fi
Riikka Vilavaara (RV)	riikka.k.vilavaara@student.jyu.fi

Tilaajan edustajat:

Jukka Kuva	jukka.kuva@gtk.fi
Arttu Miettinen	arttu.i.miettinen@jyu.fi

Ohjaajat:

Jukka-Pekka Santanen	santanen@mit.jyu.fi
Juuso Tuononen	juuso.j.tuononen@student.jyu.fi

Yhteystiedot:

Sähköpostilistat	groundhog@korppi.jyu.fi, groundhog_opetus@korppi.jyu.fi
Sähköpostiarkistot	korppi.jyu.fi/kotka/servlet/list-archive/groundhog/, korppi.jyu.fi/kotka/servlet/list-archive/groundhog_opetus/
WWW-sivut	sovellusprojektit.it.jyu.fi/groundhog/, mit.jyu.fi/palvelut/sovellusprojektit/groundhog/

Sisällys

1 Johdanto **1**

2 Termit **2**

3 Taustaa ja tavoitteita **4**

4 Käyttöliittymäkuvaus **5**
4.1 Pääikkunan rakenne 5
4.2 Map-ikkuna 7
4.3 Controller-ikkuna 8
4.4 Attached Data -ikkuna 9
4.5 Connectors-ikkuna 10
4.6 Komentovalikoiden komennot 10
4.7 Liitoskohdan lisääminen 11
4.8 Edit Connectors -tila 12
4.9 Karttakuvan lisääminen tai vaihtaminen 13
4.10 Metatietojen tarkastelu 14
4.11 Konfiguraatiodialogi 15

5 Sovelluksen rakenne ja toteutusratkaisut **17**
5.1 Ohjelmiston yleisrakenne 17
5.2 Taustalla toimiva tietorakenne 18
5.3 Ohjelmiston toimintalogiikka 19
5.4 Kuvien lukemisen toteutusratkaisut 21

6 Ohjelmointikäytänteet **22**
6.1 Muotoilu-, nimeämis- ja kommentointikäytänteet 22
6.2 Lähdekoodin ryhmittelykäytänteet 22
6.3 Kehitysympäristö 23

7 Testauskäytänteet ja tulokset **24**
7.1 Yksikkötestaus 24
7.2 Järjestelmätestauskertojen tulokset 24

8 Tavoitteiden toteutuminen **26**
8.1 Vaatimusten täyttyminen 26
8.2 Sovelluksen epätyydyttävät ratkaisut 26
8.3 Toteutusratkaisujen valinta ja kehittyminen 27

9 Ideoita jatkokehitykseen **28**
9.1 Jatkokehitysympäristön pystytysohjeet 28
9.2 Kaksiulotteisen karttakuvakomponentin jatkokehitysideat 28
9.3 Kolmiulotteinen karttakuvakomponentti 29
9.4 Muut sovelluksen toiminnallisuuteen liittyvät jatkokehitysideat 29
9.5 Sovelluksen lähdekoodin jatkokehitysideat 30

10 Yhteenveto **31**

Lähteet **32**

1 Johdanto

Groundhog-projekti[1] suunnitteli ja kehitti Geologian tutkimuskeskukselle multimodaalisen tutkimusdatan hallintaohjelmiston kevätlukukaudella 2021. Ohjelmiston avulla käyttäjä pystyy tarkastelemaan kolmiulotteisia harmaasävykuvia (kuten tomografiakuvia) ja liittämään tiettyihin kohtiin kuvassa muita tiedostoja. Nämä tiedostot voivat sisältää esimerkiksi mittaustuloksia tai huomioita. Projektin jäsenet kehittivät ohjelmiston osana Jyväskylän yliopiston informaatioteknologian tiedekunnan *Sovellusprojekti*-nimistä projektiopintojaksoa.

Sovellusraportissa kuvataan kehitetyn sovelluksen käyttöliittymää, toimintaa, rakennetta ja ohjelmointikäytänteitä, tavoitteita ja niiden toteutumista sekä jatkokehitysideoita. Lisäksi raportissa kuvataan lyhyesti sovelluksen taustaa, testauskäytänteitä ja -tuloksia. Projektiraportissa [4] kuvataan projektin läpivientiä ja vaatimusmäärittelyssä [9] kuvataan sovelluksen vaatimuksia. Järjestelmätestaussuunnitelma [8] sisältää vaatimuspohjaisen toiminnallisen testauksen testiskenaariot sekä testauskerran suorituksen ja raportoinnin ohjeet. Järjestelmätestauskertojen raportit [6] ja [7] sisältävät testauskertojen yksityiskohtaiset tulokset. Lisäksi sovelluksesta on kirjoitettu erilliset lyhyet käyttöohjeet [10]. Sovellusraportin laatimisessa on hyödynnetty sovellusprojektien ohjeen [11] ohella Peltihamsteri- ja Kodaviprojektien sovellusraportteja [3] ja [1].

Sovellusraportti on jaettu 10 lukuun. Raportin luvussa 2 kuvataan dokumentissa esiintyviä termejä. Projektin tavoitteista kerrotaan luvussa 3. Luvussa 4 esitellään ohjelman käyttöliittymää. Sovelluksen toimintaa ja rakennetta kuvataan luvussa 5. Ohjelmointikäytänteitä kuvataan luvussa 6. Luvussa 7 esitellään testauskäytänteitä ja -tuloksia. Tavoitteiden toteutumista ja sovelluksen puutteita kuvataan luvussa 8. Jatkokehitysideoita esitellään luvussa 9.

[1] Groundhog Daytä vietetään Kanadassa vuosittain 2. helmikuuta.

2 Termit

Dokumentissa esiintyvät aihealueen termit ovat seuraavat:

Dataliitos	on kartan ja vähintään yhden liitedatan välinen liitos, johon liittyy liitoskohta ja metadataa.
Kartta	on kolmiulotteinen harmaasävykuva, jota voidaan kuvata esim. kuvasiivujen avulla.
Karttatiedosto	on tiedosto, joka luetaan sovellukseen kartaksi.
Kuvakulma	on koordinaatiston avulla ilmoitettava suunta, josta näytekuvaa tarkastellaan.
Kuvapino	on näytekuvasta muodostettu pino kaksiulotteisia kuvia.
Leikkauskuva	on näytekuvan halki kulkevaa leikkausta esittävä kaksiulotteinen kuva.
Liitosdata	sisältää näytteeseen liittyvät mittaustulokset sekä muut tiedostot ja tiedot.
Liitoskohta	on kartalla sijainti, johon yksittäinen dataliitos liittyy. Se voi olla piste, taso, suora tai muu geometrinen muoto.
Malli	on karttatiedoston esitystapa, jossa karttatiedoston kolmiulotteinen data projisoidaan kaksiulotteiseksi kuvaksi.
Metatieto	on näytteeseen tai liitedataan liittyvää kuvailevaa tietoa, kuten tiedoston laatijan nimi, maantieteellinen sijainti tai päivämäärä.
Näyte	on kokonaisuus, joka sisältää kaiken sovelluksessa auki olevan tiedon liittyen yhteen geologiseen näytekappaleeseen.
Näytekuva	on geologista näytekappaletta esittävä kuva, joka voi olla kolmiulotteinen.
Pääakseli	on näytekuvan sivua vastaan kohtisuorassa oleva X-, Y- tai Z-koordinaattiakseli.
Siivu	on yksittäinen kaksiulotteinen kuva kuvapinosta.
Syvyys	kertoo, monesko kuva tietystä kuvapinosta on kyseessä.
Tilakartta	on malli, joka esittää kartan avaruuden dimensioita.

Dokumentissa esiintyvät kehitysvälineisiin, kehitystekniikoihin ja projektin hallintaan liittyvät termit ovat seuraavat:

AvalonDock	on käyttöliittymäkirjasto, joka mahdollistaa WPF-sovellusten ikkunoinnin, kuten moderneissa IDE:issä.
C#	on Microsoftin kehittämä ohjelmointikieli .NET-alustalle.
ExcelDataReader	on Microsoft Excel -tiedostojen käsittelyyn kehitetty C#-kirjasto.
Git	on hajautettu versiohallintatyökalu, jonka avulla voidaan hallita lähdekoodia.
ImageMagick	on tehokas kuvankäsittelyohjelma, joka tukee yli 100 kuvaformaattia.
Magick.NET	on ImageMagick-ohjelmaa tukeva kuvankäsittelyn mahdollistava C#-kirjasto.
SharpGL	on OpenGL-kirjasto, jolla voidaan piirtää tietokonegrafiikkaa WPF-sovelluksen komponenttiin.
WPF	on Windows-käyttöjärjestelmän graafisten käyttöliittymien kehittämiseen tarkoitettu sovelluskehys .NET-alustalle.
WpfExToolkit	on käyttöliittymäkirjasto, joka sisältää valmiita WPF-sovelluksissa käytettäviä komponentteja.

3 Taustaa ja tavoitteita

Geologian tutkimuskeskuksella (GTK) on menetelmäkehityksen ja uudehkon tomografialaboratorion myötä noussut tarve ohjelmistolle, jolla voidaan helposti hallita usean mittalaitteen tutkimusdataa yhtäaikaisesti näiden keskinäiset suhteet hahmottaen. Ohjelmiston kehitystyön toteutti Jyväskylän yliopiston informaatioteknologian tiedekunnan opiskelijaprojekti opintojaksona *TIES405 Sovellusprojekti*.

Röntgentomografia on kolmiulotteista matriisimuotoista harmaasävykuvaa, jonka kuvakoot tyypillisesti vaihtelevat sadoista megatavuista kymmeniin gigatavuihin. PerGeos-ohjelman vaatiman laskentatehon ja lisenssien rajallisen määrän seurauksena nykyiset prosessit asettavat rajoitteita perustietokoneiden käyttämiselle tutkimustyössä. Käyttäjien tarpeista johdettuja kehitettävän sovelluksen tavoitteita kuvataan tarkemmin vaatimusmäärittelyssä [9].

Kehitettävän sovelluksen osalta tavoitteena ja ohjelman uutuusarvona oli liittää näytteeseen kolmiulotteisen tomografiakuvan ohella muun tyyppistä tutkimusdataa sekä kyetä visualisoimaan näiden välisiä yhteyksiä samanaikaisesti. Liitettävä tutkimusdata voi olla esimerkiksi tekstimuistiinpano, mikroskooppikuva tai diffraktiospektri. Tutkimusdataa olisi tarve lisätä pisteeseen, tasoon tai viivaan. Pääasiassa ohjelmaa kuitenkin hyödynnetään piste- ja tasoanalyysissä, joten nämä ovat myös olennaisimmat liitoksen tyypit.

Tomografiakuvan näyttämistä varten sovelluksella tulisi pystyä mm. muuttamaan harmaasävyskaalaa, kirkautta ja kuvakulmaa manuaalisesti. Numeerisia arvoja (kuten kirkkaus tai näytettävän tason kulma) tulisi pystyä muokkaamaan liukusäätimen lisäksi täsmällisillä numeerisilla arvoilla. Alkuperäistä kuvaa ei saa muokata. Tomografiakuvaa tulisi pystyä katsomaan vähintään pääakselikoordinaattien vastaisesti. Vapaan kuvakulman valinta oli tärkeä, mutta ei kriittinen toiminnallisuus.

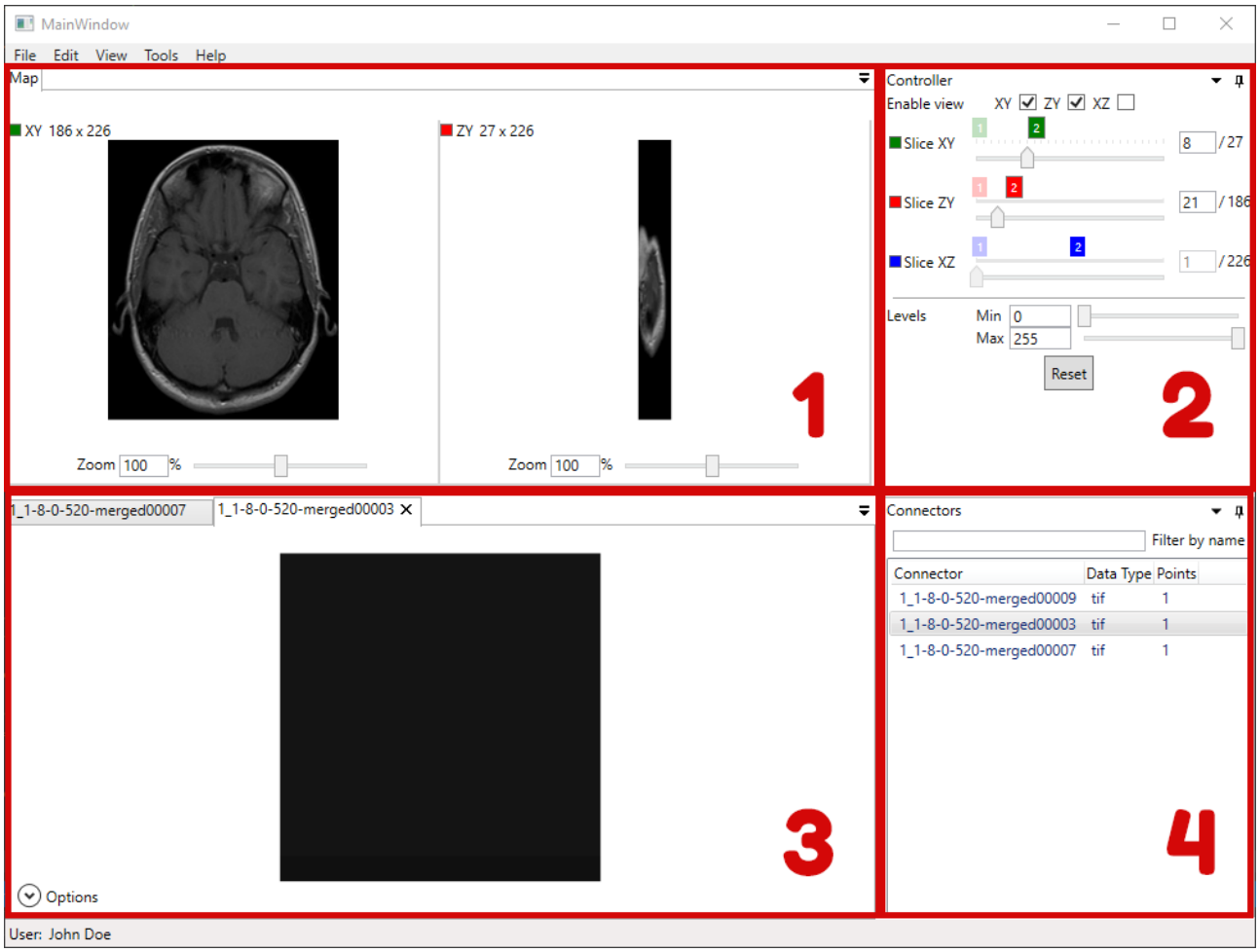
4 Käyttöliittymäkuvaus

Sovelluksen käyttöliittymää ja toimintoja kuvataan erillisissä käyttöohjeissa [10], joka tulee sovelluksen mukana. Käyttöohjeet ovat tiiviit, joten käyttöliittymää esitellään luvussa yksityiskohtaisemmin.

4.1 Pääikkunan rakenne

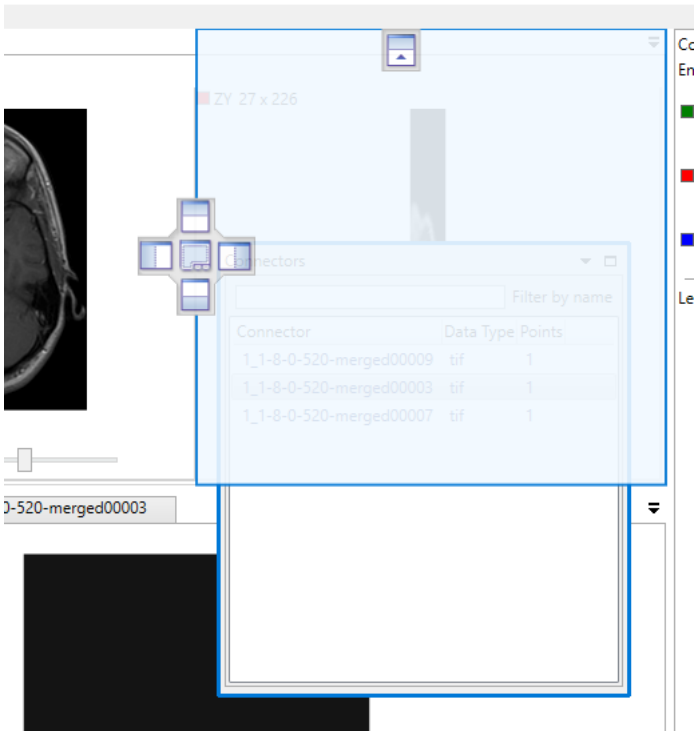
Sovelluksen pääikkunan päänäkymä on jaettu neljään osaan kuvan 4.1 osoittamalla tavalla. Valikko löytyy pääikkunan yläosasta ja ikkunan alaosassa on tilapalkki. Käyttäjä voi muuttaa päänäkymän ikkunoiden kokoa pääikkunan sisällä haluamallaan tavalla raahaamalla niiden reunaa hiirellä. Ikkunoiden paikkaa voi pääikkunan sisällä vaihtaa raahaamalla hiirellä ikkunan otsikkoriviltä. Ikkunan voi myös halutessaan jättää muiden pääikkunan elementtien päälle ikään kuin erilliseksi ikkunaksi, jonka kokoa ja sijaintia voi vapaasti muokata.

Pääikkunan alaosassa olevassa tilapalkissa näkyy tieto senhetkisestä käyttäjän nimestä ja mahdollisista käynnissä olevista kuvapinojen luomisoperaatioista. Kuvassa 4.1 jaoteltuja pääikkunan päänäkymän ikkunoita ovat *Map*, *Controller*, *Attached Data* ja *Connectors*. Ikkunoita kuvataan tarkemmin luvuissa 4.2, 4.3, 4.4 ja 4.5.



Kuva 4.1: Pääikkunan näkymät 1. *Map*, 2. *Controller*, 3. *Attached Data* ja 4. *Connectors*.

Kuvassa 4.2 esitetään *Connectors*-ikkunan siirtämistä *Map*-ikkunan viereen. Kun ikkunaa raahataan toisen osion päälle, käyttöliittymässä näytetään viisikohtainen elementti, joista käyttäjä voi valita, halutaanko raahattavan ikkunan menevän alla olevan osion ylä-, ala-, vasemmalle vai oikealle puolelle tai alla olevan ikkunan kanssa samaan kohtaan omaksi välilehdekseen.



Kuva 4.2: *Connectors*-ikkunan siirtäminen *Map*-ikkunan viereen.

4.2 Map-ikkuna

Karttanäkymä *Map* (kuvassa 4.1 merkitty numerolla 1) sisältää näytteeseen lisätyn karttakuvan siivuja kolmesta eri kuvakulmasta. Kerrallaan näytettävät kuvakulmat voi käyttäjä valita yksitellen *Controller*-ikkunasta (katso luku 4.3 ja kuva 4.1). Karttanäkymä on jaettu osiin sen mukaan, kuinka monta kuvakulmaa kulloinkin on valittuna. Tällöin esimerkiksi yhden valitun kuvakulman tapauksessa kuvasiivun sisältävä elementti vie koko karttanäkymälle varatun tilan.

Oletuksena karttanäkymässä näkyvät kuvasiivujen lisäksi myös liitoskohdat, mikäli niiden sijainti on tarkasteltavaksi valitulla kuvasiivulla. Kun liitoskohtaa klikataan hiirellä, siihen liittyvä tiedosto aukeaa *Attached Data* -näkymään omana välilehtenään (katso luku 4.4). Liitoskohdat saa pois näkyviltä poistamalla valinnan valikon *View* asetuksesta *Attached Data Points*.

Map-karttanäkymässä voi jokaista kuvasiivua tarkentaa tai loitontaa erikseen kuvasiivun alapuolella olevalla liukusäätimellä tai kirjoittamalla haluttu skaala prosentteina manuaalisesti tekstikenttään. Kuvasiivuihin saa myös valikosta näkyville värikoodatut koordinaattiakselit, joiden avulla käyttäjä näkee, mihin kohtaan muiden kuvakulmien tarkasteltavaksi

valitut kuvasiivut sijoittuvat. Koordinaattiakselit saa näkyville valitsemalla valikosta *View* asetus *Axis Color Coding*.

4.3 Controller-ikkuna

Kuvasiivujen hallintanäkymän *Controller* (kuvassa 4.1 merkitty numerolla 2) avulla voidaan hallita karttanäkymässä (katso luku 4.2) näkyviä kuvasiivuja ja kuvien harmaasävyaluetta. Ylimpänä olevilla valintaruuduilla voidaan valita, mitkä kolmesta kuvakulmasta näytetään kerrallaan. Mikäli karttakuvaa ei ole, tai muiden kuvakulmien kuvapinoja ei ole luotuna tai tallennettuina, puuttuvien kuvapinojen osalta valintaruutuja ei voi valita.

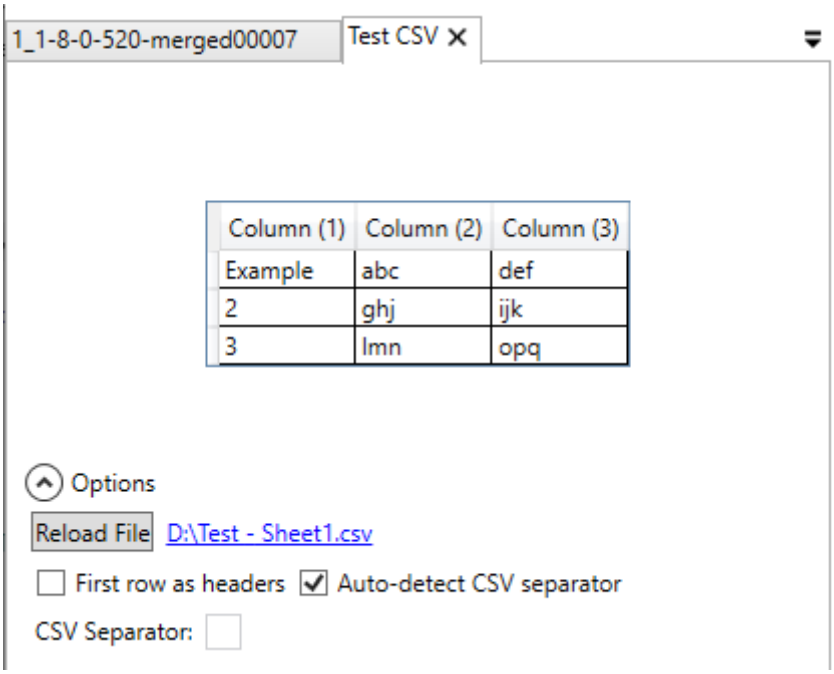
Valintaruutujen alapuolella on kolme värikoodattua liukusäädintä, joiden avulla voidaan valita, mitä kuvasiivua halutaan kustakin kuvakulmasta tarkastella. Liukusäätimet eivät ole käytössä, jos niihin liittyvää tasoa ei ole valittu karttanäkymään näkyville. Liukusäädinten yläpuolella olevat numerot kuvaavat näytteeseen liitettyjen liitoskohtien määrää ja sijaintia kyseisellä akselilla. Eniten liitospisteitä sisältävät siivut ovat väritykseltään tummempia. Numeroita klikkaamalla käyttäjä voi tarkastella kyseistä siivua karttanäkymässä, mikäli kyseinen akseli on valittu karttanäkymään näkyville. Liukusäädinten vieressä oleviin tekstilaatikoihin voi kirjoittaa manuaalisesti halutun siivun numeron ja tekstilaatikon vieressä näkyy kyseisen akselin siivujen määrä.

Alimpana kuvasiivujen hallintanäkymässä on harmaasävyalueen määrittämiseen liittyvät toiminnot. Harmaasävyaluetta voi muuttaa manuaalisesti numeerisen arvon kirjoittamalla tai liukusäädintä raahaamalla. *Reset*-painikkeesta harmaasävyalue palautuu takaisin oletusarvoihin. 8- ja 16-bittisten karttakuvien tapauksessa numeroarvot ovat ei-negatiivisia kokonaislukuarvoja ja 32-bittisten karttakuvien tapauksessa arvot ovat floating point -tyyppisiä liukulukuja.

Kuvasiivujen hallintanäkymän voi halutessaan piilottaa ohjelman sivuun painamalla oikeassa ylänurkassa olevaa nastan kuvaa, jolloin sen saa näkyville viemällä hiiren ikkunan oikeaan reunaan ilmestyneen *Controller*-sanan päälle. Näkymän saa takaisin ns. kiinteäksi elementiksi painamalla nastan kuvaa uudestaan.

4.4 Attached Data -ikkuna

Liitosdatanäkymässä *Attached Data* (kuvassa 4.1 merkitty numerolla 3) näytetään käyttäjän avaama liitostiedosto (katso luku 4.5). Kuvassa 4.3 on esimerkki näkymästä. Kun ohjelmassa avataan useampi liitostiedosto samanaikaisesti nähtäville, liitostiedostonäkymät avautuvat oletuksena omiksi välilehdikseen. Halutessaan käyttäjä voi niitä siirtää luvussa 4.1 kuvatulla tavalla. Kun kaikki liitostiedostot suljetaan, liitosdatanäkymä poistuu näkyvistä ja muut ikkunat saavat käyttöönsä enemmän tilaa.



Kuva 4.3: *Attached Data* -ikkunassa ovat avattuina CSV-tiedosto ja sen asetukset.

Liitosdatanäkymässä pystytään näyttämään kaksiulotteisia kuvia (yleisimmät tiedostotyypit), CSV-tiedostoja (`csv`), Microsoft Excel -tiedostoja (`xlsx`, `xlsb` ja `xls`) ja tekstitiedostoja. Kaikki em. poikkeavat tiedostotyypit avataan tekstitiedostoina. Tiedostoja ei voi muokata sovelluksessa, mutta käyttäjä voi halutessaan muokata tiedostoja sovelluksen ulkopuolella.

Liitosdatanäkymässä vasemmalla alhaalla olevasta *Options*-kohdasta käyttäjä voi määrittää tiedostotyyppikohtaisia asetuksia avatulle liitostiedostolle. Kuvassa 4.3 näytetään CSV-tiedosto ja siihen liittyvät asetukset, kuten erottimen syöttäminen tai ensimmäisen rivin näyttäminen sarakkeiden otsikkoina. Kaikissa liitostiedostotyypeissä on myös liitostiedoston tiedostopolku ja *Reload File* -painike, josta tiedoston voi ladata sovellukseen uudestaan. Mikäli käyttäjä muokkaa tiedostoa ohjelman aikana jollain muulla ohjelmalla, muutokset saa sovellukseen näkyville kyseisestä painikkeesta.

4.5 Connectors-ikkuna

Liitoskohtanäkymä *Connectors* (kuvassa 4.1 merkitty numerolla 4) listaa näytteeseen lisätyt liitostiedostot. Listaa voi suodattaa lisätyn liitostiedoston nimen perusteella tai järjestää eri sarakkeiden mukaiseen tai käänteiseen aakkosjärjestykseen klikkaamalla sarakkeen otsikkoa.

Listan riviä tuplaklikkaamalla valitun rivin mukainen liitostiedosto avataan liitosdatan näkymään (katso luku 4.4). Klikkaamalla hiiren oikealla painikkeella listan riviä aukeaa valikko, josta voi suorittaa kyseiselle liitosdatalle seuraavat toimenpiteet:

- *Go To* siirtää karttanäkymän (katso luku 4.2) niihin kuvasiivuihin, joilla liitoskohta sijaitsee.
- *Display* avaa liitostiedoston.
- *Open File* avaa liitostiedoston oletusohjelmalla, joka on määritelty tiedostotyypille käyttöjärjestelmässä.
- *Open Folder* avaa hakemiston, jossa liitostiedosto sijaitsee.
- *Delete Selected* poistaa kyseisen liitoskohdan näytteestä.
- *Metadata* näyttää liitoskohdan metatiedot (katso luku 4.10).

4.6 Komentovalikoiden komennot

Sovelluksen komentovalikot ja niiden komennot ovat seuraavat:

File
- → *New* luo uuden näytteen.
- → *Open* avaa aiemmin tallennetun näytteen.
- → *Save* tallentaa nykyisen näytteen aiemmin määritettyyn tiedostoon.
- → *Save As...* tallentaa nykyisen näytteen uuteen tiedostoon.

Edit
- → *Add/Change Map* lisää karttakuvan näytteeseen, jos karttakuvaa ei vielä ole lisättynä. Jos karttakuva on olemassa, komento vaihtaa näytteen karttakuvan uuteen (katso luku 4.9).
- → *Add Connector* lisää uuden liitoskohdan näytteeseen (katso luku 4.7).
- → *Configure* avaa konfiguraatioikkunan, jossa käyttäjä voi muuttaa sovelluksen asetuksia (katso luku 4.11).

View
- → *Axis Color Coding* näyttää karttanäkymässä värikoodatut koordinaattiakselit.
- → *Attached Data Points* näyttää karttanäkymän kuvasiivuilla liitoskohdat.

→ *Sample Metadata* näyttää nykyisen näytteen yleiset metatiedot. Liitoskohtien omat metatiedot löytyvät *Connectors*-listan avulla.

Tools

→ *Edit Connectors* asettaa sovelluksen *Edit Connectors* -tilaan (katso luku 4.8).

→ *Create ZY stack* luo karttakuvasta X-koordinaattiakselin suuntaiset ZY-tasot kuvina.

→ *Create XZ stack* luo karttakuvasta Y-koordinaattiakselin suuntaiset XZ-tasot kuvina.

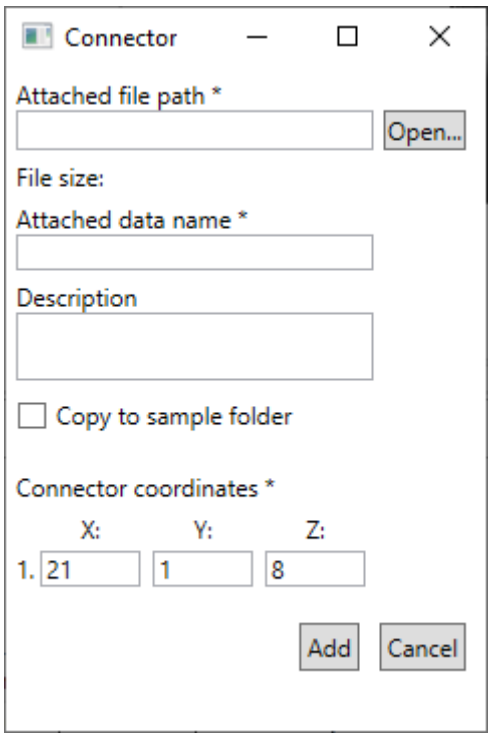
Help

→ *View Help* avaa sovelluksen käyttöohjeet.

→ *About* näyttää tietoja sovelluksesta.

4.7 Liitoskohdan lisääminen

Näytteen liitoskohtia voi lisätä joko valitsemalla valikosta *Edit* komento *Add Connector* tai klikkaamalla hiiren vasemmalla painikkeella karttakuvaa halutusta kohdasta, jos sovellus on liitoskohtien lisäämisen tilassa (katso luku 4.8). Sovellus avaa liitoskohtaa lisätessä kuvan 4.4 mukaisen dialogi-ikkunan.



Kuva 4.4: Liitoskohdan lisäysikkuna.

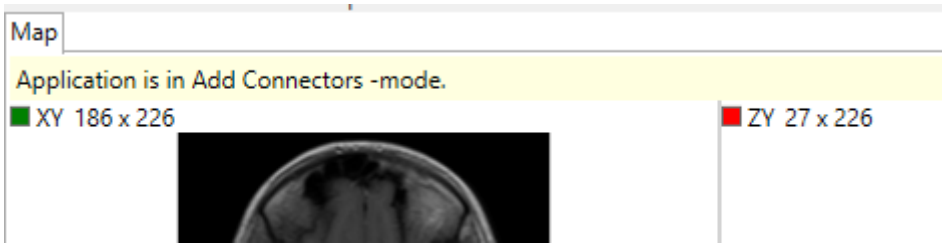
Tähdellä merkityt kentät ovat pakollisia. *Description*-kenttään voi halutessaan syöttää kuvauksen lisättävästä liitostiedostosta, ja se lisätään automaattisesti liitoskohdan yhdeksi metatiedoksi. Lisäksi metatietoihin tallentuu liitoskohdan lisäyksen päivämäärä ja kellonaika sekä käyttäjän nimi. Käyttäjän nimeä voi muokata erillisestä konfiguraatiodialogista (katso luku 4.11).

Liitoskohdan koordinaatit (*Connector coordinates*) tulevat dialogiin automaattisesti, mutta käyttäjä voi myös halutessaan muokata niitä manuaalisesti. Jos liitoskohta on lisätty valikon kautta, koordinaateiksi tulee sillä hetkellä tarkasteltavana olevien siivujen numerot. Niiden akselien tapauksessa, joiden kuvapinoa ei ole generoitu, kyseisen akselin arvoksi tulee 1. Jos liitoskohta taas on lisätty hiiren klikkauksella karttanäkymästä, tulee dialogiin koordinaateiksi hiiren klikkauspiste klikatun siivun perusteella.

Liitoskohdan koordinaatit voi merkitä näytteen karttakuvan ulkopuolelle, mutta tällöin se ei näy karttakuvassa tai kuvasiivujen hallintanäkymässä *Controller* olevassa liukusäätimessä. Liitoskohta kuitenkin näkyy edelleen *Connectors*-ikkunan (katso kuva 4.1) listauksessa.

4.8 Edit Connectors -tila

Kun *Edit Connectors* -tila on päällä (valikosta *Tools* komento *Edit Connectors*), se näkyy käyttäjälle keltataustaisena tilapalkkina karttanäkymän yläosassa karttakuvasta muodostettujen kuvasiivujen yläpuolella kuten kuva 4.5 esittää. Kun tila on päällä, käyttäjä voi lisätä *Map*-ikkunassa (katso luku 4.2) liitoskohtia kartalle hiiren klikkauksella. Klikkauksesta avautuu liitoskohdan lisäämisen dialogi (katso luku 4.7), jonne klikkauksen koordinaatit menevät automaattisesti liitoskohdan sijainniksi.

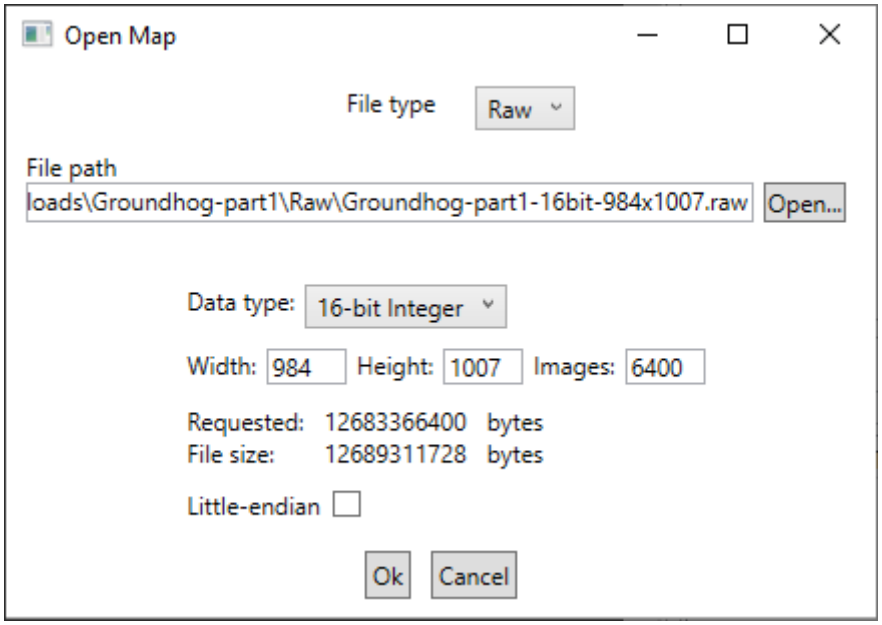


Kuva 4.5: Liitoskohtien lisäämisen mahdollistava tila näkyy keltaisella tilapalkilla.

Edit Connectors -tilan ollessa päällä kartalle merkityistä liitoskohdista (valikosta *View* asetus *Attached Data Points* on päällä) hiirellä klikatessa liitoskohtaan liitetty liitostiedosto ei avaudu näkyville. Jatkokehityksessä *Edit Connectors* -tilaan on mahdollista lisätä muitakin ominaisuuksia, kuten liitoskohdan siirtäminen (katso luku 9).

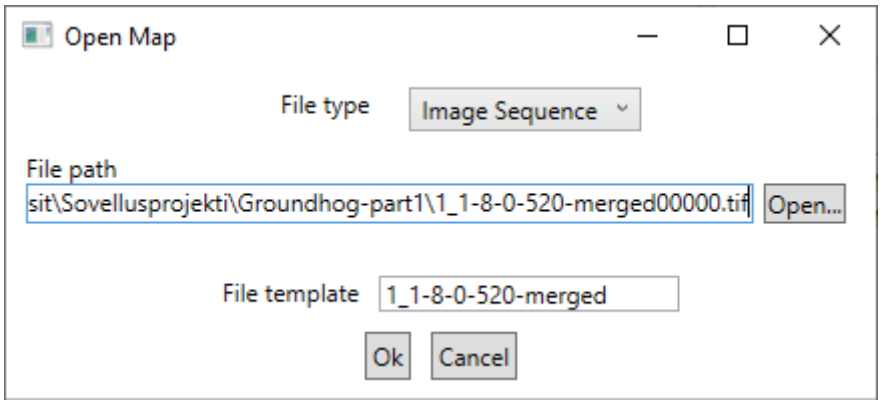
4.9 Karttakuvan lisääminen tai vaihtaminen

Karttakuvan lisäämisdialogin avulla käyttäjä voi lisätä tai vaihtaa näytteeseen liitettyä karttakuvatiedostoa. Dialogissa valitaan haluttu karttakuvatyyppi (vaihtoehtoina Image Sequence eli kuvasarja, Multiframe TIFF tai Raw) ja valinnan perusteella dialogin asetukset karttakuvalle muuttuvat. Karttakuvan lisäämisdialogi esitetään kuvassa 4.6.



Kuva 4.6: Karttakuvan lisäämisdialogi Raw-tyyppinen karttakuva valittuna.

Kuvasarjaa valitessa käyttäjä voi valita minkä tahansa kuvasarjan kuvista. Tuettuja kuvasarjan tiedostotyyppejä ovat `TIFF` (`tif` ja `tiff`) ja `PNG` (`png`). Kun tiedosto on valittu, tiedostopolun sisältävän tekstikentän alapuolella olevaan *Template*-kenttään tulee alkuarvaus kuvasarjan nimen kaavalle (kuva 4.7), eli jokaisen kuvasarjan kuvan tiedostonnimessä esiintyvälle merkkijonolle. Käyttäjä voi halutessaan muuttaa kaavaa, jos esimerkiksi samassa kansiossa on useampi kuvasarja ja alkuarvaus on väärä.



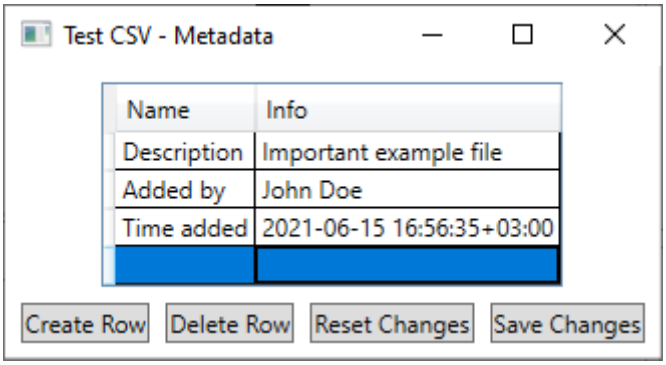
Kuva 4.7: Karttakuvan lisäämisdialogi Image Sequence -tiedostoille.

Multiframe TIFF eli kolmiulotteiset TIFF-tiedostot vaativat asetuksista vain tiedostopolun. Sovelluksessa tuettavien multiframe TIFFien kokorajoitus on 4 GB.

Raw-tiedostojen avaamisessa olevat asetukset näkyvät kuvassa 4.6. Tiedostopolun lisäksi käyttäjän tulee antaa tiedot kuvan bittisyvyydestä sekä leveys, korkeus ja kuvapinon syvyys. Kuvapinon syvyyttä käytetään tällä hetkellä vain tiedoston odotetun koon laskemiseen, jotta käyttäjä voi kuvaa lisätessä tarkistaa, syöttikö hän kuvan dimensiot oikein. Laskennallinen tiedoston koko näkyy dimensio-tekstilaatikoiden alapuolella, jonka ohella näkyy myös tiedoston todellinen koko vertailua varten. Lisäksi käyttäjän tulee valita Raw-tiedoston tavujärjestystä varten valintaruudulla, onko avattava kuva little-endian vai big-endian.

4.10 Metatietojen tarkastelu

Metatietojen esittämisen ja muokkaamisen ikkunassa metatiedot näkyvät taulukkona. Yksittäinen metatieto vastaa yhtä taulukon riviä ja jokaisella metatiedolla on sekä metatiedon nimi että tarkempi kuvaus. Automaattisesti muodostettavassa tiedoston tai näytteen lisäysajassa näkyy päivämäärä, kellonaika ja käyttäjän UTC-aikavyöhyke. Esimerkiksi kuvassa 4.8 auki olevassa liitostiedoston metatietoikkunassa aikavyöhyke on `UTC +03:00`.



Kuva 4.8: Metatietojen hallintaikkuna avattuna `Test CSV` -nimiselle liitoskohdalle.

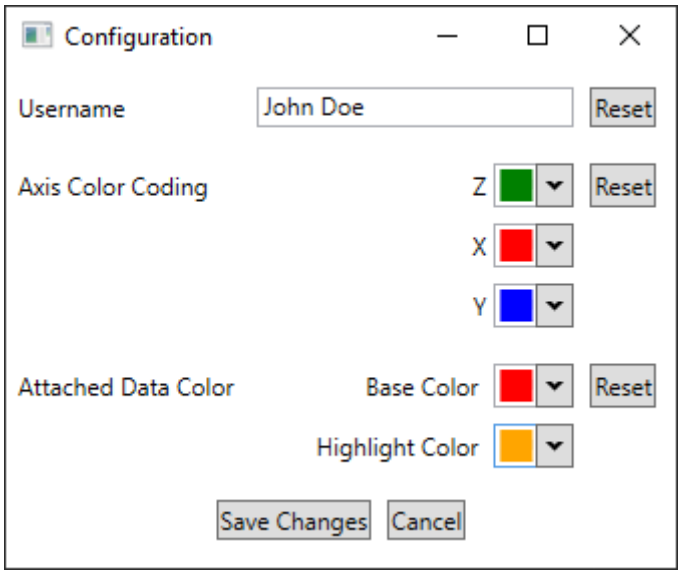
Metatietojen taulukon rivien tekstejä voi muokata taulukossa suoraan. Tämän lisäksi kuvassa 4.8 nähtävien painikkeilla voidaan suorittaa seuraavat toimenpiteet:

- *Create Row* lisää uuden rivin taulukkoon.
- *Delete Row* poistaa valitun metatiedon eli taulukon rivin.
- *Reset Changes* palauttaa metatiedot takaisin siihen tilaan, jossa ne olivat ennen ikkunan avaamista.
- *Save Changes* tallentaa metatietoihin tehdyt muutokset.
- Ikkunan sulkeminen ruksista kumoaa kaikki tallentamattomat muutokset.

Yksittäisen liitoskohdan metatietoja voi tarkastella *Connectors*-listan avulla (katso luku 4.5). Koko näytetiedoston yleiset metatiedot löytyvät valitsemalla valikosta *View* komento *Sample Metadata*.

4.11 Konfiguraatiodialogi

Kuvan 4.9 esittämässä konfiguraatioikkunassa käyttäjä voi muuttaa metatietoihin lisättävää käyttäjänimeä tai värien asetuksia. Muutettavia värejä ovat värikoodatut koordinaattiakselit sekä kartalle merkittävien liitoskohtien värit. Liitoskohtien värien tapauksessa voidaan muuttaa sekä liitoskohtien perusväri että korostusväri, joka näkyy vietäessä hiiri liitoskohdan päälle kartalla.



Kuva 4.9: Sovelluksen asetusten konfiguraatiodialogi.

Oletuksena käyttäjänimenä on se Windows-käyttäjätili, jolla sovellus avattiin. Asetukset eivät tällä hetkellä tallennu seuraavaa ohjelman käyttökertaa varten.

5 Sovelluksen rakenne ja toteutusratkaisut

Luvussa esitellään kehitetyn sovelluksen yleistä rakennetta ja toteutusratkaisuja.

5.1 Ohjelmiston yleisrakenne

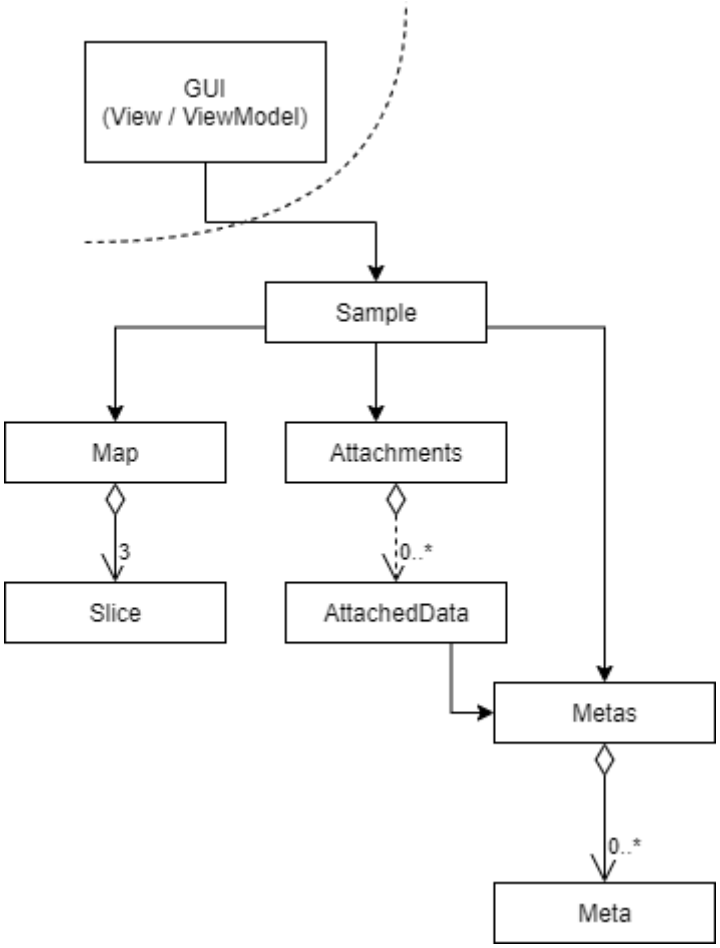
Sovelluksen yleisrakenne noudattaa kuvan 5.1 mukaista Model-View-ViewModel-arkkitehtuuria (MVVM-arkkitehtuuri). Näistä View ja ViewModel ovat käyttöliittymään liittyvää toiminnallisuutta ja Model on taustalla toimiva tietorakenne, jonka toiminnallisuutta käyttöliittymä näyttää. Arkkitehtuurin tarkoituksena oli erottaa käyttöliittymä ja taustalla oleva toiminnallisuus toisistaan omiksi kokonaisuuksikseen.

Kuva 5.1: Sovelluksessa käytetty Model-View-ViewModel-arkkitehtuuri.

MVVM-arkkitehtuurin osista View hoitaa pääasiassa käyttäjälle käyttöliittymässä näkyvien osioiden näyttämiseen liittyvän toiminnallisuuden. Tähän kuuluvat muun muassa itse käyttöliittymään kehitetyt omat `UserControl`-elementit ja dialogi-ikkunat.

ViewModelin tehtävänä on yhdistää taustalla oleva tietorakenne käyttöliittymän toiminnallisuuksiin. Tämä hoidetaan pääasiassa ns. bindauksen eli sitomisen avulla, missä Viewin elementtien arvoja on sidottu ViewModelin attribuutteihin. Itse käyttöliittymäelementtejä ViewModelin puolella ei ole. Toteutuneessa sovelluksessa ViewModel-luokkia on vain `MainWindowViewModel`, joka hoitaa pääikkunan ja Modelin välistä kommunikointia.

Model muodostaa ohjelman taustalla toimivan tietorakenteen ja sen toiminnallisuuden. Sen tehtäviin kuuluu muun muassa liitoskohtien tietojen hallinta ja karttakuvien sijainnin ja lukemisen hallinta. Model on toteutettu pääasiassa kerrosarkkitehtuurin mukaisesti, eli kutsuhierarkiat menevät ylhäältä alaspäin kuten kuvasta 5.2 voidaan havaita. Luvussa 5.2 kuvaillaan Modelin rakennetta tarkemmin.



Kuva 5.2: Sovelluksen toimintalogiikka toteutettuna kerrosarkkitehtuurin mukaisesti.

5.2 Taustalla toimiva tietorakenne

Taustalle toteutettu tietorakenne eli kuvan 5.1 Model toteutettiin kerrosarkkitehtuurin mukaisesti. Hierarkiassa ylimpänä olevalla luokalla on pääsy kaikkiin sitä hierarkiassa alempana oleviin kerroksiin, mutta kutsut eivät voi mennä hierarkiassa alhaalta ylöspäin. Kuvasta 5.2 nähdään Modelin sisältämät luokat ja niiden välinen hierarkia tarkemmin.

Ylimpänä hierarkiassa oleva `Sample`-luokka kuvaa reaalimaailman näytettä, joka käsittää sekä karttakuvan että siihen liittyvät mittaustulokset ja muut huomiot. `Sample`lla on tiedossaan sekä itse karttakuva (`Map-luokka`) että liitetyt tiedostot (`Attachments`-luokka). Lisäksi `Sample`lla on oma `Metas`-olionsa, johon on tallennettu näytteeseen liittyvät yleiset metatiedot.

`Map`-luokka pitää sisällään karttakuvaan liittyviä tietoja sekä kolme `Slice`-tyyppistä attribuuttia, joissa on tallessa sillä hetkellä näkyvillä olevat kuvapinot. `Slice`-luokka puoles-

taan hoitaa yksittäiseen kuvasiivuun liittyvät tehtävät ja tiedot, kuten harmaasävyskaalan muutokset tai kuvan muuttamisen tavutaulukoksi käyttöliittymään viemistä varten.

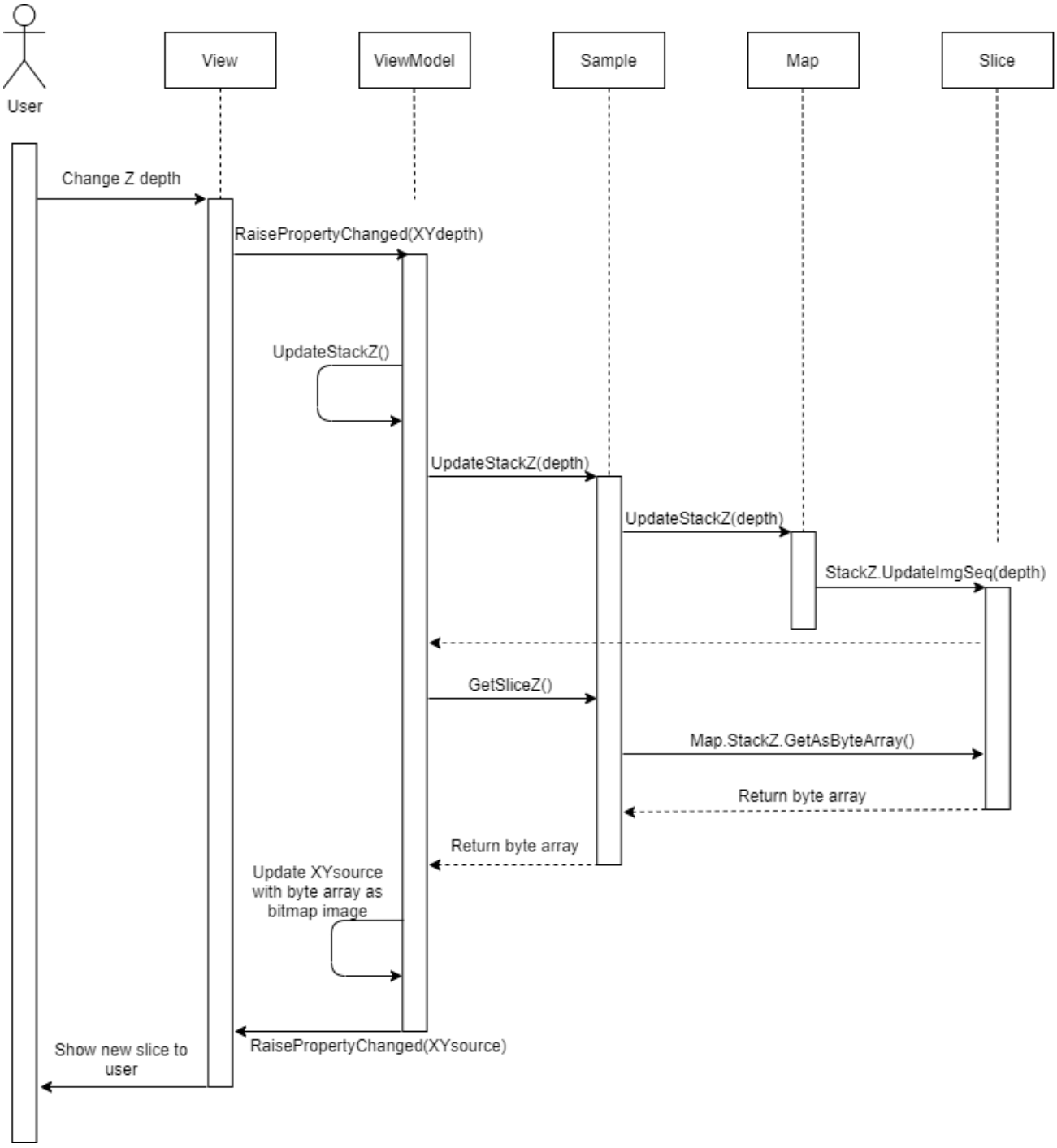
`Attachments`-luokka pitää sisällään näytteeseen liitetyt dataliitoskohdat, joita kuvataan `AttachedData`-olioilla. `Attachments`-luokan tehtävänä on koota kaikki liitoskohdat yhteen ja hoitaa niiden käsittelyyn liittyviä tehtäviä, kuten etsiminen. `AttachedData`-luokka puolestaan sisältää yhteen liitoskohtaan liittyvät tiedot, kuten liitoskohdan sijainti karttakuvalla ja liitetyn tiedoston polku. Kullakin `AttachedData`-oliolla on oma `Metas`-olionsa, joka sisältää kyseiseen yksittäiseen liitoskohtaan liittyvät metatiedot.

`Metas`-luokan tarkoituksena on koota joko näytteeseen tai liitoskohtaan liittyvät metatiedot yhteen. `Metas`-luokan alla on `Meta`-luokan olioita, jotka sisältävät metatiedon tunnisteen sekä kuvauksen. Esimerkiksi tiedoston käyttäjän nimen tapauksessa metatiedon tunnisteena olisi `Username` eli käyttäjänimi ja tietona käyttäjän nimi (kuten `Matti Meikäläinen`).

Tietorakenteen tallennus on toteutettu .NET mukana tulevalla JSON kirjastolla, jolla pystytään serialisoimaan ja deserialisoimaan olioita JSON-tiedostoksi. Serialisointi on toteutettu synkronoidusti, koska asyncronoidun tallentamisen/lataamisen yhteydessä ilmeni ongelmia tulosten synkronoinnin kanssa. Tämä ei ole tuottanut kumminkaan suuria ongelmia käytettävyyden suhteen, koska serialisoitu dataa on yleensä vähän.

5.3 Ohjelmiston toimintalogiikka

Kuvassa 5.3 on kuvattu sovelluksen toimintalogiikkaa yksinkertaistetun sekvenssikaavion avulla. Kaavio kuvaa tapausta, jossa käyttäjä vaihtaa käyttöliittymän kautta katseltavaa XY-tason kuvapinon kuvasiivua toiseen ja sitä, miten kutsuhierarkia toteutetussa ohjelmistossa toteutuu.



Kuva 5.3: Yksinkertaistettu sekvenssikaavio XY-kuvasiivun vaihtamisesta.

Kuvan 5.3 mukaisesti käyttöliittymästä lähtee pyyntö `Sample`-luokalle päivittää tallessa oleva kuvasiivu uuteen valitun syvyyden perusteella. Lopulta ViewModel pyytää `Sample`-luokalta uutta siivua tavutaulukkona, jonka jälkeen se muutetaan Bitmap-kuvaksi ja näytetään käyttäjälle.

5.4 Kuvien lukemisen toteutusratkaisut

Kuvien lukeminen on toteutettu sovellukseen niin, että sovellus lataa kiintolevyllä olevasta karttatiedostosta yhden kuvasiivun muistiin jokaista näytettävää pääakselia kohden. Kun sovellukseen lisätään uusi karttatiedosto, sovellus olettaa tämän olevan XY-pääakselin suuntainen. XZ- ja ZY-pääakselien suuntaiset näkymät on toteutettu niin, että XY-karttatiedoston avulla luodaan kyseiset karttapinot kiintolevylle asynkronoidusti.

Koska karttatiedoston sisältämät kuvasiivut ovat jonkin pääakselin suuntaisesti syvyysjärjestyksessä, sovellus voi lukea pääakselien suuntaisia kuvasiivuja vapaavalintaisesta syvyydestä ilman, että kuvasiivuja joudutaan muodostamaan dynaamisesti sovelluksen ajon aikana. Tämä mahdollistaa haluttujen kuvasiivujen näyttämisen käyttäjälle suhteellisen pienellä vasteajalla ilman, että karttatiedostoa joudutaan lataamaan kokonaisuudessaan välimuistiin. Toteutuksen heikkoutena on se, että kuvapinojen luominen vie huomattavasti aikaa ja että ylimääräiset karttatiedostot vievät tilaa kiintolevyltä.

6 Ohjelmointikäytänteet

Kehitetyn sovelluksen lähdekoodissa on pyritty noudattamaan C#-ohjelmointikielen käytänteitä [2]. Tekninen ohjaaja on tarkastanut käytänteiden noudattamista lähdekoodin katselmoinneissa. Katselmointien palautteet [12] ja [13] liittyivät pääasiassa muotoiluun tai kommentointiin, mutta joitakin huomioita projektiryhmä sai myös C#:n muiden koodauskäytänteiden noudattamisesta. Projektiryhmä otti katselmoinneissa saamansa palautteen huomioon ja korjasi puutteet lähdekoodiin. Pääasiassa projektiryhmä oli kuitenkin noudattanut koodauskäytänteitä projektin aikana.

6.1 Muotoilu-, nimeämis- ja kommentointikäytänteet

Lähdekoodi ja kommentit on kirjoitettu englanniksi Microsoftin nimeämiskäytänteitä [2] noudattaen. Nimiavaruudet, luokat, tyypit, rajapinnat, aliohjelmat, tapahtumat, vakiot ja julkiset attribuutit on nimetty *PascalCase*-tyylillä. Aliohjelmien parametrit ja lokaalit muuttujat on nimetty *camelCase*-tyylillä. Luokkien yksityisten attribuuttien nimen edessä on alaviiva. Rajapinnat alkavat I-etuliitteellä. Nimissä on vältetty sanojen välisiä alaviivoja, mistä ainoana poikkeuksena on tapahtumankäsittelijät, joissa WPF-elementin nimi on erotettu tapahtuman nimestä alaviivan avulla.

Jokaisen kooditiedoston alkuun on lisätty lisenssiteksti (The MIT License). Poikkeuksena ovat Visual Studion automaattisesti muodostamat tiedostot ja graafisen käyttöliittymän esittämiseen käytetyt XAML-tiedostot. Kooditiedostojen (`cs`-loppuiset tiedostot) rivit on pyritty pitämään 80–90 merkkiä pitkinä, mutta siitä on osin poikettu koodin luettavuuden vuoksi. Esimerkiksi pitkien funktiokutsujen osalta pilkkomista ei voinut tehdä, sillä ohjelmakoodi ei enää toimisi. XAML-tiedostoissa rivit ovat pidempiä, sillä niiden luettavuus olisi ollut liiallisen rivien pilkkomisen seurauksena huonompaa.

6.2 Lähdekoodin ryhmittelykäytänteet

Projektisuunnitelmassa [5] ei erikseen määritetty lähdekoodin ryhmittelykäytänteitä. Ohjelmisto on kuitenkin jaettu eri projekteihin erottamaan taustalla oleva tietorakenne ja ohjelman käyttöliittymä toisistaan. Lisäksi projektien sisältämiä lähdekooditiedostoja on tarpeen mukaan ryhmitelty eri hakemistoihin, esimerkiksi WPF:ään liittyvät erityyppisten ikkunoiden ym. tiedostot ovat omissa hakemistoissaan selkeyden vuoksi.

Lähdekoodi on jaettu seuraaviin projekteihin ja kansioihin:

- `CoreLibrary`-projekti sisältää sovelluksen pääasiallisen toimintalogiikan, jonka tietorakenteet ja tiedostojen lukemiseen käytetyt luokat ovat seuraavat:
  - `FileReaders`-hakemiston luokat avustavat tiedostojen lukemiseen liittyvissä tehtävissä.
  - `Model`-hakemiston luokat toteuttavat ohjelman taustalla olevan tietorakenteen.
- `GroundhogApp`-projekti sisältää käyttöliittymän toimintaan liittyvän toiminnallisuuden, sekä käyttöliittymän ulkonäön määrittelyn. Juurihakemistossa on pääikkunan lisäksi pääikkunan ViewModel sekä sovelluksen käyttöliittymän seuraavat yleiset tiedostot:
  - `Dialogs`-hakemistoon on laitettu itse toteutetut dialogi-ikkunat.
  - `Styles`-hakemistosta löytyvät sovelluksen käyttöliittymälle määritellyt elementtien tyylit.
  - `UserControls`-hakemistossa on itse toteutetut UserControl-elementit.
  - `ValueConverters`-hakemistossa on `IValueConverter`-rajapinnan toteuttavat arvojen muuntamisen hoitavat luokat.

6.3 Kehitysympäristö

Sovelluksen lähdekoodin kirjoittamiseen on käytetty Visual Studio 2019 -ohjelmaa ja Visual Studio Code -ohjelmaa. Ohjelmistokehyksenä on .NET Frameworkin versio 5.0 ja käyttöliittymän toteuttamiseen käytettiin WPF-kirjastoa. Sovelluksessa käytetyt muut kirjastot ovat Magick.NETiä ja CsvHelperiä lukuunottamatta .NET Frameworkin alaisia.

Kehitystä varten projektiryhmällä on ollut käytössään kaksi Windows 10 -työkonetta, yksi Windows 8.1 -työkone sekä Linux-työkone. Ohjelmointi on pääasiallisesti tehty Windows-työkoneilla, sillä WPF-ympäristö vaatii toimiakseen Windows-työkoneen. Vaatimusmäärittelyn [9] mukaisesti sovelluksen toiminta on rajattu Windows 10 -käyttöjärjestelmään, joten muita työkoneita ei kehityksen aikana tarvittu. Työkoneet olivat projektiryhmän omia laitteita, sillä projektin aikana vallinneiden koronarajoitusten takia jäsenillä ei ollut pääsyä projektin työhuoneisiin.

Versiohallintaan käytettiin Git-ohjelmaa ja Jyväskylän yliopiston GitLab-palvelua. Ohjelman lähdekoodin merkistökoodaus on UTF-8.

7 Testauskäytänteet ja tulokset

Luvussa kuvataan lyhyesti yksikkötestaukseen liittyviä testauskäytänteitä sekä järjestelmätestauksen testauskertojen tuloksia. Järjestelmätestauksen testauskäytänteitä on käsitelty järjestelmätestauksen testauskerran suunnitelmassa [8]. Lisäksi testauskäytänteitä ja -tuloksia on käsitelty projektiraportissa [4].

7.1 Yksikkötestaus

Lähdekoodin yksikkötestaukseen käytettiin Microsoftin MSTest-kirjastoa. Projektin aikana yksikkötestattiin joitakin omia `CoreLibrary`-projektin funktioita, mutta yksikkötestattuja funktioita ei päätynyt lopulliseen lähdekoodiin yhtäkään. Tästä johtuen yksikkötestien projekti päädyttiin poistamaan sovelluksen GitLab-varastosta (*commit: c499d628*).

Yksikkötestejä kirjoitettiin esimerkiksi tiedostojen tallentamiseen liittyviin asynkronisoituihin funktioihin, mutta tulosten synkronoinnista aiheutuneiden ongelmien vuoksi näistä funktioista luovuttiin. Yksikkötestejä kirjoitettiin myös omille luokille, jotka lopulta eivät päätyneet käyttöön esimerkiksi sen vuoksi, että näiden sijaan päätettiin käyttää valmiita kirjastoja.

7.2 Järjestelmätestauskertojen tulokset

Sovellukselle suoritettiin kaksi erillistä järjestelmätestauskertaa. Ensimmäinen järjestelmätestauksen testauskerta pidettiin 21.6.2021 – 24.6.2021 ja toinen testauskerta 29.6.2021. Testauskertojen välissä testiskenaarioihin tehtiin muutoksia, mistä johtuen huomioiden kokonaislukumäärät testauskerroilla poikkesivat toisistaan.

Järjestelmätestauksen testauskäytänteet ja siinä käytetyt testiskenaariot löytyvät Järjestelmätestauksen testauskerran suunnitelmasta [8]. Taulukkoon 7.1 on koottu testauskertojen huomioiden määrät. Yksityiskohtaisemmat tulokset huomioineen löytyvät järjestelmätestauksen testauskertojen raporteista [6] ja [7].

	Huomion tyyppi				
Järjestelmätestauksen testauskerta	**OK**	**Huomautus**	**Virhe**	**Ohitettu**	**Yhteensä**
1. Testauskerta	56	13	8	5	82
2. Testauskerta	64	2	6	0	72

Taulukko 7.1: Järjestelmätestauksen testauskertojen havaintojen määrät.

Ensimmäisellä järjestelmätestauksen testauskerralla poikettiin testauskerran suunnitelmasta siten, että käytetty käyttöjärjestelmänä oli muu kuin Windows 10 ja käytetty testidata poikkesi testaussuunnitelmasta. Lisäksi joitakin testejä jouduttiin ohittamaan, sillä testausympäristöstä puuttui xlsx-tiedostojen tallentamista varten oleva ohjelma. Tästä johtuen päätettiin järjestää myös toinen testauskerta. Lisäksi testiskenaariot tarvitsivat vielä muokkausta, sillä joitakin testiskenaarioita pystyi suorittamaan useammalla eri tavalla. Ensimmäisellä testauskerralla löydetyt virheet korjattiin sovelluksesta.

Toisella testauskerralla testauskerran suunnitelmasta jouduttiin poikkeamaan siten, että testiskenaarioihin merkattiin huomautus tai virhe jälkeenpäin. Tämä johtui siitä, että ongelmat vaativat useamman askeleen ilmentyäkseen ja tulivat esille vasta myöhemmin. Toisen testauskerran löydetyt virheet ja huomautukset jaettiin vakavuusluokiltaan vakaviin, helposti korjattaviin ja pieniin huomiohin. Vakavat ja helposti korjattavat huomiot korjattiin sovelluksesta. Havaittujen käytettävyysongelmien korjaus sovittiin jatkokehitykseen, ja yksi virheistä esimerkiksi johtui virheellisestä testiskenaariosta. Kolmatta testauskertaa ei järjestetty, koska korjatut virheet olivat helppo todeta korjatuiksi (esim. kirjoitusvirheet) tai virheiden vaikutus oli rajattu. Kyseisten osioiden toiminnan tarkastaminen onnistui ilman erillisen testauskerran järjestämistä.

8 Tavoitteiden toteutuminen

Luvussa kuvataan sovellukselle asetettujen tavoitteiden ja vaatimusmäärittelyssä esitettyjen vaatimusten toteutumista. Tämän lisäksi luvussa käsitellään myös epätyydyttäviä toteutusratkaisuja, jotka vaatisivat parannuksia.

8.1 Vaatimusten täyttyminen

Toteutusvaiheessa projektin aikaresurssit loppuivat kesken, ja osa vaatimuksista sovittiin tilaajan kanssa jatkokehitykseen. Lisäksi 3D-TIFF -karttatiedostojen tuki sekä 32-bittisten kuvien osalta harmaasävyskaalaus toteutuivat puutteellisesti. Tilaajan edustajien kanssa sovittiin vaatimusten rajaus siten, että projektin puitteissa pystyttiin toimittamaan toimiva prototyyppi. Toteutumattomia ja puutteellisia vaatimuksia on kuvattu tarkemmin vaatimusmäärittelyssä [9].

8.2 Sovelluksen epätyydyttävät ratkaisut

Sovelluksen toteutusratkaisuista epätyydyttävänä voidaan pitää muista kuvakulmista muodostettujen kuvapinojen luomisen hitautta. Verrattuna muihin samankaltaisiin ohjelmiin (kuten esimerkiksi ImageJ-ohjelmistoon) kuvapinojen luonti on erittäin hidasta. Ongelma johtunee pääasiassa siitä, että kehitetyssä sovelluksessa koko näytekuvaa ei ladata kerrallaan muistiin. Sen sijaan tarkasteltavia leikkauskuvia vaihdettaessa uusi siivu luetaan kovalevyltä ja muiden kuvapinojen luonti on toteutettu samalla periaatteella. Projektin puitteissa ei ehditty toteuttaa kuvan lukemista kokonaan muistiin kerrallaan, mutta sen toteuttaminen ja kuvapinojen luonnin muokkaaminen sitä tukevaksi saattaisi nopeuttaa kuvapinojen luontia huomattavasti.

Harmaasävyskaalauksen toimivuus 32-bittisten näytekuvien kohdalla kaipaa myös kehittämistä. 32-bittisillä liukuluvuilla minimi- ja maksimiarvojen alkuarvaukset ovat liian suuria tai pieniä, sillä oletuksiksi valittiin pienin ja suurin mahdollinen 32-bittinen liukuluku. Tämä on kuitenkin käyttäjälle hankalaa, sillä monesti kuvien kirkkausarvot eivät ole koko 32-bittisen liukuluvun arvoväliltä, vaan oikea arvoväli on huomattavasti pienempi. Kunnollisia minimi- ja maksimiarvoja varten kuvan tulisi olla kokonaan muistissa, jotta suurin ja pienin mahdollinen arvo voidaan laskea koko näytekuvasta. Vaihtoehtoisesti minimi- ja maksimiarvot pitäisi laskea uudestaan aina tarkasteltavaa siivua vaihdettaessa ja laskea kaikista nähtävillä olevista siivuista. Kumpaakaan ei ehditty toteuttaa projektin puitteissa.

3D-TIFF tiedostojen lukemisessa jouduttiin rajoittamaan tiedostojen koko 4GB. Tämä johtuu siitä, että käytettyä kuvakirjastoa Magick.NET ei saatu lukemaan yli 4GB kokoisia TIFF-tiedostoja. Yli 4GB TIFF-tiedostot käyttävät normaalista TIFF-tiedostosta poikkeavaa Big-TIFF-formaattia, jota kuvakirjasto ei suostunut lukemaan.

8.3 Toteutusratkaisujen valinta ja kehittyminen

Tilaajan edustajat antoivat ryhmälle täyden vapauden valita toteutusalustan sekä suunnitella graafisen käyttöliittymän sovelluksen muutamia reunaehtoja noudattaen. Sovelluksen tulee toimia millä tahansa perustason tietokoneella lataamatta suuria kolmiulotteisia kuvatiedostoja kerralla muistiin. Kuvatiedostojen kokorajoitukset päädyttiin ratkaisemaan vastaavalla tavalla kuin ImageJ-ohjelman *virtual stack* -toiminnossa.

Kehitettävän sovelluksen malliksi otettiin aihekuvauksen ja palavereissa käytyjen keskustelujen perusteella ImageJ/Fiji- ja PerGeos-ohjelmat. Fijistä mallia otettiin käyttöliittymään ohjelman menuvalikon laajennoksiin perustuvasta modulaarisesta rakenteesta. PerGeos-ohjelmasta käyttöliittymään otettiin mallia ohjelman hallitsevasta, mutta vapaasti asemoitavasta pääikkunasta. Toisaalta komentovalikosta avattavat modaaliset ikkunat hieman rikkovat pääikkunaan pohjautuvaa käyttöliittymä arkkitehtuuria.

Ikkunoinnin jälkeen suunniteltiin ohjelman ulkoiset rajapinnat valitsemalla käyttöliittymälle AvalonDock-kirjasto. Sovelluksen käyttöliittymän suunnittelun yhteydessä kokeiltiin myös AvaloniaUI:n ja sen Dockin yhdistelmää. AvaloniaDockista luovuttiin nopeasti, mutta se tarjosi kiinnostavan vaihtoehdon Windows-koneisiin rajoittuvalle WPF-teknologialle. Käyttöliittymän MVVM-arkkitehtuuri toteutui suunnitellusti ilman ulkoisia kirjastoja.

Kuvankäsittelyyn valittiin Magick.NET-kirjasto, jonka valinnalla saavutettiin laaja kuvaformaattien tuki sekä valmiit työkalut kuvien jatkokäsittelyyn, esimerkiksi harmaasävyskaalan muuttamiseen. Kuvankäsittelyyn harkittiin aluksi kirjastoja Emgu CV ja pi2, mutta kumpaakaan ei voitu suoraan hyödyntää tarttuvan GPL-lisenssin takia. Tilaajan edustajista Miettinen on yksi pi2-kirjaston kehittäjistä ja hän olisi tarvittaessa voinut lisensoida uudelleen osan kirjoittamistaan ohjelmaosioista.

Liitoskohtien kolmiulotteiseen visualisointiin kartoitettiin OpenGL-kirjastoja, joista SharpGL-kirjasto vaikutti alkuun lupaavimmalta. Käyttöönotossa kesti suunniteltua kauemmin, joten kolmiulotteisen tilakartan korvaajaksi kehitettiin liitoskohtia ryhmittelevä liukusäädin. Sovelluksen päämääränä olikin juuri liitoskohtien merkitseminen ja hallinta, minkä perusteella muodostettiin sovelluksen luokkarakenne.

9 Ideoita jatkokehitykseen

Luvussa kuvataan sekä jatkokehitykseen sovittuja ominaisuuksia että ideoita siitä, mihin suuntaan sovellusta voisi lähteä kehittämään jatkossa. Näiden lisäksi myös luvun 8.2 epätyydyttävät ratkaisut ovat olennaisia jatkokehityksen kohteita. Luvusta löytyy myös jatkokehitysympäristön pystytysohjeet.

9.1 Jatkokehitysympäristön pystytysohjeet

Projektin lähdekoodi sijaitsee Jyväskylän yliopiston GitLab-palvelussa yksityisenä repositoriona osoitteessa `gitlab.jyu.fi`. GitLabin avulla projektitiedostot voidaan ladata pakattuina tiedostoina ja sitä voidaan käyttää erilaisten Git-pohjaisten versiohallintasovellusten kanssa. Projektin päättyessä sovelluksen versiosta on yksi päähaara.

.NETin version 5.0 ohjelmistokehykseen perustuva sovelluksen GroundhogApp-WPF-projekti vaatii vähintään Community-ilmaisversion Visual Studio 2019 -kehitysympäristöstä. Lähdekoodit sisältävä Visual Studio -projekti avataan *Open Project/Solution* -toiminnolla valitsemalla `source`-kansiosta `Groundhog.sln`-tiedosto. Toimiakseen Visual Studioon on asennettava paketti `.NET desktop development –workload`. Mikäli tätä ei ole valmiiksi asennettuna, Visual Studio ehdottaa pakettia asennettavaksi Visual Studio Installer -palvelun kautta.

Testausympäristön pystytyksen jälkeen ohjelmakoodi on käännettävissä Visual Studiolla. Ohjelman käynnistysolion pitäisi olla oletuksena asetettuna `GroundhogApp.App`.

9.2 Kaksiulotteisen karttakuvakomponentin jatkokehitysideat

Kaksiulotteiseen karttakuvakomponenttiin (katso luku 4.2) voisi lisätä seuraavia ominaisuuksia:

- Karttakuvien tarkentamista, keskittämistä, liikuttamista ja selaamista voisi parantaa pikanäppäimillä ja käsittelemällä laajemmin käyttäjän syötteitä. Karttakuvien koordinaatisto kannattaa ehkä ensin päivittää käsittelemään liukulukuja.
- Liitoskohtia voisi pystyä liikuttamaan hiirellä. Pisteen siirtämisen sijaan voisi suunnitella tarkemmin liitoskohtien muodon, sijainnin ja asennon määrittämisen.
- Käyttöliittymän tulisi tarjota mahdollisuudet karttakuvien uudelleen asetelluun ja irroittamiseen, kuten liitostiedostojen kohdalla. Näytettävät kuvakulmat varaavat käyt-

töliittymästä tilaa dynaamisesti sitä mukaan kun eri pääakselien suuntaisia karttapinoja luodaan.
- Karttakuvia tulisi voida määrittää mistä tahansa kuvakulmasta.
- Karttakuvia tulisi pystyä määrittämään 3D-avaruuden kohtaan ja sen tietoja pystyy jälkeenpäin muuttamaan asetusnäkymän kautta.
- Asetusnäkymiä käytetään monessa kohtaa ohjelmaa, joten käyttöliittymän komponenteilla voisi olla yksilölliset asetusnäkymät.

9.3 Kolmiulotteinen karttakuvakomponentti

Prototyyppi sijoitettiin erillisenä projektina versiohallintaan (*commit: 72c9693e*). Komponentti tarkoitettiin palaverissa esiteltäväksi prototyypiksi, jota voidaan vielä parantaa ennen käyttöönottoa. Tärkeimpänä ominaisuutena on liitoskohtien muodon, sijainnin ja asennon määrittäminen laajennettavasti sekä niiden leikkauspisteiden laskeminen.

Komponentti esittää kuvatiedostot tasoihin liitettävinä tekstuureina, joka on toteutustavaltaan yhteensopiva kaksiulotteisen karttakuvakomponentin kanssa. Komponenttia voisi laajentaa esittämään kuvapisteiden väriarvot suoraan alkuperäisestä volyymidatasta.

9.4 Muut sovelluksen toiminnallisuuteen liittyvät jatkokehitysideat

Sovellussuunnitelmien näkökulmasta keskeisiä parannusehdotuksia ovat seuraavat:

- Käyttöliittymäsuunnitelmassa esitetyn dataliitoksia sisältävän listanäkymän yhteyteen voisi lisätä painikkeet kontekstivalikon toiminnoille. Painikkeet kohdistuisivat valittuna olevaan dataliitokseen.
- Käyttöliittymäsuunnitelmassa esitetyn uuteen ikkunaan aukeavat näkymät voisivat avautua pääikkunaan, kuten muutkin näkymät. Vaihtoehtoisesti uudet asetusnäkymät voisivat avautua käyttöliittymän kyseisten komponenttien tilalle.
- Käyttöliittymäsuunnitelman kuvassa liitoskohtia osoittavat liukusäätimet voisi sijoittaa niitä vastaaviin kaksiulotteisiin karttakuvakomponentteihin. Karttakuvia voisi hallita silloin kokonaan erillään toisistaan.

9.5 Sovelluksen lähdekoodin jatkokehitysideat

Karttakuvia edustavat siivut voisi sijoittaa dataliitosten tapaan tietorakenteeseen. Silloin käyttöliittymän kautta voidaan lisätä dynaamisesti useampia karttakuvia eri kuvakulmista.

Dataliitosten metatiedot voisi sijoittaa samaan säiliöluokkaan kuin näytteen metatiedot. Liitosdata saadaan erotettua metatiedoista, jolloin metatiedot olisivat yhdessä paikassa ja luokkien välillä olisi vähemmän riippuvuuksia.

Karttakuvat ja liitosdatat voisivat huolehtia omasta tiedoston käsittelystään staattisten apufunktioiden sijaan. Vaihtoehtoisesti staattiset apufunktiot voisi sijoittaa niitä käyttäviin luokkiin.

Lähdekoodia voisi parannella refaktoroimalla esimerkiksi kuvien lukemiseen liittyviä aliohjelmia `MapReader`-luokassa. Tällä hetkellä jokaiselle eri tiedostotyypille on oma aliohjelmansa, mutta tämä voitaisiin muuttaa yhdeksi aliohjelmaksi ja välttää toistoa. Aliohjelmissa on tosin tehty eri tiedostomuotojen lukemisen optimointia, mikä tulee refaktorointia tehdessä huomioida.

Lähdekoodin poikkeuksenkäsittelyä tulee myös parantaa jatkokehityksessä. Poikkeusviestien näyttämistä varten on globaali käsittelijä, jonka avulla käyttäjälle näytetään virheviestin sisältävä ponnahdusikkuna. Tämän voisi muuttaa siten, että globaalissa poikkeuksenkäsittelijässä käsiteltäisiin vain itse ohjelmoidut poikkeusluokat ja yleisten ohjelmointikielen poikkeusten virheilmoitusten näyttäminen hoidettaisiin alkuperäisessä tapahtumankäsittelijässä. Tällöin ohjelmoijan ei tarvitsisi muistaa, mitkä C#-ohjelmointikielen poikkeukset saattavat vaatia erillistä käsittelyä. Tässä täytyy kuitenkin huomioida se, että käyttöliittymän ominaisuuksia (kuten ponnahdusikkunoita) on mahdollista luoda vain käyttöliittymään liittyvissä luokissa luvussa 5.1 kuvatulla tavalla. Lisäksi joissain tapauksissa esimerkiksi funktioiden argumenttien tarkastaminen ennen funktiokutsuja if-lauseilla voisi olla järkevämpää kuin poikkeuksen heittäminen.

Lähdekoodissa liitoskohtien sijaintia käsitellään kolmiulotteisena pisteenä. Sovelluksen suunnittelussa puhuttiin myös liitoskohdista viivoina ja tasoina, joita ei toteutettu projektissa. Nykyinen toteutustapa ei mahdollista uusien liitoskohdan muotojen lisäämistä helposti. Siksi olisi hyvä jos `AttachedData`-luokassa liitoskohdan sijainti esitettäisiin kolmiulotteisen avaruuden pisteen sijaan rajapintaluokkana. Tällöin uusien liitoskohtien muotojen lisääminen jatkokehityksessä olisi helpompaa.

10 Yhteenveto

Groundhog-projekti kehitti kevään 2021 Sovellusprojekti-kurssilla Geologian tutkimuskeskukselle multimodaalisen tutkimusdatan hallintaohjelmiston tukemaan tomografialaboratorion näytteiden analyysiä. Prototyyppi helpottaa erityyppisten tutkimusdatojen yhdistämistä, hallintaa ja visualisointia, mutta sovellusta tulee vielä kehittää ennen kuin se voidaan ottaa laajempaan tuotantokäyttöön.

Projektiryhmän kehitettämällä prototyyppillä voi tarkastella isokokoisia kolmiulotteisia karttakuvia perustietokoneella. Projektissa kehitetylle sovellukselle toteutettiin käyttöliittymä, joka tarjoaa käyttäjälle pääsyn kaikkiin ohjelman toimintoihin. Toiminnoista olennaisimmat ovat karttatiedostojen selaaminen, liitostiedostojen lisääminen ja liitostiedostojen näyttäminen. Sovelluksen toteutuksessa on käytetty MVVM-arkkitehtuuria, jonka avulla käyttöliittymä on erotettu toimintalogiikasta vastaavasta tietorakenteesta. Tämä mahdollistaa käyttöliittymän ja tietorakenteiden muokkaamisen erillisinä kokonaisuuksinaan jatkokehityksessä.

Tärkeimmät sovellukselle asetetut tavoitteet toteutuivat projektissa. Kaikkia vaatimusmäärittelyssä asetettuja ominaisuuksia ei ehditty projektissa toteuttamaan. Projektiryhmä suoritti kahdesti suunnittelemansa järjestelmätestauskerran. Tämän lisäksi projektin tekninen ohjaaja katselmoi lähdekoodin kahdesti. Sovellus hyväksyttiin niin tilaajan kuin teknisen ohjaajankin toimesta.

Olennaisimmat jatkokehityksen kohteet ovat luvun 8.1 kuvailema 3D-TIFF -karttatiedostojen tuen lisääminen ja luvun 8.2 epätyydyttävien ratkaisujen parantelu. Luvussa 9 on kuvattu muita jatkokehityksen kohteita.

Lähteet

[1] Lauri Antila, Outi Hilola, Antti Kauppi ja Anne Vaarala. Kodavi-projekti, Sovellusraportti. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2020.

[2] Krzysztof Cwalina et al. Naming Guidelines. `https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines`. Microsoft, 2008.

[3] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra. Peltihamsteri-projekti, Sovellusraportti. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2019.

[4] Iiro Iivanainen, Harri Linna, Jere Pakkanen ja Riikka Vilavaara. Groundhog-projekti, Projektiraportti. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[5] Iiro Iivanainen, Harri Linna, Jere Pakkanen ja Riikka Vilavaara. Groundhog-projekti, Projektisuunnitelma. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[6] Iiro Iivanainen, Harri Linna, Jere Pakkanen ja Riikka Vilavaara. Groundhog-projekti, Järjestelmätestauksen ensimmäisen testauskerran raportti. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[7] Iiro Iivanainen, Harri Linna, Jere Pakkanen ja Riikka Vilavaara. Groundhog-projekti, Järjestelmätestauksen toisen testauskerran raportti. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[8] Iiro Iivanainen, Harri Linna, Jere Pakkanen ja Riikka Vilavaara. Groundhog-projekti, Järjestelmätestauksen testauskerran suunnitelma. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[9] Iiro Iivanainen, Harri Linna, Jere Pakkanen ja Riikka Vilavaara. Groundhog-projekti, Vaatimusmäärittely. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[10] Iiro Iivanainen. Groundhog-projekti, Sovelluksen käyttöohjeet. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[11] Jukka-Pekka Santanen. Tietotekniikan Sovellusprojektien ohje. `http://www.mit.jyu.fi/palvelut/sovellusprojektit/projohje.pdf`. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2017.

[12] Juuso Tuononen. Groundhog-projektin 1. koodikatselmuksen kirjalliset huomiot. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.

[13] Juuso Tuononen. Groundhog-projektin 2. koodikatselmuksen kirjalliset huomiot. Jyväskylän yliopisto, informaatioteknologian tiedekunta, 2021.