TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

LAURI LAAKSONEN

LIIKENNEOHJAUSJÄRJESTELMÄN VMX-KAMERAINTEGRAATIO

Diplomityö

Tarkastaja: Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
28. maaliskuuta 2018

TIIVISTELMÄ

LAURI LAAKSONEN: Liikenneohjausjärjestelmän VMX-kameraintegraatio
Tampereen teknillinen yliopisto
Diplomityö, 47 sivua
Helmikuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja:

Avainsanat: suoratoisto, videokoodekki, kamerajärjestelmä

Suomen liikennekeskukset vastaavat tieverkostojen tilan valvonnasta. Liikennekeskuksissa työskentelevät tieliikennepäivystäjät käyttävät tähän tehtävään T-LOIK-tietojärjestelmää. Tietojärjestelmä koostuu useasta eri työkalusta, joista yksi on liikennekameroiden kuvien katseluun tarkoitettu kameratyökalu.

Kameratyökalu näyttää liikennekameroiden ottamia kuvia ja videoleikkeitä. Tämän työn tavoitteena on kehittää kameratyökalua uusien toiminnallisuuksien muodossa. Pelkkien staattisten kuvien ja ladattujen videoleikkeiden lisäksi kameratyökaluun toteutetaan tuki suoratoistettavan videokuvan esittämiseen sekä liikennekameroiden ohjaamiseen, jolloin tieliikennepäivystäjien voivat hahmottaa liikennetilannetta paremmin.

Tässä diplomityössä käydään läpi edellä mainittujen suoratoisto- ja kameraohjaus-toimintojen suunnittelu- ja toteutusvaihe ohjelmistotuotannon näkökulmasta. Ensin kuvataan nykyisen tietojärjestelmän osia, jonka jälkeen selitetään hieman videoiden suoratoistossa käytettävää videopakkaamisen teoriaa. Tämän jälkeen käydään läpi toimintojen suunnittelu- ja toteutusprosesseja kuvaamalla erilaisia ratkaisuja prosessien aikana ilmenneisiin haasteisiin.

Työn lopussa arvioidaan vielä toteutusprosessin tuloksia eli millaisia suoratoisto- ja kameraohjaus-toiminnoista lopulta tuli vallitsevien reunaehtojen puitteissa. Arvioinnissa keskitytään siihen, miten toteutus erosi alkuperäisestä suunnitelmasta ja nostaa esiin vaihtoehtoisia ratkaisuja, jotka olisivat ehkä soveltuneet joidenkin ongelmien ratkaisuiksi paremmin.

ABSTRACT

LAURI LAAKSONEN: VMX camera integration of traffic control system
Tampere University of Technology
Master of Science Thesis, 47 pages
February 2018
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner:

Keywords: streaming, video codec, camera system

In Finland, road traffic centers are responsible for monitoring road traffic flow. Road traffic operators in these road traffic centers use special information system called T-LOIK in order to control traffic different traffic situations. T-LOIK consists of multiple different components. One of the components is a camera tool which is responsible for displaying images taken by traffic cameras.

The main purpose of the camera tool is to display images and recorded videoclips. The implementation part of this master's thesis aims to develop camera tool further by implementing new features such as video streaming and camera control. By having an option to view video streams of traffic cameras and control camera angles, traffic operators are able to understand better the general view of traffic situation.

This master's thesis analyzes the design and implementation processes from the perspective of software development. It starts by describing the current system where the new features will be implemented. After that, some key parts of video compression are explained so that the full context of the feature integration can be understood. Finally, the thesis goes over the design and implementation processes by describing some problems encountered during the processes and explaining chosen solutions for them.

At the end of this master's thesis, the results of the implementation process are evaluated. The evaluation focuses on the differences between initial design and the implemented one. Additionally, some alternative solutions to the used ones are presented and thoughts on how the implementation of the features could have been done better.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CLR	Common Language Runtime
DXVA	DirectX Video Acceleration
HHJ	Häiriöhavaintojärjestelmä
HTTP	Hypertext Transfer Protocol
T-LOIK	Tieliikenteen ohjauksen integroitu käyttöliittymä
JMS	Java Message Service
JPEG	Joint Photographic Experts Group
LOTJU	Liikenteen olosuhdetietojärjestelmä
MJPEG	Motion Joint Photographic Experts Group
OSI	Open Systems Interconnection Reference Model
REST	Representational State Transfer
RTP	Real-Time Transport Protocol
TCP	Transmission Control Protocol
VMX	Telesten kehittämä keskitetty kameranhallintajärjestelmä
WPF	Windows Presentation Foundation

1. JOHDANTO

Suomen laajan tieverkoston valvonnasta vastaa pääosin Liikennevirasto. Tämän tehtävän hoitamiseksi Liikennevirasto on aloittanut hankkeen uuden tietojärjestelmän rakentamiseksi, jonka avulla liikenteen valvontaa voidaan sujuvasti suorittaa. Toteutettavan tietojärjestelmän nimi on T-LOIK eli tieliikenteen ohjauksen integroitu käyttöliittymä.

Järjestelmän yksi olennainen osa on kameratyökalu. Kameratyökalua käytetään liikennekameroiden ottamien kuvien katseluun, joiden perusteella tieliikennepäivystäjä voi valvoa liikennettä ja tehdä harkittuja päätöksiä kamerakuvissa havaitun tilanteen mukaisesti.

Tämän työn tavoitteena on tuoda kameratyökaluun liikennekameroiden keskitetystä VMX-kamerajärjestelmästä uusia toiminnallisuuksia, joiden avulla voidaan esittää reaaliaikaisempaa, suoratoistettavaa kamerakuvaa sekä ohjata liikennekameroita. Tätä kautta voidaan tarjota tieliikennepäivystäjille mahdollisuus saada tarkempaa tietoa liikennetilanteista.

Tämä diplomityö on tehty osana T-LOIK-tietojärjestelmän kameratyökalun kehitystä. Kameratyökalun kehityksestä osana T-LOIK-tietojärjestelmää on vastuussa ohjelmistoyritys Bitwise Oy. Diplomityön tarkoituksena on toteuttaa edellä mainitut uudet toiminnallisuudet eli suoratoistetun videokuvan esitys ja liikennekameroiden ohjaaminen kameratyökaluun ja esittää näiden toiminnallisuuksien suunnittelu- ja toteutusprosessien keskeisimmät vaiheet yleisesti ohjelmistokehityksen näkökulmasta.

Diplomityössä käydään läpi ensin jo olemassa olevan T-LOIK-järjestelmän kameratyökalun toiminnallisuuksia sekä kameratyökalun käyttämien muiden komponenttien toimintaa työn taustan hahmottamiseksi. Lisäksi käsitellään hieman projektissa käytettyjä teknologioita, joiden puitteissa uudet toiminnallisuudet on saatava toteutettua.

Luku 3 käsittelee yleisesti videodatan pakkaamista ja purkamista sen välittämiseksi verkon yli. Luvussa kuvataan datan pakkaamisprosessia osana videokuvan suoratoistoa sekä annetaan yleiskäsitys muutamasta olennaisesta pakkaamisstandardista VMX-kamerajärjestelmän tuottaman videodatan kannalta.

Viimeiset luvut taas keskittyvät yksityiskohtaisemmin toiminnallisuuksien toteutusprosessiin ohjelmistokehityksen näkökulmasta. Ne pyrkivät kuvaamaan toteutuksessa käytettyjä teknisiä ratkaisuja sekä ilmenneitä haasteita.

2. LIIKENNEOHJAUS- JA VMX-KAMERAJÄRJESTELMÄ

Suomessa tieliikenteen tilaa seuraavat tieliikennekeskuksissa työskentelevät tieliikennepäivystäjät. Näissä Liikenneviraston alaisissa keskuksissa on käytössä liikenneohjausjärjestelmä, jonka avulla tieliikennepäivystäjät hoitavat erilaisia tieliikenteeseen liittyviä tehtäviä ja seuraavat liikenteen kulkua ympäri maata. Liikenneohjausjärjestelmä koostuu useasta eri sovelluksesta, joista yksi on liikennekameroiden tuottamien kuvien katseluun toteutettu sovellus. Tähän sovellukseen viitataan tämän työn yhteydessä termillä kameratyökalu. Liikenneohjausjärjestelmään on tarkoituksena integroida toinen, keskitetty kamerajärjestelmä, johon teiden varsilla olevat liikennekamerat on kytketty. Tätä kamerajärjestelmää kutsutaan VMX-kamerajärjestelmäksi. Seuraavat aliluvut pyrkivät avaamaan näitä kahta järjestelmää omina kokonaisuuksinaan ennen niiden varsinaista integraatiota toisiinsa.

2.1 T-LOIK-liikenneohjausjärjestelmä

Aiemmin tieliikennekeskuksissa työskentelevillä tieliikennepäivystäjillä oli käytössään joukko erilaisia sovelluksia eri tehtävien hoitoon. Sovellusten joukkoa haluttiin yksinkertaistaa tuomalla ne yhden kokonaisuuden alle. Tästä sovelluskokonaisuudesta käytetään nimitystä tieliikenteen ohjauksen integroitu käyttöliittymä. Tämän työn puitteissa tästä ohjelmistosta käytetään lyhennettä T-LOIK. Nykyään tieliikennepäivystäjät käyttävät T-LOIKia, joka käyttäjän näkökulmasta käyttäytyy kuin yksi ainoa sovellus, jossa on uusien toiminnallisuuksien lisäksi myös aiemmin käytettyjen työkalujen toiminnallisuuksia.

Todellisuudessa T-LOIK on edelleen useasta eri alisovelluksesta koostuva kokonaisuus. Se noudattaa korkean tason arkkitehtuuriltaan asiakas–palvelin-mallia. Asiakas–palvelin -malli on tapa, jolla hajautetun järjestelmän eri komponentit voidaan jakaa kahteen eri kategoriaan: asiakas- ja palvelinsovelluksiin. T-LOIKissa asiakkaita ovat tieliikennepäivystäjien käytössä olevat käyttöliittymäsovellukset, joihin lukeutuu myös seuraavassa aliluvussa esiteltävä kameratyökalu. Nämä käyttöliittymäsovellukset kommunikoivat asiakas-palvelin-mallin mukaisesti T-LOIK-kokonaisuuteen toteutettujen palvelinsovellusten kanssa. Palvelinsovelluksilla on kullakin oma vastuualueensa, esimerkiksi kameratyökalu (asiakas) kommunikoi pääasiassa vain kamerapalvelimen kanssa pyytämällä sen määrittämän rajapinnan kautta uusia tieliikennekuvia sekä muuta kameroihin liittyvää tietoa.

2.2 Kameratyökalu

Kameratyökalu on yksi T-LOIKin osa, joka on tieliikennepäivystäjien käytössä käyttöliittymäsovelluksen muodossa. Kameratyökalu on Windows-käyttöjärjestelmälle kehitetty työpöytäsovellus, jonka pääasiallisena tarkoituksena on tarjota tieliikennepäivystäjille mahdollisuus tarkastella ajoteiden varsille sekä tunneleihin asennettujen liikennekameroiden ottamia kuvia tai tallentamia nauhoitteita. Näiden kuvien ja nauhoitteiden avulla tieliikennepäivystäjät voivat arvioida liikenteen nykyistä tilaa. Tieliikennepäivystäjät voivat esimerkiksi havaita kuvista tai nauhoitteista onnettomuuden, josta he voivat edelleen lähettää tiedotteita muille viranomaisille ja medialle.

2.2.1 Kameratyökalun nykyiset ominaisuudet

Ennen VMX-kameraintegraatiota kameratyökalu tuki vain yksittäisten kuvien katselua viimeisimmältä tunnilta. Suoraa kamerakuvaa ei siis voitu katsoa kameratyökalun kautta, vaan siihen käytetiin erillistä Teleste-nimisen yrityksen kehittämää VMX-asiakassovellusta, joka on T-LOIKiin kuulumaton sovellus. VMX-asiakassovellukseen ja koko VMX-kamerajärjestelmään keskitytään enemmän seuraavassa aliluvussa. Puhuttaessa kameran avaamisesta kameratyökaluun avaa kameran valitseminen näkymään kameran viimeiseltä tunnilta ottamat kuvat, jotka asetetaan aikajanalle. Näitä kuvia sitten näytetään peräjälkeen aikajanan alusta loppuun. Tästä aikajanalle asetettujen kuvien muodostamasta sarjasta käytetään nimitystä kuva-aikasarja.

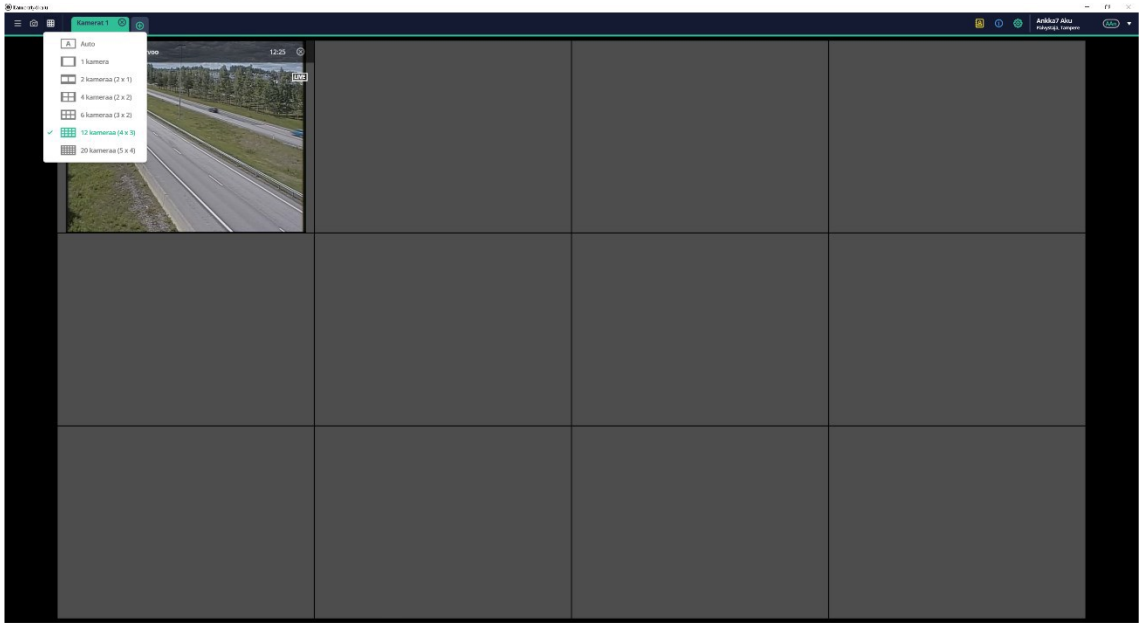
Lukuun ottamatta videotallenteita kameratyökalu ei pystynyt esittämään niin sanottua liikkuvaa kuvaa, vaan staattisia JPEG-kuvia eri ajankohdilta. JPEG eli Joint Photographic Experts Group on yleinen pakatun kuvadatan tallennusformaatti. JPEG-formaattiin ja erityisesti siitä kehitettyyn videonpakkausformaattiin (Motion JPEG) tutustutaan paremmin luvussa 3. Yksittäisen kameran kuva-aikasarja muodostetaan siten, että tien varrella olevasta kamerasta otetaan kuvankaappaus muutaman minuutin välein. Tämä kuvankaappaus tallennetaan kovalevylle JPEG-formaatissa. Liikennekamera lähettää jatkuvasti kuvaa, mutta videokuvasta hyödynnetään vain yksittäisiä kuvia. Kun kuvankaappaus kamerakuvasta otetaan, samalla tallennetaan myös ajankohta, jolloin tiedetään, mihin kohtaan aikajanaa kuva lopuksi kuva-aikasarjassa sijoitetaan. Aina, kun uusi kuvankaappaus on otettu ja tallennettu, vanhin kuva kuva-aikasarjassa pudotetaan pois ja uusin otettu kuva lisätään kuva-aikasarjaan. Kuva-aikasarjan kuvien aikajärjestys säilytetään kuva-aikasarjaa päivitettäessä.

Edellisessä kappaleessa selitetty kuvankaappausten ottaminen ja tallentaminen eivät ole kamerapalvelimen eli pääasiallisesti kameratyökalun käyttämän palvelinkomponentin vastuulla. Kyseinen tehtävä on LOTJUn vastuulla. LOTJU eli liikenteen olosuhdetietojärjestelmä koostuu keruu- ja metatietosovelluksista, joista oleellisempia sovelluksia kameratyökalun kannalta ovat kamerakeruu-, kameravarasto- ja metatietosovellukset.

LOTJUn kamerakeruujärjestelmä suorittaa kuvankaappaukset ja kameravarastointisovellus tallentamisen.

Kuva-aikasarjat voivat tukea myös suunnan valintaa riippuen siitä, kuvaako valittu kamera moneen eri suuntaan. Yleensä liikennekamerat ovat jossakin tietyssä esiasennossa, joka kertoo, mihin suuntaan kamera kuvaa. Tieliikennepäivystäjä voi näin ollen vaihtaa kuva-aikasarjan kuvia suunnan mukaan valitsemalla suunnan, jolloin kaikki kuva-aikasarjan kuvat korvaantuvat valittuun suuntaan otetuilla kuvilla.

Yksittäisistä kuva-aikasarjoista voidaan myös muodostaa suurempia kokonaisuuksia. Näistä usean kuva-aikasarjan muodostamista sarjoista käytetään nimitystä diasarja. Tieliikennepäivystäjät voivat luoda diasarjoja kameratyökalun avulla valitsemalla kamerat, joiden kuvat halutaan diasarjaan liittää. Diasarjaan muodostetaan tämän jälkeen valittujen kameroiden kuva-aikasarjat niin, että kuva-aikasarjojen aikaleimat täsmäävät keskenään. Diasarjan kaikki kuva-aikasarjat näkyvät samassa näkymässä yhtä aikaa ja kuva-aikasarjojen aikajanat täsmäävät keskenään. Näin ollen näytettävät kuvat ovat samalta ajanhetkeltä, jolloin tieliikennepäivystäjä voi nopeasti hahmottaa liikenteen kokonaiskuvan valitsemaltaan alueelta, jossa valitut kamerat sijaitsevat.



Kuva 1. *Kameratyökalun ruudukko kuva-aikasarjoille*

Kameratyökalu tukee myös videotallenteiden toistamista. T-LOIK vastaanottaa videotallenteita tikettien mukana. Tiketit ovat tilanne- tai tehtäväkuvauksia erilaisista liikennetilanteista. T-LOIKiin kuuluu toinen tieliikennepäivystäjien käytössä oleva käyttöliittymäsovellus, josta käytetään nimitystä päivystystyökalu. Päivystystyökalu pitää kirjaa edellä mainituista tiketeistä ja tarjoaa erilaisia toimintoja tiketeille riippuen niiden tyypistä. Esimerkiksi liikennetiedotteet ovat yksi monista tikettityypeistä, josta voidaan päi-

vystystyökalun kautta luoda jatkotiedotteita. Tikettejä luodaan kahdella tapaa: tieliikennepäivystäjien tekeminä tai jonkin integroidun palvelun kautta. Häiriöhavaintotiketit ovat yksi tikettityyppi, joita päivystyspalvelin (palvelinkomponentti, jonka kanssa päivystystyökalu kommunikoi) vastaanottaa, ja jotka se tiedottaa edelleen päivystystyökalulle. Päivystystyökalun vastaanottaessa tiedon uudesta häiriöhavainnosta se luo uuden tiketin tikettilistaan vastaanotetuista häiriöhavaintotiedoista. Häiriöhavaintoja luodaan normaalista poikkeavista liikennetilanteista. Esimerkiksi, jos ajoneuvon havaitaan liikkuvan normaalia liikenteen ajosuuntaa vastakkaiseen suuntaan tunnelissa, tilanteesta luodaan häiriöhavainto. Tunnelissa voi olla asennettuna liikennekamera, joka tallentaa kamerakuvaa kovalevylle. Tällaisilla tallenteilla on myös oma tunnisteensa, joka voidaan liittää lähetettäviin häiriöhavaintotietoihin. Jos häiriöhavainnosta on saatu tallennettua videotallenne, päivystystyökalun häiriöhavaintotiketti tarjoaa tieliikennepäivystäjälle toiminnon, jolla päivystystyökalu käskyttää kameratyökalua toistamaan kyseisen videotallenteen.

Vastaanotettuun häiriöhavaintoon voi olla liitettynä videotallenne kyseisestä tilanteesta. Kameratyökalun yhtenä pääasiallisena tehtävänä on näiden tallenteiden haku häiriöhavainnon mukana tulleen tallennetunnisteen avulla sekä itse tallenteen toisto. Videotallennetta ei toisteta suoraan verkon yli, vaan ensin se ladataan ja tallennetaan työaseman kiintolevylle. Kun lataus on valmis, kameratyökalu avaa uuden välilehden, johon luodaan mediatoistin. Lopuksi mediatoistin avaa ja toistaa ladatun videotallenteen kyseisestä häiriöhavainnosta.

Tyypillisessä käyttötapauksessa tieliikennepäivystäjä avaa ainakin yhden kamerakuvan kameratyökalun tarjoamasta kameralistasta välilehteen. Hän voi esimerkiksi avata kaikki kamerat yhden tien varrelta, ja tällä tapaa hän kykenee seuraamaan koko tietä ilman tarvetta siirtyä näkymästä toiseen. Tämä helpottaa huomattavasti liikennetilanteen kokonaistilanteen hahmottamista.

Edellä mainittu kameratyökalun välilehti on toiminto, jonka avulla tieliikennepäivystäjä voi avata useita välilehtiä, joista jokaiseen voidaan avata useita kuva-aikasarjoja tai yksittäisiä videotallenteita. Tieliikennepäivystäjä pystyy katsomaan jopa 20 kuva-aikasarjaa samanaikaisesti yhdeltä välilehdeltä. Pelkkien kuvien esittäminen käyttöliittymässä ei ole kovin raskas operaatio, joten kameratyökalun käyttö pysyy sulavana, vaikka maksimimäärä kuva-aikasarjoja olisi avattu. Avattujen kuva-aikasarjojen kokonaismäärä on rajattu kameratyökalussa 64 kuva-aikasarjaan, jolla pyritään takaamaan kameratyökalun sulava toiminta. Eroten kuva-aikasarjojen välilehdistä häiriöhavaintojen videotallenteet avataan aina erillisiin välilehtiin eli jokaisella häiriöhavaintovälilehdellä on yksi mediatoistin, joka toistaa videotallennetta.

2.2.2 Kameratyökalussa käytetyt teknologiat

Kameratyökalu on ainoastaan Windows-käyttöjärjestelmille toteutettu sovellus. Kameratyökalun kehityksen suunnitteluvaiheessa pääasialliseksi sovelluskehykseksi valittiin

Microsoftin kehittämä .NET Framework -kirjasto, joka tarjoaa monia hyödyllisiä palveluja Windows-pohjaisten työpöytäsovellusten kehittämiseen. .NET Framework on Windowsille toteutettu komponenttikirjasto, joka koostuu kahdesta pääkomponentista[1]. Ensimmäinen komponentti on ajoympäristö, josta käytetään englanninkielistä nimeä Common Language Runtime (CLR). CLR hallinnoi suoritettavaa koodia ajonaikaisesti tarjoten samalla erilaisia palveluita, kuten esimerkiksi muistin- ja säikeenhallintaa. Näin ollen kehittäjän ei tarvitse huolehtia itse esimerkiksi varaamaansa muistin vapauttamisesta. Toinen .NET-kehyksen pääkomponentti on sen tarjoama luokkakirjasto (engl. Class Library), joka tarjoaa joukon yleisiä, uudelleen käytettäviä tyyppejä, jotka mahdollistavat kehittäjän keskittymisen itse business-logiikan toteuttamiseen. Luokkakirjasto tarjoaa myös erilaisia rajapintoja, jotka mahdollistavat omien toteutuksien saumattoman käytön muiden luokkakirjaston tyyppien kanssa.

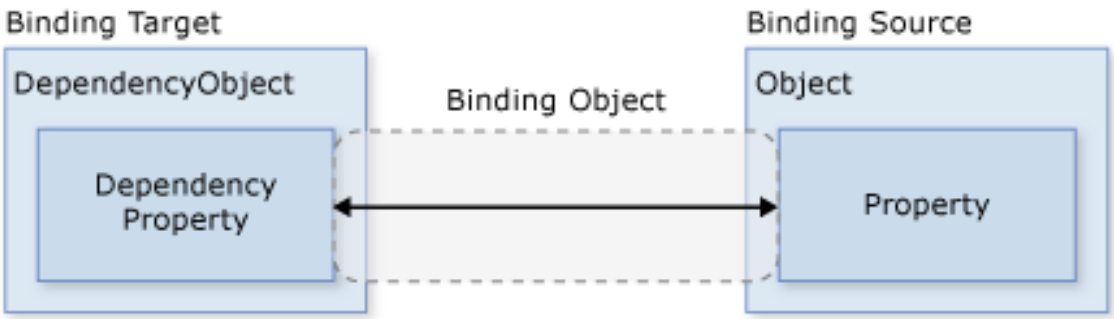
.NET Framework -sovelluskehys tukee monia ohjelmointikieliä, kuten Visual Basicia, C#, F# sekä C++ [1]. Näistä C# on valittu kameratyönkalun kehityskieleksi. C# on vahvasti tyypitetty, oliopohjainen kieli, jota voidaan ajaa .NET Framework -alustan päällä [2]. Oliopohjaisena kielenä C# tukee muun muassa kapselointia, periytymistä ja polymorfismia. Polyformismilla tarkoitetaan saman funktion sitomista eri toteutuksiin riippuen tilanteesta.

Itse käyttöliittymän näkymien toteutukseen on valittu Windows Presentation Foundation -käyttöliittymäkirjasto eli WPF. WPF on osa .NET Framework -kirjastoa, joten se on varsin luonteva valinta myös käyttöliittymän kehitykseen. Se on resoluutioriippumaton sekä sen renderöintimoottori hyödyntää vahvasti nykyaikaista grafiikkalaitteistoa[3]. WPF:n tarkoituksena on korvata aiemmin käytetty Windows Forms -käyttöliittymäkirjasto.

WPF määrittelee oman XML-pohjaisen kielensä eri näkymien kuvaamiseen. Extensible Application Markup Language eli XAML on deklaratiivinen kieli, jolla kuvataan näkymässä näytettävät elementit ja niiden hierarkia, joka luo kokonaisen elementtipuun. WPF:ssä on paljon valmiiksi määriteltyjä elementtejä, esimerkiksi edellä mainittu videotoistin on WPF:ssä määritelty MediaElement-luokka, jolle syötetään videolähteeksi ladattu videotallenne. Näin ollen kehittäjän ei tarvitse kirjoittaa mitään erityistä logiikkaa videon esittämistä varten. WPF sisältää myös paljon muita hyödyllisiä ominaisuuksia, kuten elementtien kustomoitujen tyylien ja uudelleenkäytettävien näkymäpohjien määrittelyn, mutta ehkä keskeisin ominaisuus on yksinkertainen datan sitominen (engl. data binding) malleista itse näkymään.

WPF tarjoaa yksinkertaisen ja yhtenäisen tavan esittää sovelluksessa käsiteltävää dataa, jota kutsutaan datan sitomiseksi[4]. Datan sitominen on prosessi, jossa sovelluksen business-logiikka sidotaan yhteen sovelluksen näkymien kanssa. Siinä datan lähde, kuten esimerkiksi jokin business-logiikan mallin ominaisuus, sidotaan johonkin näkymän ominaisuuteen. Malli on siis datan sitomisen lähdeobjekti, jonka ominaisuus sidotaan näkymän

eli kohdeobjektin ominaisuuteen. Kun tämä sidos on saatu luotua, lähdeobjektin ominaisuuden muuttuessa myös sidotun kohdeobjektin ominaisuus muuttuu automaattisesti eikä muuta logiikkaa tarvita. Datan sitominen on keskeinen ominaisuus VMX-kameran suoratoistetun videon esityksessä kameratyökalun näkymissä.



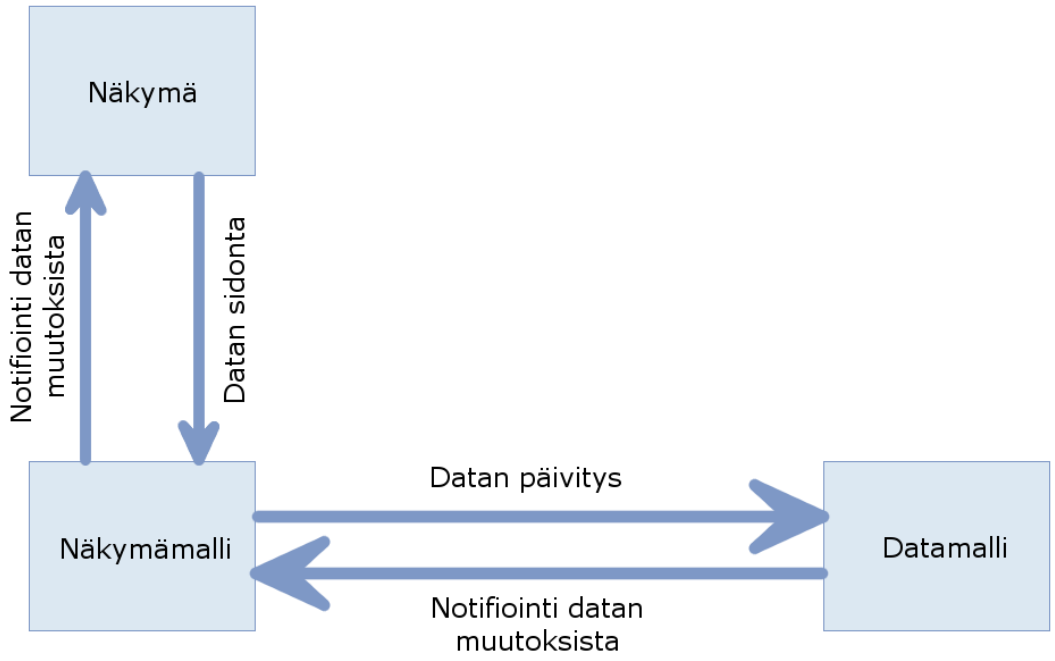
Kuva 2. *WPF-kirjaston toteuttama datan sitominen [4].*

Datan sitominen mahdollistaa myös luontevan malli-näkymä-näkymämalli-arkkitehtuurimallin käytön XAMLia käyttävää sovellusta kehitettäessä. Malli-näkymä-näkymämalli-arkkitehtuurimallissa eli MVVM-mallissa on kolme pääkomponenttia: malli, näkymä ja näkymän malli[5].

Näkymän tehtävänä on määritellä käyttöliittymän ulkoasu eli käyttäjälle näkyvä sovelluksen osa. Tämä osa toteutetaan XAMLia käyttäen. Näkymä kommunikoi ainoastaan oman näkymämallinsa kanssa datan sidontaa hyödyntäen. Näkymän ominaisuuksien arvot sidotaan näkymämallin ominaisuuksiin, jolloin datan muutokset heijastuvat automaattisesti näkymään, kun näkymän malli muuttuu. Datansidonnassa on myös mahdollista päivittää näkymämallia näkymästä päin. Esimerkiksi käyttäjä voi syöttää tekstikenttään merkkijonon, joka datasidonnan ansiosta kirjoittuu suoraan näkymämallin ominaisuuteen.

Näkymämalli toimii välikappaleena varsinaisen mallin ja näkymän välissä. Sen tehtävänä on tarjota mallin sisältämä data itse näkymälle. Näkymämalli lukee datan mallista ja tekee datalle tarvittavat konversiot, jos näkymä ei kykene sitä suoraan sellaisenaan esittämään. Näkymämalli myös määrittelee näkymän vaatimat logiikat, koska XAML on kuvauskieli ja määrittelee vain käyttöliittymän ulkoasun, ei sovelluslogiikkaa. Samaan tapaan kuin näkymä päivittää näkymämalliaan, näkymämalli päivittää itse mallia. Näkymä tuntee siis vain oman näkymämallinsa, ei varsinaista mallia. Näkymämalli taas tuntee vain mallin eikä sillä ole tietoa itse näkymästä.

Mallilla tarkoitetaan datamallia, joka sisältää sovelluskohtaisen datan sekä itse sovelluksen business-logiikan. Tällainen datamalli voi olla esimerkiksi DTO (engl. data transfer object) tai vaikkapa tietovarastoluokka, joka kommunikoi tietokantakerroksen kanssa.



Kuva 3. *MVVM-arkkitehtuurimallin kaavio*

MVVM-arkkitehtuurimallin etuna on se, että se erottaa käyttöliittymän toteutuksen itse sovelluksen business-logiikasta. Tämä mahdollistaa näkymien ja mallin kehittämisen sekä testaamisen erillään toisistaan. Käyttöliittymäsuunnittelija voi siis itsenäisesti työstää näkymää ilman sovelluskehittäjää. MVVM-arkkitehtuurimallissa komponentit ovat löysästi sidoksissa toisiinsa, joten niiden vaihtaminen on helppoa ja niiden sisäistä toteutusta voidaan muuttaa ilman, että muutoksilla olisi vaikutusta muiden komponenttien toimintaan. Tällaisen arkkitehtuurimallin hyödyt nousevat esiin erityisesti suurempien T-LOIKin kaltaisten sovellusten kehityksessä.

2.3 Kamerapalvelin

Kamerapalvelin on yksi T-LOIK-järjestelmän palvelinpään komponenteista, joka pääasiallisesti tarjoaa erilaisia palveluita kameratyökalun käyttöön. Kamerapalvelin muun muassa tarjoaa kameratyökalulle tiedot avattavista kameroista, kuten esimerkiksi kameran sijainnin ja kuvasuunnat. Kun tieliikennepäivystäjä tekee toimintoja kameratyökalun välityksellä, kameratyökalu lähettää tästä tarvittavat pyynnöt kamerapalvelimelle. Esimerkiksi tieliikennepäivystäjän avatessa kameran kameratyökalusta kameratyökalu tiedottaa kamerapalvelimelle, mikä kamera avattiin, jolloin kamerapalvelin osaa tehdä tarvittavat toimenpiteet, kuten esimerkiksi hakea avatun kameran viimeisimmät kuvat, mikäli kyseessä on kuva-aikasarjan avaus. Seuraavat aliluvut pyrkivät tarjoamaan hieman tarkemman kuvan kamerapalvelimen ominaisuuksista ennen VMX-kameraintegraation implementointia, jotta voidaan erottaa, mitä uusia ominaisuuksia integraatio loppuen lopuksi kamerapalvelimelle tuo.

2.3.1 Kamerapalvelimen nykyiset ominaisuudet

Kamerapalvelin tarjoaa rajapintojen kautta erilaisia palveluja kameratyökalulle. Kuten edellisessä aliluvussa mainittiin, ensinnäkin kamerapalvelin tarjoaa tiedot kaikista tietokannasta löytyvistä kameroista. Kameroiden tiedot eivät kuitenkaan ole tallennettuna kamerapalvelimen omaan tietokantaan, vaan tarvittavat tiedot koostetaan muiden T-LOIK-järjestelmän tietokannoista löytyvistä tiedoista. Kamerat ovat käytännössä vain laitteita muiden T-LOIKin kanssa kommunikoivien laitteiden, kuten sääantureiden ja valokylttien joukossa, joten laitetiedot haetaan T-LOIKin ohjauskomponentin tietokannasta. Kameroihin liittyy käsite *havaintoasema.* Havaintoasema on asema, johon laitteita asennetaan, tässä tapauksessa siis kameroita. Havaintoaseman sijainti on siis yhtä kuin kameran sijainti. Havaintoaseman sijaintitiedot taas ovat toisessa tietokannassa. Kamerapalvelin kerää kaikki nämä tiedot yhteen ja lähettää ne kameratyökalulle, joka sitten esittää tiedot kameralistassa. Tämä tietojenhakuominaisuus tulee VMX-kameraintegraation mukana muuttumaan hieman monimutkaisemmaksi, sillä VMX-kamerajärjestelmän tarjoamia tarkempia tietoja ei ole tallennettuna tietokantaan, vaan ne haetaan ulkoisen palvelun tarjoaman REST-rajapinnan kautta.

Kameratyökalu ja -palvelin muodostavat yhdessä tietokannan kanssa kolmikerrosmallin. Kolmikerrosmalliin kuuluu käyttöliittymäkerros (kameratyökalu), välikerros (kamerapalvelin) sekä tietokantakerros. Kolmikerrosmallin mukaisesti kamerapalvelin hoitaa kommunikoinnin käyttöliittymä- ja tietokantakerroksen välillä. Kamerapalvelimella on oma tietokantansa, johon tallennetaan muun muassa tieliikennepäivystäjien tekemiä esivalintoja. Esivalinnaksi kutsutaan kameratyökalun välilehtinäkymää, johon on valittuna tiettyjä kameroita. Tämä tieliikennepäivystäjän valitsema näkymä muutetaan tietokantaan tallennettavaan muotoon. Kun tieliikennepäivystäjä avaa kameratyökalun, hänen omat esivalintansa sekä yleiset esivalinnat listataan kameratyökalussa hänen käyttöönsä.

Kamerapalvelin pitää yllä myös tilatietoa siitä, mitä kameroita on valittuna kameratyökalussa. Kamerapalvelin tallentaa avattujen kameroiden tunnisteet, jotta se osaa seurata palvelimen ulkopuolelta tulevia kameroihin liittyviä päivityksiä ja ilmoittaa niistä kameratyökalulle. Kun kamera avataan kameratyökalusta käsin, kameran tunniste lisätään seurattavien kameroiden listaan. Kun kamerapalvelin vastaanottaa uutta tietoa seurattavaan kameraan liittyen, se ilmoittaa siitä kameratyökalulle. Tämän jälkeen kameratyökalu osaa pyytää uudet kuvat ja päivittää ne kameranäkymään. Tätä kameran seurantaominaisuutta kutsutaan kameran tilaamiseksi. VMX-kameraintegraatio tuo kamerapalvelimelle mukanaan uusia kameratilauksia eri kameroiden ominaisuuksiin liittyen, kuten esimerkiksi tarkemman suunnan seurannan. Tilaukset ovat ajastettuja eli ne päättyvät tietyn ajan kuluessa, ellei niitä uusita. Tällöin kameran tunniste poistetaan tilauslistasta ja uuden päivityksen tullessa kamerapalvelin jättää päivityksen huomioimatta.

Ennen VMX-kameraintegraatiota kamerapalvelimen ehkäpä olennaisin tehtävä on ollut kamerakuvien päivittäminen kameratyökaluun. Kamerapalvelin kuuntelee päivityksiä

uusista kamerakuvista ja suodattaa olennaiset kuvapäivitykset kameratyökalussa auki olevien kamerakuvien mukaan. Kuvien päivityksiä vastaanotetaan kahta eri väylää pitkin riippuen siitä, mikä järjestelmä päivityksen lähettää. LOTJU-järjestelmä tallentaa uudet kuvankaappaukset johonkin ennalta määrättyyn tiedostopolkuun ja ilmoittaa eteenpäin uusista otetuista kuvista. Kamerapalvelin saa tiedon uusista kuvista Java Message Service eli JMS-viestiväylän kautta. Päivityksen vastaanotettuaan kamerapalvelin osaa hakea tallennetun kuvan määritellystä tiedostopolusta. JMS-rajapintaa avataan tarkemmin aliluvussa 2.3.2.

LOTJU-järjestelmän lisäksi kamerapalvelimeen on integroitu Flux-kamerajärjestelmä[6], joka toimii osana häiriöhavaintojärjestelmää (HHJ). Flux-järjestelmästä vastaanotetaan häiriöhavaintojärjestelmän kameroiden kuvia. Toisin kuin LOTJU-järjestelmäintegraatiossa Flux-järjestelmä lähettää JPEG-kuvat kokonaisuudessaan suoraan kamerapalvelimelle ja kamerapalvelin välittää kuvat suoraan kameratyökalulle. Flux-integraatiossa kamerapalvelin toimii siis asiakkaana Flux-järjestelmän palvelinta vasten.

2.3.2 Kamerapalvelimen teknologiat

Kamerapalvelin on Java-ohjelmointikielellä toteutettu palvelin, joka hyödyntää Spring Framework -sovelluskehystä. Spring Framework tarjoaa monenlaisia yleiskäyttöisiä moduuleita palvelintoiminnallisuuksien kehittämiseen. Kamerapalvelin hyödyntää Spring Frameworkin moduuleja muun muassa palvelimen omien komponenttien riippuvuuksien injektoimiseen (dependency injection) sekä tietokantatransaktioiden hallintaan. Lisäksi Spring Framework toteuttaa myöhemmin esiteltävän Java Message Service -rajapinnan, joka on yhdessä HTTP-protokollan kanssa kamerapalvelimen pääasiallinen viestintätapa.

Kamerapalvelin noudattaa REST-arkkitehtuurimallia. REST eli Representational state transfer on HTTP-protokollaan perustuva arkkitehtuurimalli, jolla sovellukset voivat kommunikoida keskenään. HTTP-protokolla on tilaton, hypertekstin siirtämiseen käytetty tiedonsiirtoprotolla, jota käytetään verkkosovellusten välisessä viestinnässä. Koska REST-arkkitehtuurimalli pohjautuu HTTP-protokollaan, sillä on myös HTTP-protokollan mukaisia ominaisuuksia, kuten esimerkiksi asiakas-palvelin-malli ja tilattomuus. REST-arkkitehtuurimallia noudattavassa järjestelmässä on siis aina olemassa asiakas ja palvelin, jossa asiakas lähettää pyynnön palvelimelle ja palvelimen vastaa pyyntöön määritellyllä tavalla. Tämä asiakkaan ja palvelimen välinen viestintä on tilatonta, mikä tarkoittaa sitä, että jokainen asiakkaan lähettämä pyyntö on riippumaton muista lähetetyistä pyynnöistä. Pyynnön tulee siis sisältää aina kaikki sen käsittelyyn tarvittava tieto. Pääasiallisesti kamerapalvelin ja kameratyökalun välinen viestintä on toteutettu REST-arkkitehtuurimallin mukaisesti. VMX-kameraintegraatio tuo mukanaan REST-arkkitehtuurimallia käyttävän VMX-palvelimen, jolle kamerapalvelin toimii asiakkaana.

HTTP-protokollan lisäksi kamerapalvelin käyttää Java Message Service -rajapintaa kommunikoinnissaan muiden T-LOIKiin integroitujen järjestelmien kanssa. Java Message

Service on Java-ohjelmointikielellä kuvattu kokoelma rajapintoja, jonka avulla voidaan lähettää ja vastaanottaa viestejä eri järjestelmien välillä[7]. JMS-rajapinnan käytössä etuna on, että lähettäjän ei tarvitse juurikaan tietää vastaanottajasta mitään. Lähettäjä tietää vain JMS-kohteen (engl. JMS destination), jonne viesti lähetetään ilman tietoa siitä, mikä todellinen vastaanottaja on. Näin ollen viestinnän molemmat osapuolet ovat vain löysästi riippuvaisia toisistaan.

JMS tukee kahdentyyppistä viestitysmallia: lähteestä kohteeseen -viestintää ja julkaise-tilaa-viestintää. Lähteestä kohteeseen -viestinnässä (engl. point-to-point) on tasan yksi lähettäjä ja yksi vastaanottaja, joiden välillä on viestijono[7]. Lähettäjä lähettää JMS-viestin viestijonoon, joka ylläpitää saapuvien viestien järjestystä. Vastaanottaja taas lukee JMS-viestit samasta jonosta niiden saapumisjärjestyksessä. Julkaise-tilaa-viestintä (engl. publish-subscribe) eroaa lähteestä kohteeseen -viestinnässä siinä, että samalla JMS-viestillä voi olla useita vastaanottajia. Lisäksi julkaise-tilaa-viestintä käyttää jonojen sijasta aiheita (engl. topic), jonne viestit lähetetään. Viestinlähettäjä siis julkaisee viestin johonkin tiettyyn aiheeseen yleensä viestin sisällön perusteella. Vastaavanlaisesti tästä tietyn aiheen viesteistä kiinnostuneet vastaanottajat tilaavat tämän aiheen.

Kuten mikä tahansa muukin puhdas rajapinta, JMS vaatii toimiakseen rajapintansa kuvaamien palvelujen toteutuksen. Tällaisia JMS-rajapinnan toteuttavia kirjastoja ovat muun muassa Apache ActiveMQ, OpenJMS ja RabbitMQ. JMS-rajapinta määrittelee, millaisia olioita JMS-sovellukset vaativat toimiakseen. Tällaisia olioita ovat esimerkiksi aiemmin mainittu JMS-kohde ja JMS-yhteysolio (engl. JMS connection). Kirjaston on toteutettava kaikki JMS-rajapinnan kuvaamat JMS-oliot, jotta se toteuttaa viestinvälityksen kokonaisuudessaan.

2.4 VMX-kamerajärjestelmä

VMX-kamerajärjestelmä on Liikenneviraston käytössä oleva keskitetty kamerajärjestelmä, johon teiden varteen ja tunneleihin sijoitetut liikenne- ja häiriöhavaintokamerat on kytketty. VMX-kamerajärjestelmän on kehittänyt suomalainen yritys nimeltä Teleste, joka tarjoaa erilaisia video- ja laajakaistateknologioita sekä palveluita.

VMX-kamerajärjestelmä perustuu asiakas-palvelin-malliin, joka tukee monimutkaisempiakin verkkotopologioita. Järjestelmä koostuu sekä kokoelmasta erilaisia sovelluksia, kuten käyttöliittymä- ja palvelinsovelluksista, että kustomoiduista laitteistosta. Se on vahvasti skaalautuva, ja se pystyy käsittelemään jopa tuhansista kameroista koostuvia laiteverkostoja. Näin ollen VMX-kamerajärjestelmä on varsin perusteltu valinta liikennekameroiden hallintaan.

VMX-kamerajärjestelmä koostuu kolmesta eri pääkomponentista, jotka ovat VMX Server, VMX Client ja VMX NVR. VMX Server on järjestelmän palvelinsovellus, kun taas VMX Client on asiakkaana toimiva käyttöliittymäsovellus. VMX Serverin tehtävänä on

käsitellä VMX Clientilta lähetetyt pyynnöt, pitää huolta pääsynhallinnasta sekä monitoroida muita järjestelmän osia. VMX Server on vastuussa myös videoyhteyksistä sekä integraatioista muihin palveluihin.

VMX Client on käyttöliittymäsovellus, jota tieliikennepäivystäjät käyttävät liikennekameroiden videokuvan katseluun sekä muuhun kameranhallintaan liittyviin tehtäviin. VMX Client on vastuussa videokuvan dekoodauksesta ja se tukee muun muassa MPEG-4 Part 2- ja Part 10 -koodattuja videoita. Videokuvan dekoodamisella tarkoitetaan pakatun videodatan purkua esitettävään muotoon. Tähän palataan luvussa 3. Videokuvan katselun lisäksi VMX Clientin toimintoihin kuuluvat kameran kääntäminen, kuvan suurentaminen sekä kameran lukitseminen tiettyyn asentoon. Tämän työn tavoitteena on tuoda ainakin edellä mainitut toiminnot kameratyökaluun tieliikennepäivystäjien käyttöön, jolloin tarve VMX Clientin käyttöön poistuisi ja yksi T-LOIK-projektin tavoitteista eli työkalujen yhtenäistäminen toteutuisi.

VMX NVR eli Network Video Recorder on vastuussa VMX-kamerajärjestelmässä tuotetun videokuvan ja äänen tallentamisesta. Se tarjoaa myös erilaisia toimintoja tallenteiden hakuun ja käsittelyyn. VMX NVR ei ole kovin oleellinen osa VMX-kameraintegraation tavoitteiden kannalta, joten se jätetään vähemmälle huomiolle.

Näiden kolmen pääkomponentin lisäksi voidaan mainita VMX-järjestelmän tarjoama VMX Software Development Kit eli VMX SDK. VMX SDK tarjoaa ohjelmistokehittäjille tarvittavat työkalut oman asiakassovelluksen luontiin, jonka kautta VMX-kamerajärjestelmää voidaan hallita. Käytännössä se on kokoelma C-ohjelmointikielen kirjastoja, joiden avulla voidaan kommunikoida järjestelmän eri komponenttien kanssa. VMX SDK:ta on hyödynnetty VMX-kameraintegraatiossa niin, että VMX-kamerajärjestelmän päälle on kehitetty SDK:ta hyödyntäen REST-palvelin, jonka kautta muut VMX-kameraintegraation osapuolet, pääasiassa kamerapalvelin, lähettävät pyyntöjä itse järjestelmään. Tämä ratkaisu yksinkertaistaa huomattavasti kamerapalvelimelle toteutettavia integraatiotarpeita. Käytännössä kamerapalvelimen ei siis tarvitse tietää mitään itse VMX-kamerajärjestelmästä.

3. VIDEON SUORATOISTO

Perinteinen tapa toistaa videoita tietokoneella on ollut jonkin mediasovelluksen, kuten esimerkiksi VLC-mediasovelluksen käyttäminen. Mediasovellukselle on annettu paikallisesti saman tietokoneen kovalevylle tallennettu videotiedosto, jota mediasovellus on alkanut lukemaan ja esittämään käyttäjälleen. Nykyään tietoverkot ovat vahva osa videoja muiden mediatiedoston esittämistä. Seuraavat aliluvut keskittyvät avaamaan tietoverkkojen roolia mediatiedostojen toistoprosessissa ja avaamaan medioiden suoratoiston käsitettä.

3.1 Videon suoratoiston prosessi

Videon suoratoistossa videodata siirretään verkon yli lähteestä kohteeseen. Lähde voi olla esimerkiksi itse kamera tai vaikkapa palvelimen kovalevylle tallennettu videotiedosto. Kohteeksi kutsutaan suoratoistoprosessin toisessa päässä olevaa videodatan vastaanottajaa, joka yleensä toistaa videodatan suoraan. Suoratoisto (engl. streaming) on siis prosessi, jossa dataa siirretään reaaliaikaisesti niin, että vastaanottaja voi käyttää jatkuvasti dataa haluamallaan tavalla[8]. Tämän työn puitteissa keskitytään pelkästään videodatan prosessointiin, sillä kameratyökalu tulee toistamaan videodataa. Ääntä ei tulla toistamaan, joten audiodatan käsittely pudotetaan pois.

Suoratoiston käyttö videoiden toistamisessa mahdollistaa sen, että koko videota ei tarvitse ladata ennen kuin sen voi toistaa. Esimerkiksi kameratyökalun nykyisessä toimintatavassa, häiriöhavaintovideoita ei toisteta suoraan, vaan ne on ladattava työasemalle ensin kokonaan, jotta niitä voidaan käsitellä. Kun lataus on valmis, vasta silloin voidaan videota katsella. Tällainen ratkaisu tuottaa ongelmia erityisesti silloin, kun videotiedostot ovat suuria, jolloin niiden lataaminen voi kestää kauan, mikä viivästyttää videon katselun aloittamista. Suoratoiston käyttö ratkaisee tämänkaltaiset ongelmat.

Täysin ongelmaton suoratoiston käyttö videoiden katseluun ei kuitenkaan ole. Normaalissa tiedonsiirrossa, kun tiedosto on saatu perille, se ei ole enää riippuvainen verkon tilasta, toisin kuin suoratoisto. Käytännössä suoratoisto on jatkuvaa datapakettien lähetystä lähteestä kohteeseen, jolloin datapakettien mahdollinen hukkuminen tai viivästyminen voi aiheuttaa ongelmia vastaanottajan päässä, kun videota yritetään toistaa. Mikäli datapaketteja hukkuu tarpeeksi siirron aikana, videon toistaminen ei enää onnistu, mikä näkyy videon katsojalle katkoksina tai mahdollisina kuvan vääristyminä. Vastaanottajan video voi pysähdellä myös, jos datapaketit tulevat liian myöhässä.

Datapakettien myöhästymisestä johtuvia ongelmia voidaan yrittää korjata käyttämällä vastaanottajan päässä puskurointia [8]. Puskuroinniksi (engl. buffering) sanotaan tekniik-

kaa, jossa vastaan otettavaa videodataa tallennetaan tietty määrä, ennen kuin mediatoistimen annetaan alkaa toistamaan vastaanotettua dataa. Jos datapaketteja myöhästyy, videon toisto ei katkea heti, vaan mediatoistin lukee dataa puskurista. Tämä antaa lisäaikaa uusien datapakettien saapua kohteeseensa. Mikäli datapaketit myöhästyvät liikaa, puskuri tyhjenee ja videontoisto katkeaa. Tämän jälkeen puskuria yritetään täyttää uudestaan ennen kuin videota aletaan toistaa uudestaan. Mediatoistin yleensä indikoi yleensä tällaisesta tilanteesta katsojalle jonkinlaisena latausanimaatiolla tai -ikonilla. Puskurointi on siis yleensä jätetty datapaketteja vastaanottavan mediatoistimen vastuulle. Työssä käytettävä VLC-mediatoistin hyödyntää myös puskurointia videon suoratoistossa.

3.2 Videodatan pakkaus ja purku

Videon suoratoistossa tärkeitä vaiheita ovat videodatan pakkaaminen ja purkaminen, sillä raaka videodata vie paljon tilaa ja videodatan lähettäminen sellaisenaan verkon yli aiheuttaisi mitä luultavammin jatkuvaa puskurointia ja videon katkeamista riippuen verkon nopeudesta. Pakkaamattoman videodatan lähetys melkein reaaliajassa nykyisen Internetin välityksellä ei ole tarpeeksi nopeaa pienempiresoluutioisten videoidenkaan lähetykseen.

Videon pakkaaminen tehostaa datan lähetystä, jolloin korkearesoluutioisia videoitakin voidaan lähettää tarpeeksi nopeasti, jotta suoratoisto olisi mahdollista. Lisäksi videon pakkaaminen voi ratkaista esimerkiksi käyttöympäristöön liittyviä ongelmia. Käyttöympäristö ei ehkä tue suoraan raakavideon vastaanottoa, esimerkiksi rajoitetun verkon nopeuden tai tallennustilan puutteen takia.

Käytännössä videon pakkaaminen tarkoittaa redundanssin poistoa datasta. Pakkaamiseen käytettävät algoritmit voidaan jakaa karkeasti kahteen eri luokkaan: häviöttömään ja häviölliseen pakkaukseen. Häviötön pakkaus tarkoittaa videon pakkaamista niin, että se voidaan rakentaa alkuperäiseen, pakkaamattomaan muotoonsa vastaanottajan päässä. Käytännössä tämä tarkoittaa tilastollisen redundanssin poistamista eli datassa useasti esiintyvät elementit tiivistetään. Esimerkiksi useasti esiintyvä sama merkkijono voidaan korvata lyhyemmällä muodolla. Häviöttömät pakkausalgoritmit eivät kuitenkaan ole datan pakkauksen suhteen kovin tehokkaita eli datan määrää ei saada pienennettyä kovin suurissa määrin[9]. Käytännöllisempi vaihtoehto on häviöllinen pakkaus, jossa pakkaussuhde on parempi, mutta pakatusta datasta ei saada rakennettua täysin alkuperäistä vastaavaa videota. Pakkaussuhde paranee siis laadun kustannuksella. Häviöllinen pakkaus vie redundanssin poiston häviötöntä pakkausta pidemmälle poistamalla lopputuloksen kannalta merkityksettömiä elementtejä datasta. Videokuvassa voi olla suuria samankaltaisia alueita, kuten seiniä, joiden pienet yksityiskohdat ovat merkityksettömiä katsojan kannalta ja ne voidaan pudottaa pakattavasta datasta pois. Käyttökontekstista riippuen optimaalisimpaan tulokseen päästään siis löytämällä oikea tasapaino pakkaussuhteen ja videon laadun väliltä. Mikäli vastaanottaja ei tarvitse yksityiskohtaista, korkearesoluutioista videokuvaa vaan vähempikin riittää, voidaan valita pakkausalgoritmi, joka on pakkaustehokkuudeltaan korkea, jolloin bittinopeus voidaan valita hyvinkin matalaksi.

Kuva 4: *Kooderin ja dekooderin sijainti videodatan siirrossa*

Videodatan pakkaamisesta ja purkamisesta käytetään usein nimitys videokoodaus. Videokoodauksessa videota pakkaavaa osaa eli lähteen päässä käytettävää osaa kutsutaan kooderiksi (engl. encoder). Vastaavanlaisesti kohteen tai vastaanottajan päässä, videopakkauksen purkavasta osasta käytetään nimeä dekooderi (engl. decoder). Näiden kahden komponentin luomaa paria kutsutaan koodekiksi (engl.codec)[9]. Kuvan 4 mukasesti kooderi ottaa syötteekseen raa'an videodatan ja pakkaa sen ennen sen välitystä verkon yli. Dekooderi puolestaan ottaa syötteekseen pakatun videodatan ja yrittää luoda datasta mahdollisimman hyvän approksimaation alkuperäisestä kuvasta. Jos pakkaus on häviötön, dekooderi kykenee luomaan datasta täydellisen kopion alkuperäisestä kuvasta.

Videokoodekkikirjastot noudattavat yleensä jonkinlaista kansainvälistä standardia. Yksi tällainen standardi on MPEG-4. MPEG-4 on Moving Picture Experts Group-työryhmän eli MPEGin kehittämä videon ja äänen pakkausstandardi[9]. MPEG on kansainvälisen standardoimisjärjestö ISOn ja sähköalan standardoimisjärjestön muodostama työryhmä, jonka tarkoituksena on kehittää uusia videonpakkaustapoja ja standardoida ne. MPEG on kehittänyt useita eri standardeja, kuten esimerkiksi MPEG-2-standardit ja MPEG-4-standardit. Edellä mainitut standardit voidaan jakaa edelleen vielä pienempiin osiin kehitysvaiheiden mukaan. Esimerkiksi MPEG-4-standandilla on tämän työn kirjoitushetkellä yli 30 eri osaa, joista VMX-kameraintegraation kannalta oleellisimmat ovat MPEG-4 Visual eli Part 2 ja MPEG-4 Advanced Video Coding eli Part 10. MPEG-4 AVC tunnetaan myös nimellä H.264. VMX-kamerajärjestelmä käyttää kamerakuvan pakkaamiseen MPEG-4 Visual- ja MPEG-4 AVC-standardit toteuttavia koodereita. Tämän lisäksi pieni osa kamerakuvista pakataan vanhemmalla MPEG-2-standardin mukaisella kooderilla, mutta tämä standardi jätetään tarkastelun ulkopuolelle, sillä kaikki kamerat ollaan tuomassa loppuen lopuksi MPEG-4-standardin piiriin. Seuraavat aliluvut kuvaavat hieman tarkemmin MPEG-4 Visual- ja MPEG-4 AVC -standardien ominaisuuksia.

3.2.1 MPEG-4 Visual

MPEG-4 Visual on MPEG-4:n toinen osa, joka määrittelee, miten visuaalinen informaatio eli käytännössä video tulisi esittää videokoodauksessa. Kyseinen standardi kuvaa joukon erilaisia koodaustapoja, joiden avulla yksittäiset kuvat (engl. frame) videon sisällä tai tietyt alueet kuvassa voidaan esittää pakatussa muodossa. MPEG-4 Visual -standardin koodaustyökaluja voidaan käyttää monissa eri sovelluksissa, kuten vanhemmissa digitaalisissa TV-lähetyksissä, tietokonegrafiikassa tai vaikkapa videoiden suoratoistossa verkon yli.

MPEG-4 Visual pääasiassa määrittelee koodauksessa käytettävän syntaksin eli tavan, miten pakattu video tulisi esittää[9]. Standardi ei siis ota kantaa siihen, miten kooderin tulisi toimia, ainoastaan koodauksen lopputulos eli pakattu video tulee olla määrätyssä muodossa. Se siis määrittelee datan muodon, jossa se tulee olla kooderin ulostulossa (kuva 4 sivulla 15). Lisäksi standardi määrittelee tavan, jonka avulla dekooderin tulisi rakentaa alkuperäisen informaation approksimaatio pakatusta informaatiosta. Tällainen lähestymistapa antaa koodekkien kehittäjille vapauden tuottaa varsinaisen koodekin toteutuksen haluamallaan tavalla.

Verrattuna myöhemmin esiteltävään MPEG-4 AVC- eli H.264-standardiin, MPEG-4 Visual keskittyy enemmän joustavuuteen, siinä missä H.264 on fokusoitunut tehokkuuteen ja luotettavuuteen[9]. Juurikin tämän joustavuuden ansiosta MPEG-4 Visualia voidaan hyödyntää niin monella eri sovellusalueella. Sen koodaustyökalut onkin jaettu eri kategorioihin sen mukaan, mihin sovellusalueeseen niitä on suositeltu käytettävän. Näitä kategorioita kutsutaan profiileiksi. MPEG-4 Visual sisältää muun muassa seuraavanlaisia profiilityyppejä:

- yksinkertaiset profiilit
- objektipohjaiset profiilit
- liikkumattomat tekstuurit
- skaalautuvat profiilit.

Näistä profiileista VMX-kameraintegraatiossa MPEG4 Visual-koodausta käyttävät suoratoistot käyttävät yksinkertaisia profiileja. Tästä koodaustyypistä käytetään lyhennettä MPEG-4 SP (engl. simple profile). Yksinkertaista profiilia käytetään videon suorakaiteen muotoisten yksittäisten videokuvien koodaamiseen. Objektipohjaista profiilia käytetään mielivaltaisten muotojen koodaamiseen kuvan sisällä ja tekstuuriprofiilia still-kuvien tai tekstuurien koodaamiseen. Skaalautuvalla profiililla taas käsitellään erilaisia resoluutioita sekä laatuasteita. Edellä mainittujen profiilien lisäksi MPEG-4 sisältää lukuisia muitakin profiileja eri käyttötarkoituksiin.

Erilaisten profiilien tarkoituksena on rajata implementoitavien toiminnallisuuksien joukkoa erilaisten sovellusten tarpeiden mukaisesti. MPEG-4 itsessään ei tarjoa mitään toteutusta profiilien kuvaamille työkaluille, vaan se jää standardin toteuttavan koodekin vastuulle. Koodekin ei kuitenkaan tarvitse toteuttaa kaikkia MPEG-4 kuvaamia toiminnallisuuksia. Koodekin suunnittelija voi rajata vaadittavia toiminnallisuuksia valitsemalla profiilin, joka tarjoaa ne työkalut, joita kohdesovellus tarvitsee. Koodekin laajuutta ja kompleksisuutta voidaan siis muokata sen mukaan, mitä käyttökonteksti oikeasti tarvitsee. Yksinkertainen profiili on suunniteltu nimenomaan yksinkertaisille sovelluksille, jotka siirtävät videota Internetiin tai tallentavat sitä esimerkiksi muistisirulle[10]. Yksinkertaisen profiilin käyttö kameralaitteiden, kuten tässä tapauksessa VMX-kamerajärjestelmän kameroiden kanssa, on varsin perusteltua.

3.2.2 MPEG-4 Advanced Video Coding

MPEG-4 Advanced Video Coding (AVC) eli H.264 on toinen VMX-kamerajärjestelmän käyttämä videopakkausstandardi. MPEG-4 AVC -standardin mukaisten koodekkien käyttö on varsin suosittua erityisesti videoiden toistossa Internetin välityksellä. Esimerkiksi kaikki Youtube-sivuston kautta katseltavat videot ovat tarjolla myös H.264-koodauksella[11]. Lisäksi MPEG-4 AVC on nykyään yksi pakollisista Blu-ray-levyillä käytettävistä videopakkausformaateista.

MPEG-4 AVC-videopakkausstandardin ensimmäinen versio julkaistiin vuonna 2003 ja sen kehitti MPEG yhdessä VCEG -työryhmän (Video Coding Experts Group) kanssa. VCEG on osa kansainvälistä International Telecommunication Union-telekommunikaatioyhdistystä, jonka tehtävänä on kehittää standardeja telekommunikaation osa-alueella. MPEG-4 AVC-standardin kehitys jatkuu edelleen ja siitä on julkaistu sittemmin lukuisia uusia versioita.

MPEG-4 AVCn pääasiallisena tavoitteena on tarjota parempaa videokuvan laatua pienemmällä bittinopeudella. Bittinopeus kuvaa suoratoiston tiedonsiirtonopeutta. Datansiirtoon palataan aliluvussa 3.3. Pakkaamisen ja datansiirron tehokkuus sekä luotettavuus ovat avainasemassa tässä standardissa, mutta sitä ei voida käyttää yhtä monella sovellusalueellaa kuin aiemmin mainittua MPEG-4 Part 2:ta. Verrattuna MPEG-4 Part 2:een, MPEG-4 AVC:ssa on saatavilla vain kolme eri profiilia ja se keskittyy vain yleisimpien sovellusten tarpeisiin siinä, missä MPEG-4 Part 2 taas pyrkii olemaan mahdollisimman joustava ja yleiskäyttöinen[9].

Arkkitehtuuriltaan MPEG-4 AVC on suurimmaksi osaksi samanlainen kuin MPEG-4 Part 2. Se sisältää samat videokoodauksen perusosat, mutta osien tekniset toteutukset vaihtelevat hieman, eikä niiden avaaminen ole tässä kohtaa järkevää. MPEG-4 AVC sisältää kuitenkin lisäkomponentin, jota kutsutaan lohkonsuodatukseksi (engl. deblocking filter)[9]. Videokoodauksessa videokuva jaetaan lohkoihin, jonka koko voi vaihdella 4x4-kokoisesta pikselialueesta aina 16x16-kokoiseen alueeseen. Lohkoista kootaan alkuperäinen kuva dekoodausvaiheessa ja tässä prosessissa lohkojen välille voi syntyä näkyviä reunoja. Lohkonsuodattimen tehtävänä on pehmentää näitä reunoja, jotta koottu videokuva näyttäisi sulavammalta.

MPEG-4 AVC tuo myös rajoitteen koodattavien videokuvien muodon suhteen. Siinä missä MPEG-4 Part 2 kykenee käsittelemään mielivaltaisia kuva-alueita, MPEG-4 AVC pystyy käsittelemään vain suorakulmaisen muotoisia kuvia. Tämä ei kuulosta kovin suurelta rajoitteelta, mutta käytännössä se karsii kuitenkin sovelluksien joukkoa, johon kyseistä standardia voidaan hyödyntää.

Toinen rajoittava tekijä on profiilien määrä. Edellä kuvattu MPEG-4 Part 2 sisältää kymmeniä erilaisia profiileja, joiden avulla koodekkeja voidaan räätälöidä erilaisiin tarkoituksiin. MPEG-4 AVC sisältää vain kolme erilaista profiilia, jotka ovat Baseline-, Main- ja Extended-profiilit. Tarkempiin teknisiin ominaisuuksiin pureutumatta Baseline-profiili on potentiaalinen vaihtoehto, kun kyseessä on esimerkiksi videopuhelu- tai videokonferenssisovellus. Main-profiilin toiminnallisuudet taas tukevat paremmin muun muassa televisiolähetyksiä ja videoiden tallennusta. Viimeinen profiili eli Extended-proflii tuo prosessiin lisää virhesietoisuutta ja on näin ollen hyvä suoratoistoa hyödyntävissä sovelluksissa.

MPEG-4 AVC -standardista on jo julkaistu kehittyneempi versio vuonna 2013 MPEG-H Part 2. Se tunnetaan myös nimillä High Efficiency Video Coding (HEVC) ja H.265. Se on pakkaussuhteeltaan 50 prosenttia parempi kuin MPEG-4 AVC[12] eli se kykenee koodaamaan videodataa saman verran kuin MPEG-4 AVC, mutta huomattavasti pienemmällä bittinopeudella. Toisin sanoen MPEG-H Part 2 kykenee samalla bittinopeudella lähettämään paljon laadukkaampaa kuvaa. Kääntöpuolena kuitenkin on, että se vaatii huomattavasti enemmän laskentatehoa kuin MPEG-4 AVC.

3.2.3 Motion JPEG

Kolmas VMX-kamerajärjestelmässä videon pakkaamiseen käytettävä formaatti on Motion JPEG eli MJPEG. MJPEG:ssä jokainen videossa esiintyvä yksittäinen kuva pakataan yksitellen JPEG-formaattiin[13]. JPEG on yleisesti digitaalisissa kuvissa käytetty häviöllinen pakkausformaatti. MJPEG-formaattia käytetään muun muassa web- ja IP-kameroiden videokuvan siirrossa.

MJPEG eroaa MPEG-standardin mukaisista toteutuksista siinä, että se käsittelee kuvia yksitellen siinä, missä MPEG taas hyödyntää pakattavan kuvan edellisiä ja seuraavia kuvia. MJPEG-formaattia käytettäessä kaikki videon sisältämät kuvat käsitellään siis erikseen ilman, että kuvilla olisi riippuvuutta toisiinsa pakkaamisen aikana. Tämä tapa on erityisen hyödyllinen siinä, että videon laatu ei kärsi, vaikka peräkkäisten kuvien data vaihtelisi suuresti. Esimerkiksi videossa esiintyvät nopeat liikkeet voivat MPEG-tyyppisessä koodauksessa heikentää videon laatua tilapäisesti, sillä MPEG-koodauksessa kuvia käsitellään ryhmittäin temporaalisen redundanssin poistamiseksi. MJPEG:ssä videon laatu pysyy samana vaihteluista huolimatta. Koska kuvia käsitellään yksitellen, koodaus tarvitsee myös vähemmän laskentatehoa, sillä käsiteltävää dataa on vähemmän verrattuna kuvajoukon käsittelyyn.

MJPEGin heikkous on kuitenkin sen vaatima datan määrä. MJPEG ei poista temporaalista redundanssia eli se ei ota huomioon peräkkäisten kuvien välistä korrelaatiota johtuen siitä, että kuvat käsitellään aina yksitellen. Näin ollen pakkaussuhde on huomattavasti heikompi kuin MPEG-koodekkeja käytettäessä. Tästä johtuen siirrettävää dataa on enemmän, eikä yhtä pieniin bittinopeuksiin päästä kuin MPEG-koodekkien tuottamassa datan

siirrossa[14]. Lisäksi MJPEG käyttää tallennuksessa enemmän levytilaa, koska jokainen kuva tallennetaan erillisenä JPEG-kuvana. MJPEG onkin vahvimmillaan, kun videokuvat sisältävät paljon vaihtelua suhteessa toisiinsa.

3.3 Videodatan siirto

Pakatun videodatan lähetyksen yhteydessä puhutaan usein bittinopeudesta (engl. bitrate). Bittinopeus kuvaa sitä, kuinka paljon pakattua videodataa lähetetään aikayksikköä (yleensä per sekunti) kohden[8]. Yhdessä koodekin kanssa bittinopeudella on suuri vaikutus lopulliseen vastaanottajan päässä näytettävään videokuvaan. Mitä suurempi bittinopeus, sitä enemmän pakattua videodataa pääsee läpi lähteestä kohteeseen, jolloin laadullisesti parempaa kuvaa voidaan lähettää ja suoratoisto ei katkea puskuroinnin takia. Yksinkertaisimmillaan bittinopeus voi olla asetettu vakioksi, mutta kehittyneemmissä ratkaisuissa bittinopeus voi vaihdella riippuen esimerkiksi verkon nopeudesta. Mikäli verkon nopeus muuttuu, bittinopeus sopeutuu tähän verkon muutokseen joko laskemalla tai nostamalla arvoaan. Tällainen dynaaminen ratkaisu on varsin käyttökelpoinen ratkaisu muun muassa mobiililaitteiden kanssa, sillä verkon nopeus voi vaihdella hyvinkin radikaalisti.

Suoratoisto-protokollat

Suoratoistossa pakattu videodata lähetetään verkon yli lähettäjältä vastaanottajalle, esimerkiksi suoratoistopalvelimelta asiakkaalle. Datan siirtämiseen on olemassa monenlaisia protokollia, joista VMX-kameraintegraation kannalta oleellisimpia ovat HTTP sekä RTP.

HTTP eli Hypertext Transfer Protocol on hypertekstin siirtoon tarkoitettu sovellustason protokolla[15]. HTTP-protokollaa käytetään asiakas-palvelin-mallisissa sovelluksissa, esimerkiksi WWW-palvelimien ja verkkoselaimen välisessä viestinnässä. Asiakas lähettää jonkin HTTP-protokollan määrittelemän tyypin mukaisen pyynnön, johon palvelin vastaa. Verkkoselain voi esimerkiksi lähettää GET-tyyppisen pyynnön jonkin resurssin hakua varten ja palvelin vastaa tähän pyyntöön palauttamalla kyseisen resurssin. HTTP-protokolla on tilaton, joten jokainen asiakkaan lähettämä pyyntö on riippumaton aiemmista tai tulevista pyynnöistä. Tämä tarkoittaa myös sitä, että jokaisessa HTTP-pyynnössä on oltava kaikki palvelimen tarvitsema data, jotta pyyntö voidaan käsitellä.

HTTP-protokollaa käytetään VMX-kameraintegraatiossa MJPEG-suoratoiston lähettämiseen. Jokainen MJPEG-pakkauksen lopputuotteena syntynyt JPEG-kuva lähetetään HTTP-protokollaa käyttäen verkon yli kameratyökalun VLC-mediatoistimelle. VLC-mediatoistin toimii tässä yhteydessä siis asiakkaana, jolla on tiedossa VMX-kamerapalvelimella sijaitsevan resurssin osoite. VLC-asiakas lähettää osoitteeseen HTTP-protokollan mukaisen GET-pyynnön VMX-kamerapalvelimelle, joka ohjaa taas ohjaa pyynnön varsinaiselle suoratoistopalvelimelle, josta kuvaa saadaan.

VLC-asiakas saa vastaukseksi sarjan HTTP-vastauksia, jotka sisältävät videokuvia JPEG-pakattuina. Asiakas saa siis yhdelle pyynnölle useita vastauksia. Kun asiakas lähettää pyynnön HTTP-protokollaa käyttäen, asiakkaan ja palvelimen välille avataan TCP-yhteys[15], jonka kautta pyyntö välitetään palvelimelle ja asiakas saa vastauksensa. Yleensä yhdelle HTTP-pyynnölle odotetaan yksittäistä vastausta, jonka jälkeen TCP-yhteys suljetaan. MJPEG-suoratoistossa palvelin kertoo HTTP-vastauksensa metatiedoissa, että vastaus on moniosainen, jolloin asiakas tietää odottaa useita vastauksia, eikä näin ollen TCP-yhteyttä suljeta ensimmäisen vastauksen jälkeen. Palvelin lisää vastauksen metatietoihin HTTP-protokollan määrittelemän Content Type -otsikon, jonka päätyyppi on asetettu arvoon *multipart*. Kyseinen arvo kertoo asiakkaalle, että vastaus on moniosainen, jolloin asiakas jää vastaanottamaan JPEG-kuvia, kunnes palvelimelta ei enää kuvia tule tai jos asiakas eksplisiittisesti sulkee yhteyden.

MPEG-4 Part 2- ja Part 10 -koodatun videodatan siirtoon käytetään VMX-kameraintegraatiossa RTP-protokollaa. RTP eli Real-Time Transport Protocol on erityisesti median suoratoistoon suuntautunut protokolla siinä, missä HTTP on melko yleiskäyttöinen. Kuljetustason erona HTTP-protokollaan on se, että HTTP:n hyödyntäessä varmempaa TCP-protokollaa, joka kuittaa pakettien vastaanoton, RTP-protokolla normaalisti hyödyntää UDP-protokollaa, joka on latenssiltaan pienempi kuin TCP[9]. UDP ei tarjoa TCP:n kaltaisia toipumis- tai synkronointimekanismeja datapaketin hukkuessa. RTP-protokolla ei kuitenkaan ole toteutukseltaan sidottu tähän protokollaan, mutta on erittäin yleisesti käytetty UDP-protokollan päällä[16]. Lisäksi oleellinen ero HTTP-protokollaan on se, että RTP on OSI-mallin mukaisen kuljetustason protokolla, joka toimii toisen kuljetustason protokollan, kuten esimerkiksi UDP:n päällä HTTP:n ollessa sovellustason protokolla. RTP tarjoaa itsessään vain toiminnot datan siirtoon eikä takaa siirron laatua[8]. Tästä syystä RTP:n yhteydessä käytetään yleensä toista protokollaa nimeltään Real-Time Control Protocol eli RTCP.

RTCP:tä käytetään RTP-protokollan kanssa tarkkailemaan RTP-protokollan avulla siirretyn datan siirron laatua. RTCP-protokollan päätarkoituksena on tarjota tietoa suoratoiston laadusta suoratoiston eri osapuolille[8]. Tällaista istuntokohtaista tietoa voidaan käyttää hyväksi esimerkiksi dynaamisten bittinopeuksien säätämisessä. Myös kehittyneemmät koodekit ja suoratoistopalvelimet voivat sopeutua suoratoistosession muutoksiin RTCP-protokollan kautta välitetyn laadullisen informaation avulla. RTCP ei siis kuljeta varsinaista videodataa, vaan ainoastaan tietoja suoratoistosession laadusta, kuten esimerkiksi hävinneiden datapakettien määrästä tai pakettien kuljetuksen viiveestä.

RTP- ja RTCP-protokollien yhteydessä mainittu suoratoistosessio kuvataan tekstipohjaisen Session Description Protocol-protokollan (SDP) avulla. SDP:n avulla voidaan kuvata erilaisia multimedia-istuntojen sisältämiä ominaisuuksia[17], kuten esimerkiksi median tyyppi ja osoite, josta suoratoisto on saatavilla. VMX-kameraintegraatiossa RTP-protokollan kautta lähetettävät videoiden suoratoistot kuvataan SDP-protokollaa noudattavien merkkijonojen avulla. Nämä SDP-merkkijonot sisältävät tarvittavat tiedot uuden istunnon

aloittamiseen kameratyökalun eli asiakkaan sekä palvelimen eli VMX-kamerapalvelimen välille. SDP-merkkijonojen avulla VLC-mediatoistin saa tietää, mistä osoitteesta ja portista varsinainen videodata tulee sekä millaista koodekkia datan pakkaamiseen on käytetty. SDP sisältää siis kaiken sen informaation, joka tarvitaan videon suoratoiston katselun aloittamiseksi.

4. VMX-KAMERAINTEGRAATION VAATIMUKSET JA SUUNNITTELU

VMX-kamerajärjestelmän integraation päätavoitteena on tuoda VMX Client -sovelluksen oleellisimmat toiminnallisuudet kameratyökalun puolelle, jolloin tieliikennepäivystäjien tarve VMX Client -sovelluksen käytölle vähenisi. Tämä tavoite mukailee T-LOIK-projektin tarkoitusta eli tieliikennepäivystäjien käyttämien työkalujen yhtenäistämistä yhden sovelluksen alle, joka näin helpottaisi tieliikennepäivystäjien työtä. Tämän luvun tarkoituksena on kuvata tarkemmin VMX-kameraintegraation vaatimusmäärittely- ja suunnitteluvaiheet.

4.1 VMX-kameraintegraation vaatimukset

VMX-kamerajärjestelmän integrointi aloitettiin kartoittamalla integraatioon liittyvien alustavien vaatimusten määrittelyllä. Käytännössä tämä tarkoitti selvitystä siitä, mitä ominaisuuksia VMX Client -sovelluksesta haluttiin tuoda kamerapalvelimen ja kameratyökalun vastuulle sekä käyttäjien eli tieliikennepäivystäjien tarpeiden kartoittamista esimerkiksi kameratyökalun käyttöliittymän suunnittelun kannalta. T-LOIK-projektissa on käytössä iteratiivinen ohjelmistokehitysmalli, joten vaatimukset kehittyvät projektin edetessä. Iteratiivisella kehitysmallia tarkoitetaan mallia, jossa vaatimukset voidaan jakaa iteraatioihin, joissa ne toteutetaan [18]. Iteraation päätteeksi toteutettu vaatimus demonstroidaan asiakkaalle. Iteratiivisen kehitysmallin etuna on se, että mikäli vaatimuksia ei heti alussa tunneta tarkasti, niitä voidaan aina tarkentaa iteraation päätteeksi.

Ohjelmistotuotannon perusteiden mukaisesti VMX-kameraintegraation vaatimusmäärittely on jaettu kolmeen eri kategoriaan: toiminnallisiin ja ei-toiminnallisiin vaatimuksiin sekä reunaehtoihin [18].

4.1.1 Toiminnalliset vaatimukset

Toiminnallisilla vaatimuksilla tarkoitetaan toimintoja ja ominaisuuksia, jotka ohjelmiston tulee sisältää. Lisäksi ne kuvaavat, miten ohjelmiston tulee toimia. Toiminnalliset vaatimukset eivät niinkään ota kantaa ohjelmiston teknisiin ratkaisuihin, vaan kuvaavat, miten ohjelmiston tulisi käyttäytyä tietyissä tilanteissa. Esimerkiksi toiminnallinen vaatimus määrittelee, miten ohjelmiston tulisi reagoida käyttäjän antamiin syötteisiin.

VMX-kameraintegraatiolla on kaksi suurta toiminnallista vaatimusta. Ensimmäinen toiminnallinen vaatimus on VMX-kameran videokuvan seuraaminen kameratyökalusta samaan tapaan kuin nykyisten kuva-aikasarjojen ja häiriöhavaintovideoiden seuranta. Ka-

meratyökalun on siis pystyttävä aukaisemaan samanaikaisesti useita tieliikennepäivystäjän valitsemia VMX-kameroiden videokuvia näkymään. Videokuvaa ei tallenneta paikallisesti tieliikennepäivystäjän työasemalle, kuten HHJ-videoita, vaan videokuvan on oltava reaaliaikaista. Käytännössä tämä tarkoittaa videokuvan suoratoistoa

Toinen suuri toiminnallinen vaatimus on saman VMX-kameran hallinta, jonka videokuva on kameratyökaluun avattu. VMX-kamerakuvan hallinnalla tarkoitetaan sitä, että kyseistä VMX-kameraa pitää pystyä kääntämään. Kääntämisen on oltava mahdollista sekä kameran esiasentoja (valmiiksi kalibroituja suuntia) hyödyntäen että näppäimistön nuolinäppäimiä ja hiirtä käyttäen. Tieliikennepäivystäjän on siis pystyttävä valitsemaan kameran esiasentolistalta esiasento, jonka seurauksena kamera kääntyy kyseiseen esiasentoon. Lisäksi kameraa on pystyttävä liikuttamaan manuaalisesti hiirellä hiiren siirtosuunnan mukaisesti sekä nuolinäppäimien avulla näppäimen osoittamaan suuntaan.

VMX-kameran hallintaan kuuluvat myös kamerakuvan suurentaminen, loitontaminen sekä fokusointi. Näiden toimintojen käyttäminen on onnistuttava näppäimistöä (plus- ja miinusnäppäimet) käyttäen sekä hiiren rullalla.

Näiden lisäksi on määritelty useampia pienempiä hallintaan liittyviä toiminnallisuuksia, kuten mahdollisuus lukita kamera tietyksi aikaa kameratyökalusta. Tieliikennepäivystäjän on pystyttävä lukitsemaan kamera tiettyyn asentoon, jolloin kamera jää määritellyksi ajaksi sen hetkiseen asentoonsa. Tämä ominaisuus on vaadittu, koska VMX-kamerajärjestelmässä on automaattinen kotiasentoon kääntäminen tietyn ajan kuluttua viimeisimmästä kameran kääntöoperaatiosta. Jos kamera halutaan pitää samassa asennossa pidemmän aikaa, on se lukittava, jotta kamerajärjestelmä ei suorita automaattista kotiasentoon kääntöä.

Edellä kuvatut toiminnalliset vaatimukset ovat kriittisimmät VMX Client -sovelluksesta perittävät toiminnot, joita tarvitaan tieliikennepäivystäjän tehtävien hoitamisessa. VMX-kameraintegraatio on inkrementaalinen prosessi, jossa vaatimuksia tarkennetaan iteraatioiden jälkeen. Tämä tarkoittaa sitä, että tulevaisuudessa uusien vaatimusten mukana muitakin VMX Client -sovelluksen toimintoja tuodaan kameratyökalun puolelle.

4.1.2 Ei-toiminnalliset vaatimukset

Siinä missä toiminnalliset vaatimukset kuvaavat, mitä ohjelmiston pitäisi tehdä, ei-toiminnalliset vaatimukset kuvaavat, miten ohjelmiston pitäisi kyseisen toiminnalliset vaatimukset toteuttaa. Ne asettavat ehdot ja rajat toiminnallisten vaatimusten toteutuksille. Ei-toiminnalliset vaatimukset voidaan jakaa useisiin kategorioihin, kuten laadullisiin, arkkitehtuurillisiin sekä kehitystyöhön liittyviin vaatimuksiin.

Laadullisia ei-toiminnallisia vaatimuksia voivat olla esimerkiksi luotettavuuteen ja suorituskykyyn liittyvät vaatimukset. VMX-kameraintegraation kannalta laadullisia vaatimuksia ovat nimenomaan edellä mainitut ominaisuudet. Integraation myötä kameratyökalun on oltava toiminnaltaan luotettava, tarkoittaen sitä, että mahdollisissa häiriötilanteissa kameratyökalun on pystyttävä toimintakykyisenä vioista huolimatta. Kameratyökalun on oltava vikasietoinen eli esimerkiksi VMX-kameran videon suoratoistossa tapahtuvasta viasta on palauduttava asianmukaisella tavalla. Käytännössä tämä tarkoittaisi suoratoistoyhteyden avaamista uudelleen toisesta lähteestä. Suorituskykyyn liittyvät rajoitteet liittyvät kameratyökalun resurssien käyttöön videon suoratoistossa sekä kamerapalvelimen nopeaan kameratyökalun tekemien pyyntöjen käsittelyyn. Kameratyökalun on kyettävä toistamaan samanaikaisesti noin kymmentä eri videokuvaa käyttäen muistia ja prosessorin laskenta-aikaa siedettävissä määrin eli T-LOIKin muiden käyttöliittymäsovellusten toiminta ei saa häiriintyä tai hidastua. Lisäksi kamerapalvelimen on vastattava riittävän nopeasti esimerkiksi videokuvan avaamiseen liittyviin pyyntöihin, jotta kameratyökalun käyttö säilyy mahdollisimman sujuvana. Arkkitehtuurilliset vaatimukset ovat käyttö- ja asennusympäristöön liittyviä vaatimuksia, kuten ohjelmistokirjastoihin ja käyttöjärjestelmään liittyvät vaatimukset. VMX-kameraintegraatiolle ei ole määritelty erikseen arkkitehtuurillisia vaatimuksia, vaan sen on noudatettava samoja vaatimuksia kuin muidenkin T-LOIK-järjestelmän osien. Tämä tarkoittaa sitä, että integraation myötä kamerapalvelimen on edelleen oltava yhteensopiva saman palvelinalustan kanssa kuin muidenkin T-LOIKin palvelinsovellusten. Kameratyökalun puolella arkkitehtuurilliset vaatimukset näkyvät ohjelmakirjastojen valinnassa. Käytettävät kirjastot on valittava niin, että ne ovat yhteensopivia liikennekeskuksissa käytössä olevien työasemien käyttöjärjestelmien kanssa. Tässä tapauksessa valittujen kirjastojen on oltava uusimmille Windows-käyttöjärjestelmille yhteensopivia.

Kehitystyön vaatimuksia VMX-kameraintegraatiolle ei ole myöskään erikseen määritelty, vaan oletusarvoisesti on noudatettava T-LOIK-projektin kehitystyön yleisiä käytäntöjä. Kehitystyön vaatimuksia ovat esimerkiksi kustannuksien ja kehityksen takarajojen huomioon ottaminen, ohjelmiston ylläpidettävyys ja testattavuus. Integraatiosta syntyvien uusien komponenttien on oltava ylläpidettäviä ja mahdollisimman helposti jatkokehitettäviä ja varauduttava toiminnallisten vaatimusten mahdollisiin muutoksiin. Uusille komponenteille on myös suoritettava kattava testaus ennen niiden käyttöönottoa.

4.1.3 Reunaehdot

Ohjelmistotuotannossa reunaehdoilla tarkoitetaan eräänlaisia projektissa voimassa olevia rajoitteita ja oletuksia. Tällaiset ongelmakentän ominaisuudet ovat projektissa aina voimassa, joten ne on otettava huomioon jo vaatimusmäärittely- ja suunnitteluvaiheessa[19]. Reunaehdot ovat luonteeltaan samankaltaisia kuin ei-toiminnalliset vaatimukset. Erona on se, että reunaehtoja ei voida kiertää, eikä niistä voida näin ollen neuvotella vaatimus-

määrittelyn aikana. Reunaehtoja voivat olla esimerkiksi ympäristöön liittyviä ehtoja, kuten kehitysalustaan, ohjelmointikieleen tai työaseman fyysiseen laitteistoon liittyvät ehdot.

VMX-kameraintegraation reunaehtoihin kuuluvat ohjelmointikieleen liittyvät rajoitteet. Kun kyseessä on toisen järjestelmän integrointi jo olemassa olevaan järjestelmään, olemassa oleva järjestelmä asettaa integraatiolle reunaehtoja. Kamerapalvelin on toteutettu Java-ohjelmointikielellä, mikä tarkoittaa sitä, että VMX-kameraintegraatio on kamerapalvelimeen toteutettava Java-ohjelmointikielellä tai vähintäänkin ohjelmointikielellä, jota voidaan kääntää Javan tavukoodiksi ja näin ollen ajaa Javan virtuaalikoneen sisällä. Javan virtuaalikone (engl. Java Virtual Machine) eli JVM on abstrakti, laitteistoriippumaton tietokone, jolla on omat konekäskynsä[20]. JVM ei suoraan ymmärrä Java-ohjelmointikieltä, vain siitä käännettävää binääristä muotoa eli tavukoodia. JVM kykenee siis ajamaan koodia, joka on kirjoitettu ohjelmointikielellä, josta voidaan kääntää Java-tavukoodia.

Kameratyökalu asettaa myös ohjelmointikieleen oman rajoitteensa. Kuten luvussa 2 on kuvattu, kameratyökalu on toteutettu C#-ohjelmointikielellä .NET Framework -sovelluskehystä hyödyntäen. Kameratyökaluun tehtävät integraatiosta johtuvat muutokset on siis toteutettava tällä kielellä. Lisäksi näkymien toteutus oli tehtävä WPF-kirjaston XAML-kuvauskielellä.

Käyttöjärjestelmä ja palvelinsovelluksien sovellusalusta muodostavat VMX-kameraintegraation ympäristölliset reunaehdot. Liikennekeskusten työasemat käyttävät Windows-käyttöjärjestelmää, joten kameratyökalusta luotava binäärimuoto on oltava integraation toteutuksen jälkeenkin Windows-yhteensopivaa. Lisäksi kamerapalvelimesta käännetty ja pakattu sovellus on oltava palvelinkoneille asennetun sovellusalustan kanssa yhteensopiva. Ympäristöllisiin reunaehtoihin voidaan T-LOIKin tapauksessa lukea myös tietokannan valinta. Kaikkien palvelinsovelluksien on käytettävä samaa tietokantaversiota, joten tämä rajaa VMX-kameraintegraatiossa muut tietokannat pois. Edellä mainittuja reunaehtoja on toki mahdollista kiertää, mutta esimerkiksi uuden tietokantaversion tai palvelinalustan asentaminen VMX-kameraintegraation takia toisi tarpeetonta ja ylimääräistä työtä, mikä hyötyyn nähden ei ole järkevää ja rikkoisi yhtenäisten pohjan T-LOIKin eri komponenttien kesken.

Liikennekeskusten työasemien laitteisto rajaa sovellusten muistinkäyttöä. Työasemissa on asennettuna kahdeksan gigatavua keskusmuistia, joten kameratyökaluun tehtävät muutokset eivät saa kasvattaa muistinkäyttöä siinä määrin, että keskusmuistia ei riitä enää muiden sovellusten ajoon. Keskusmuistin luomat rajoitteet ovat tavallaan ei-toiminnallinen vaatimus, sillä keskusmuistin lisääminen on neuvoteltavissa oleva asia, mutta lisämuistin hankkiminen luo siinä määrin kustannuksia, että sitä tulee välttää.

4.2 VMX-kameraintegraation suunnittelu

Alustavan vaatimusmäärittelyn valmistuttua aloitettiin VMX-kameraintegraation suunnitteluvaihe. Suunnitteluvaiheessa kartoitettiin, minkälaisia uusia entiteettejä ja rajapintoja kameraintegraatio vaatisi sekä mitä teknologioita videon suoratoistossa käytettäisiin.

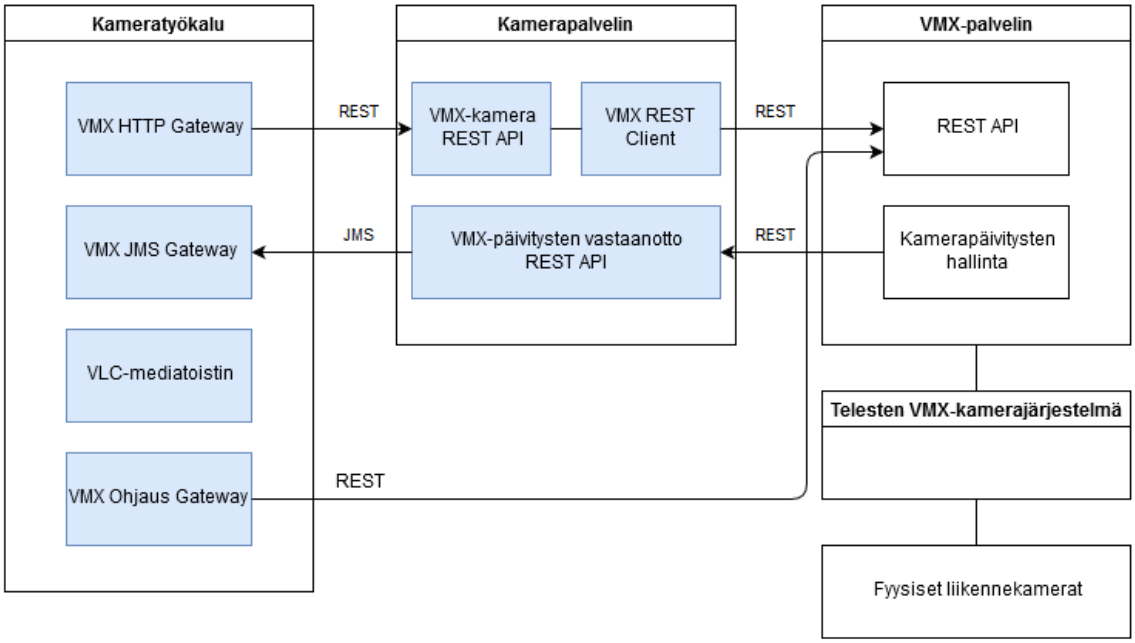
VMX-kameraintegraation suunnittelu oli rajattu lähinnä kamerapalvelimen ja kameratyökalun piiriin. Suunnittelun alussa VMX-kamerajärjestelmän päälle oli jo REST-palvelin olemassa sekä alustava REST-rajapinta mietittynä, jota kamerapalvelin tulisi hyödyntämään. Tämä VMX-REST-palvelin oli kuitenkin vielä tässä kohtaa vahvasti kehitysvaiheessa ja mahdollisiin kamerapalvelimen vaatimiin muutostarpeisiin oltiin varauduttu. VMX-palvelinta oli aloitettu kehittämään hieman aikaisemmin, jotta kamerapalvelimelle tehtävä integraatiotyö olisi helpompaa ja varsinaista integraatiotestausta pystyttiin suorittamaan. VMX-palvelin ei siis kuulu tämän työn piirissä toteutettavaan osaan, vaikka onkin olennainen entiteetti integraation kannalta.

4.2.1 Kamerapalvelimen suunnittelu

Suunnittelu aloitettiin miettimällä, mitä kamerapalvelimelta vaaditaan, jotta kameratyökalu onnistuu avaamaan VMX-kameran videokuvan suoratoistona. Kommunikoinnissa VMX-palvelimelta kamerapalvelimeen päin oltiin suunniteltu käytettävän vain yhtä REST-rajapintaa eli käytännössä kaikki kommunikointi tapahtuisi HTTP-protokollan avulla. REST-kutsujen data päätettiin kuvata JSON-syntaksia (engl., Java Object Notation) käyttäen. JSON on rakenteisen datan kuvaamiseen käytetty tekstipohjainen formaatti[21]. Kamerapalvelimen kannalta tämä tarkoitti uuden REST-asiakaskomponentin toteuttamista, joka osaa lähettää HTTP-pyyntöjä JSON-muodossa sekä tulkita VMX-palvelimelta vastaanotettuja JSON-muodossa olevia vastauksia.

Kameratyökalulla on useita samanaikaisia käyttäjiä eli jokaisen tieliikennepäivystäjän työasemalla on kameratyökalusta päällä yksi instanssi, joka keskustelee kamerapalvelimen kanssa. Jokaisella instanssilla on oltava validit tiedot kameran tilasta, kuten esimerkiksi näkymässä näytettävästä kameran suunnasta. Jos kameraa käsketään yhdeltä työasemalta kääntymään suunnasta ”Itään” suuntaan ”Länteen”, on tämän suuntatiedon muutos näyttävä kaikilla työasemilla. Samaa kameraa voidaan myös ohjata useammalta työasemalta yhtä aikaa, mikä saattaa johtaa outoon lopputulokseen. Tällaiset tilanteet katsottiin kuitenkin olevan harvinaisia, joten yhtäaikaisten kääntöjen estoa ei suunniteltu toteutettavaksi. Edellä mainittujen kameran tilaa koskevien päivitysten hallinta jätettiin kamerapalvelimen vastuulle. Kamerapalvelin ei kuitenkaan näe kameroiden oikeaa tilaa, vaan luottaa siihen, että VMX-palvelin ilmoittaa tilanmuutoksista. Nämä tilapäivitykset ovat käytännössä kameran lukitusstatuksen ja suunnan muutoksia. Kamerapalvelin suunniteltiin vastaanottavan tilapäivityksiä tekemällä VMX-palvelimelle tilauksia. Tilaus luotaisiin siten, että kamerapalvelin tekisi VMX-palvelimen REST-rajapinnan määrittämän ti-

lauspyynnön, jossa se määrittelisi tilauksen keston, kamerat, joita tilaus koskisi sekä takaisinkutsu-osoitteen (engl. callback), jonne VMX-palvelin lähettäisi päivityksiä. Näin VMX-palvelin pysyisi riippumattomana ja tietämättömänä varsinaisesta tilaajasta. Kamerapalvelimelle tarvittiin siis myös REST-asiakaskomponentin lisäksi HTTP-viestejä vastaanottava rajapinta, joka osaisi lähettää päivitykset myös eteenpäin kaikille auki oleville kameratyökaluinstansseille. Kameratyökaluinstanssien ja kamerapalvelimen välillä päivitysten lähetys toteutettaisiin JMS-viestiväylän avulla, jolloin kamerapalvelimen ei tarvitsisi tietää kameratyökaluinstansseista eli vastaanottajista mitään. Tällä tapaa kaikki kameratyökalujen VMX-kameroihin liittyvä kommunikointi pysyisi kamerapalvelimella ja ainoastaan kamerapalvelimen sallittaisiin kommunikoivan VMX-palvelimen kanssa kuvan 5 mukaisesti.



Kuva 5: *VMX-kameraintegraation entiteettikaavio*

Kuva 5 hahmottaa VMX-kameraintegraation kokonaiskuvaa ja esittää korkean tason entiteettien suhteita toisiinsa. Sinisellä taustalla merkatut komponentit ovat tämän työn piiriin kuuluvat komponentit, jotka on määrä toteuttaa VMX-kameraintegraatiossa.

VMX-kameraintegraation määrittelyssä päätettiin myös, että kamerat tulee lukita käännettyyn asentoonsa kahdeksi minuutiksi viimeisimmästä ohjauskomennosta. Tämä tehtiin siksi, että VMX-kamerajärjestelmä käyttää omaa 15 minuutin oletusaikaa kameroiden lukituksiin ja sitä haluttiin lyhentää. Tämän ajan muuttaminen suoraan VMX-kamerajärjestelmän konfiguraatiosta ei ollut mahdollista T-LOIK-projektin puolelta, vaan sitä hallinnoi Teleste. Oletuslukitusajan ylikirjoitus kamerapalvelimen puolella toisi siis lisää logiikkaa itse käännön suorittamiseen, sillä kamerapalvelimen pitäisi tarkistaa ennen kääntämistä, että kamera on lukittu kahdeksi minuutiksi ennen kuin varsinaista kääntöä tehdään. Lisäksi tieliikennepäivystäjän on voitava lukita kamera pidemmäksi aikaa ilman,

että kameran kääntö ylikirjoittaisi lukitusajan arvon takaisin kahdeksi minuutiksi, joten tiedettiin, että lukitusaikoja on ylläpidettävä. Tämä ominaisuus suunniteltiin toteutettavan niin, että kamerapalvelin tallentaisi lähettämiensä lukituspyyntöjen kestot tietokantaan. Kun kameraa käännetään, voidaan tarkistaa suoraan tietokantakyselyllä, onko kameralla voimassa olevaa lukitusta ja sen mukaan lukitus joko tehdään tai jätetään tekemättä.

4.2.2 Kameratyökalun integraation suunnittelu

Kameratyökalun osalta suunnittelu keskittyi suurilta osin uuden mediatoistimen valintaan, jonka tuli kyetä purkamaan sekä H.264- ja MPEG-4-koodattua videota. Osa kameroista käytti myös vanhempaa MPEG-2-koodausta, mutta sen tukeminen päätettiin jättää tekemättä, sillä nämä kamerat oli määrä jossakin vaiheessa tuoda MPEG-4-koodauksen piiriin. Lisäksi mediatoistimen piti pystyä avaamaan videon suoratoisto sekä HTTP-osoitteiden että SDP-tiedostojen kautta. Lopuksi oli varmistettava, että mediatoistinta voitaisiin käyttää WPF-kirjastolla toteutetuissa näkymissä. Käytännössä se tarkoitti sitä, että mediatoistimesta oli oltava WPF-elementtitoteutus olemassa tai se oli tehtävä itse. Lisäksi mediatoistimen tuli olla suorituskyvyltään riittävän kevyt, jotta useita samanaikaisia suoratoistoja voitaisiin avata samaan näkymään. Jokainen suoratoisto nimittäin vaatii oman mediatoistimen, joten mediatoistimia on käynnissä kymmenkunta kerrallaan. Tällainen määrä suoratoistoja on haasteellista suorituskyvyn kannalta.

Kartoitusta mediatoistimien suorituskyvystä usean videon toistamisessa oltiin tehty jo aikaisemmin VLC-mediatoistimen osalta. VLC on ilmainen, avoimen lähdekoodin mediatoistin, joka tukee suurta osaa videopakkaustavoista, kuten esimerkiksi edellä mainittuja H.264- ja MPEG-4-koodauksia. VLC-mediatoistin osoittautui lupaavaksi vaihtoehdoksi. Sen lisäksi muita potentiaalisia vaihtoehtoja olivat DirectShow-mediakirjasto sekä suoraan WPF-kirjastosta saatava mediaelementti, joka oli jo käytössä kameratyökalussa HHJ-videoiden toistossa. WPF:n toteuttama mediaelementti käyttää samoja kirjastoja, joita Windows Media Player-toistin käyttää, mikä tarkoitti sitä, että kyseinen mediatoistin tuli olla asennettuna työasemalla, jotta WPF:n mediaelementtiä voitiin käyttää.

WPF:n mediaelementin huomattiin nopeasti olevan käyttökelvoton VMX-kameraintegraatiossa, sillä kyseisen elementin taustalla käyttämä mediatoistin ei ymmärtänyt SDP-formaatissa olevia suoratoistotietoja. Tämä vaihtoehto hylättiin ja fokus siirtyi VLC- ja DirectShow-mediatoistimiin. DirectShow on Microsoftin kehittämä mediakirjasto, joka tarjoaa komponentit muun muassa videoiden ja äänen toistamiseen Windows-käyttöjärjestelmässä[22]. DirectShow-mediatoistimelle löytyi sopiva kirjasto nimeltään WPF Mediakit, joka toteuttaa tarvittavat WPF-elementit mediatoistimen päälle. Vastaavanlaisesti VLC-mediatoistimelle löytyi kaksi potentiaalista WPF-elementit toteuttavaa kirjastoa: Vlc.Dotnet ja Meta.VLC.

WPF Mediakit-, Vlc.Dotnet- ja Meta.VLC-kirjastoille tehtiin nopeat prototyypit suorituskykymittausta varten. Käytännössä prototyypit olivat siis WPF-sovelluksia, jossa näkymässä oli testattavan kirjaston toteuttama mediatoistinta käyttävä WPF-elementti. Nopeasti huomattiin, että usean mediatoistimen käyttö pelkän prosessorin varassa oli mahdotonta. Kirjastojen suorituskykyä testattiin yrittämällä toistaa useita H.264-koodattuja videoita kirjaston toteuttamalla WPF-elementillä. Sekä DirectShow- että VLC-mediatoistimet oletusarvoisesti suorittivat H.264-pakkauksen dekoodaamisen prosessorin avulla, jonka seurauksena prosessorin käyttö nousi 100 prosenttiin jo muutamaa videota samanaikaisesti toistettaessa.

Prosessorin raskas kuormitus saatiin ratkaistua siirtämällä videon dekoodaus prosessorilta näytönohjaimen laskettavaksi. WPF Mediakit-kirjaston kanssa tämä ei kuitenkaan osoittautunut kovin suoraviivaiseksi operaatioksi. Kirjaston käytön edellytyksenä nimittäin on, että DirectShow-kirjaston käyttämät videon renderöintiin tarvittavat suodattimet löytyvät ajoympäristöstä. Käytännössä tämä tarkoittaa sitä, että liikennekeskusten työasemille olisi asennettava joko Windows SDK, jonka mukana tulisi DirectShow-kirjaston oletussuodattimet tai jokin kolmannen osapuolen koodekkikirjasto, joka toteuttaa DirectShow'n kuvaaman suodatin-rajapinnan. Uusien kirjastojen asentaminen olisi välttämätöntä myös VLC:n käytössä, mutta ratkaisevaksi eroksi muodostui konfiguroitavuuserot näytönohjaimeen liittyen. WPF Mediakit ei nimittäin tarjoa mitään järkevää tapaa laitteistokiihdytyksen eli käytännössä näytönohjaimen hyödyntämiseen videon renderöinnissä. Tuki laitteistokiihdytykselle kyllä löytyy DirectShow'n puolelta, mutta mitään suoraviivaista ohjelmallista tapaa sen hyödyntämiseen WPF Mediakit ei testaushetkellä tarjonnut. Suositelluin tapa laitteistokiihdytyksen käytölle oli käydä laittamassa kiihdytys päälle manuaalisesti asennettujen videosuodattimien asetuksista, mikä ei ole järin optimaalinen ratkaisu. WPF Mediakit -kirjaston suorituskyky ei ollut myöskään parempi verrattuna VLC-mediatoistimen WPF-kirjastoihin niin, että sitä olisi kannattanut alkaa sen takia käyttämään.

WPF Mediakit -kirjaston tutkimisen jälkeen siirryttiin valittuihin Vlc.Dotnet- ja Meta.Vlc-kirjastoihin. Suorituskykyä testattiin toistamalla samaa H.264-koodattua videota näkymässä, jossa oli kuusi mediatoistinta toteutettuna kyseisillä kirjastoilla. Videon toistamiseen käytettiin kummassakin testissä samoja media-asetuksia, jotta tulokset olisivat mahdollisimman vertailukelpoisia. Kumpikin kirjasto vaatii toimiakseen myös libVLC-mediakirjastot työasemalta. LibVLC on SDK, jonka tarjoamia mediatoimintoja VLC-mediatoistin käyttää.

Vertailtaessa tuloksia huomattiin, että VLC.Dotnet käytti keskusmuistia hieman enemmän kuin Meta.Vlc. Kuudella toistimella Vlc.Dotnet varasi keskusmuistia käyttöönsä noin 850 megatavua, kun taas Meta.Vlc varasi vain noin 670 megatavua. Suoritinta molemmat kirjastot kuormittivat suurin piirtein yhtä paljon eikä näytönohjaintakaan hyödynnettäessä merkittävää eroa ollut. Näin ollen keskusmuistin käytössä oli ainoa huomattava ero suorituskyvyn kannalta.

Tutkittaessa kirjastojen toteutuksia Vlc.Dotnet-kirjaston huomattiin käyttävän todellisuudessa Windows Forms -käyttöliittymäkirjastoa. Windows Forms on Microsoftin kehittämä vanha käyttöliittymäkirjasto, jonka uuden WPF-käyttöliittymäkirjaston on määrä korvata.

WPF tukee Windows Forms -kirjaston käyttöä yhdessä muiden WPF-elementtien kanssa, mutta tämä ei ole aivan ongelmatonta johtuen siitä, että WPF ja Windows Forms käyttävät täysin eri teknologioita käyttöliittymän renderöintiin. WPF käyttää renderöintiin DirectX-ohjelmointikirjaston grafiikkarajapintaa[23], kun taas Windows Forms käyttää vanhempaa Graphics Device Interface-grafiikkakirjastoa[24].

Kun Windows-alustoilla käyttöliittymälle luodaan ikkuna, sille luodaan kahva, jonka datatyyppi on HWND. Jokainen HWND-kahva osoittaa omaan varattuun alueeseensa näytön ruudulla. Tästä alueesta käytetään nimitystä ilmatila (engl. airspace)[25]. Näin ollen jokainen näytön pikseli kuuluu kerrallaan vain yhdelle HWND-kahvalle. Lisäksi HWND-kahvan päässä oleva ilmatila voi käyttää vain yhtä renderöintiteknologiaa kerrallaan, mikä luo rajoitteita Windows Forms -kontrollien renderöinnille WPF-kontrollien päälle. Kaikkien kontrollien on siis käytettävä samaa renderöintiteknologiaa saman ilmatilan sisällä. Useamman eri käyttöliittymäkirjaston käyttö voi siis johtaa nopeasti näiden rajoitteiden rikkomiseen, jonka seurauksena käyttöliittymä voi käyttäytyä odottamalla tavalla esimerkiksi fokusoinnin tai skaalauksen kanssa.

WPF on sovittanut yhteen Windows Forms -kontrollien käytön omassa ilmatilassaan WinFormsHost-luokan avulla. Käytännössä tämä on tapa kiertää edellä mainittua rajoitetta, joka määrää, että Windows Forms -kontrolleja ei voida suoraan asettaa WPF-tilaan. WindowsFormsHost-elementti nimittäin luo uuden ikkunan WPF-ikkunan päälle, johon Windows Forms -kontrolleja voidaan asettaa. WindowsFormsHost-elementillä on kuitenkin useita erilaisia rajoitteita. Esimerkiksi WPF-elementtien limittyminen on toteutettu niin, että Windows Forms -kontrollit ovat aina piirrettynä näytölle WPF-elementtien päälle. Näin WPF-elementit on pakotettu jäämään WindowsFormsHost-elementin sisällön alle. Kameratyökalulla on taas tarve näyttää esimerkiksi WPF-dialogeja videoelementin päällä, joten Vlc.Dotnet sellaisenaan on käyttökelvoton. Yksi ratkaisu tähän olisi ollut luoda uusi WPF-ikkuna WindowsFormsHost-elementin päälle ja sitoa sen koko WindowsFormsHost-elementtiin. Tällöin tähän erilliseen WPF-ikkunaan olisi voitu renderöidä tarvittavat WPF-elementit. Ratkaisu on kuitenkin tarpeettoman monimutkainen eikä välttämättä korjaa fokusointiin ja skaalauksiin liittyviä ongelmia. Lisäksi Vlc.Dotnetin käyttö olisi vaatinut Windows Forms -kirjastojen tuontia kameratyökaluun.

VLC-mediatoistimen WPF-kirjastojen vertailun lopputuloksena päädyttiin siis valitsemaan Meta.Vlc-kirjasto VLC-mediatoistimen toteuttamiseksi kameratyökaluun. Meta.Vlc käyttää ainoastaan WPF-kirjastoa käyttöliittymäelementtien toteuttamiseen. Meta.Vlc:n käyttö vaatii, että tarvittavat VLC-kirjastot löytyvät työasemalta, joten tämän

kirjaston valinnan sivutoimena kaikkiin liikennekeskuksien työasemiin asennettiin VLC-mediatoistin, jonka mukana kirjastot tulevat.

4.2.3 Suunnitteluvaiheessa ilmenneet ongelmat

VMX-kameraintegraation suunnittelua hankaloittivat erityisesti vähäiset tiedot itse VMX-kamerajärjestelmästä. Dokumentaatiota kamerajärjestelmän toiminnasta ei ollut, eikä Telesten puoleltakaan liiammin lisätietoja saatu.

Erityisesti kameran ohjaamisen suunnittelua hankaloitti se, että tietoja kameran lukituksien pituudesta ei ollut ennen kuin se varta vasten testattiin VMX Clientin avulla. Sama ongelma toistui, kun mietittiin kotiasentoon kääntämistä. Tietoa ei siis ollut siitä, kuinka kauan kestää viimeisestä käännöstä, ennen kuin VMX-kamerajärjestelmä itse automaattisesti kääntää kameran takaisin kotiasentoon.

Epäselvää oli myös se, millä logiikalla VMX-kamerajärjestelmä priorisoi käyttäjiään. Käytännössä tämä tarkoitti sitä, ettei tiedetty, mitä tapahtuisi, jos VMX Clientin käyttäjä ja kameratyökalun käyttäjä käsittelisivät samaan aikaan samaa kameraa. VMX-kamerajärjestelmän käyttäjillä on olemassa keskinäiset prioriteetit, mutta kumpi käyttäjä olisi prioriteetiltaan korkeammalla, oli epäselvää. Tämäkin oli arvioitava VMX Client-sovellusta käyttämällä ja kokeilemalla.

Hieman samankaltainen suunnitteluongelma oli T-LOIK-käyttäjien priorisoinnin kanssa. Jos usea käyttäjä yrittää kääntää samaa kameraa samaan aikaan kameratyökalun kautta, ei ollut määritelty, mitä tällaisessa tilanteessa tulisi tapahtua. Tällaiseen tilanteeseen olisi määriteltävä omanlaisensa priorisointilogiikka, mutta siihen ei kuitenkaan ryhdytty. Kysymys jäi auki ja sen pohdinta siirrettiin VMX-kameraintegraation jatkokehitykseen. Tämän työn osalta suunnittelua jatkettiin luottamalla siihen, että todennäköisyys kyseiselle tilanteelle on varsin pieni, joten ongelma sivuutettiin.

Lisäksi kameran ohjaamista kahden palvelimen kautta HTTP-protokollaa käyttäen oli suorituskyvyltään erittäin hankala arvioida ilman testaamista. Jokainen ohjauspyyntö kulkee kamerapalvelimen kautta VMX-palvelimelle, joka taas lähettää pyynnön lopuksi VMX-kamerajärjestelmälle. Kameran kääntäminen hiirtä käyttäen tiedettiin lähettävän paljon ohjaamispyyntöjä, jolloin HTTP-pyyntöjen vastaanotto saattaisi mennä tukkoon joko kamerapalvelimella tai VMX-palvelimella. Epäiltiin, että kahden erillisen REST-rajapinnan käyttö ohjaamiseen olisi aivan liian hidasta, joten toiseksi vaihtoehdoksi nousi HTTP-pyyntöjen lähettäminen suoraan kameratyökalulta VMX-palvelimelle eli kamerapalvelin pudotettaisiin välistä pois. Tämä ratkaisu loisi kuitenkin uusia haasteita muun muassa kameran lukituksen seuraamiselle. Loppuen lopuksi suunnitteluvaiheessa päätettiin, että toteutetaan ensimmäinen vaihtoehto eli kierrätetään kaikki pyynnöt edelleen kamerapalvelimen kautta ja testataan, onko ohjaamisen http-pyyntöjen käsittely riittävän

nopeaa. Toinen vaihtoehto otettaisiin uudelleen harkintaan, mikäli ohjaaminen olisi liian hidasta ehdotetulla ratkaisulla.

5. VMX-KAMERAINTEGRAATION TOTEUTUS

VMX-kameraintegraation toteuttaminen on jaettu useaan eri osaan VMX Client -sovelluksesta kameratyökaluun integroitavien ominaisuuksien mukaan. Näistä ominaisuuksista oleellisimmat ovat liikennekameran videokuvan katselu sekä itse kameran ohjaaminen. Integraatio sisältää muitakin ominaisuuksia, mutta ne jätetään tarkastelun ulkopuolelle. Seuraavat aliluvut keskittyvät kuvailemaan edellä mainittujen ominaisuuksien toteutusprosessia.

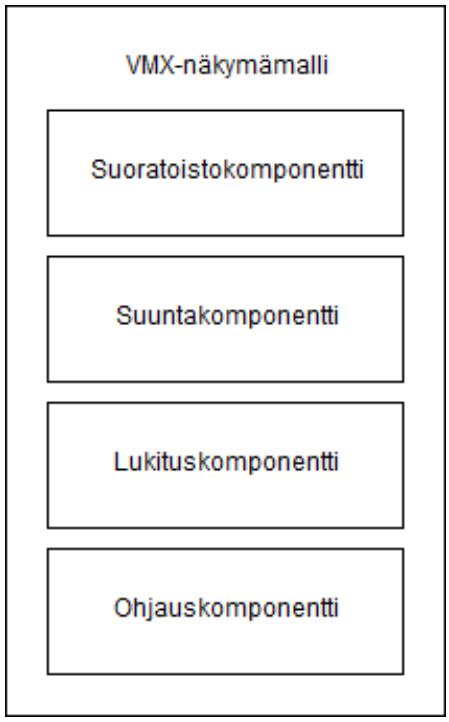
5.1 VMX-suoratoiston mallintaminen kameratyökaluun

VMX-kameroiden toimintojen mallintaminen kameratyökaluun aloitettiin samalla näkymä-näkymämalli-malli-rakenteella, jolla HHJ-videot ja kuva-aikasarjatkin oli toteutettu. Koska kyseessä ovat samat tieliikennekamerat kuin aikaisemmin, oli jo malli kameroiden tiedoista jo olemassa. Tätä samaa mallia vain laajennettiin lisäämällä VMX-tunniste uudeksi ominaisuudeksi, jonka avulla voidaan hakea oikean kameran tarkemmat ominaisuudet kameran avaamisen yhteydessä kameratyökaluun.

Täysin uusina osina lisättiin VMX-suoratoistolle näkymämalli sekä itse näkymä. Näkymämallin tuli toteuttaa kaikille kameraelementeille (eli kuva-aikasarjoille, diasarjoille, HHJ-videoille) yhteinen rajapinta, jonka avulla varmistetaan yksinkertainen, kameraelementin varsinaisesta tyypistä riippumaton käsittely. Kameratyökalussa on toteutettuna ruudukkonäkymä (kuva 1 sivulla 4), jonka taustalla on kokoelma tämän kameraelementtirajapinnan toteuttavia näkymämalleja. Ruudukon ja kameravälilehtien mallit eivät siis tunne kameraelementtien oikeita tyyppejä, vaan käsittelevät niitä vain kameraelementtirajapinnan kautta. Tästä syystä VMX-suoratoiston näkymämallin tuli noudattaa tätä rajoitetta. Tällä tapaa noudatetaan myös hyvää ohjelmointitapaa, jossa riippuvaisuus rajataan rajapintoihin, eikä ylimääräisiä toiminnallisuuksia paljasteta riippuvuushierarkian ylemmille tasoille.

VMX-suoratoiston näkymämalli jaettiin sisäiseltä toteutukseltaan pienempiin komponentteihin niin, että kullakin komponentilla on vain yksi vastuualue. Tällä lähestymistavalla vältyttiin yhdeltä suurelta luokalta, jossa olisivat kaikki toiminnot. Usean pienen, yhden vastuualueen komponentin avulla saadaan pidettyä näkymämallin kompleksisuus minimissä ja rajattua toimintojen näkyvyyttä näkymämallin sisällä. Kuva 6 esittää toteutetun näkymämallin sisäiset komponentit, jotka vastaavat kukin yhdestä osa-alueesta. Suoratoistokomponentti toteuttaa tarvittavat toiminnallisuudet suoratoiston esittämiseksi. Käytännössä toiminnot liittyvät VLC-mediatoistimen alustuksiin, kuten suoratoistolähteen asetukseen ja itse WPF-elementin luontiin. Suuntakomponentti taas huolehtii VMX-

kameran suunnan päivittämisestä käyttöliittymään. VMX-kameroilla on kalibroituja esiasentoja, joilla on yleensä jokin nimi, esimerkiksi ”Itään”. Tämä tieto näytetään käyttöliittymässä. Suuntakomponentti lukee kamerapalvelimen tarjoamaa JMS-viestiaihetta, josta kameroiden suuntapäivityksiä tulee, ja muuttaa näkymää päivitysten mukaan datansidontaa hyödyntäen.



Kuva 6: *VMX-kameran näkymämallin sisäiset komponentit*

Hieman samaan tapaan kuin suuntakomponenttikin lukituskomponentti lukee omasta JMS-viestiaiheestaan lukituspäivityksiä ja päivittää käyttöliittymää sen mukaan, onko kamera lukittuna vai ei. Lisäksi lukituskomponentti tarjoaa toiminnallisuudet kameran lukitsemiseksi eli käytännössä lukituskäskytykset kamerapalvelimen REST-rajapintaa vasten. Sekä lukitus- että suuntakomponentti hyödyntävät toiminnassaan kuvassa 5 (sivulla 27) esitettyjä HTTP- ja JMS-väyliä. JMS-väylä tarjoaa päivityksiä ja HTTP-väylä REST-kutsujen lähetyksen kamerapalvelimelle.

Viimeinen komponentti on ohjauskomponentti, joka vastaa kameran kääntämisestä ja kuvan suurentamisesta. Sen tehtävänä on ottaa vastaavan VMX-kameran näkymän kautta annettuja näppäimistö- ja hiirisyötteitä ja muuttaa syötteet VMX-palvelimen ymmärtämään muotoon, joilla varsinainen kameran kääntö voidaan onnistuneesti suorittaa.

Näistä komponenteista suoratoistoon ja ohjaamiseen keskitytään hieman syvemmin, sillä ne olivat työn ongelmallisimpia toiminnallisuuksia toteuttaa. Aliluku 5.2 kuvaa VLC-mediatoistimen integrointia ja konfigurointia videon esittämiseksi ja aliluku 5.3 kameran ohjaamisen implementointia ja sen mukana tuomia haasteita.

5.2 Kamerakuvan suoratoisto

Liikennekameroiden videokuvan suoratoistoon liittyvä logiikka on rajattu kuvassa 6 esiintyvään suoratoistokomponenttiin. Kyseinen komponentti on vastuussa suoratoiston esitykseen käytettävän WPF-elementin alustamisesta sekä suoratoiston lähteen asetuksesta.

5.2.1 Mediatoistimen WPF-elementin toteutus

Suoratoistokomponentin toteuttaminen aloitettiin toteuttamalla tarvittava WPF-käyttöliittymäelementti, joka on siis itse videokuvaa käyttöliittymässä näyttävä elementti. Suunnitteluvaiheen suorituskykytestauksen tuloksena päädyttiin Meta.VLC-kirjaston toteuttamaan WPF-elementtiin. Kyseisen elementin kirjastototeutusta ei kuitenkaan voitu vain lisätä näkymään, sillä se vaati omanlaisensa alustuksen.

Meta.VLC-kirjastosta oli saatavilla vain 32-bittisille järjestelmille käännetty versio. Kameratyökalulle tarvittiin 64-bittinen versio, sillä tieliikennepäivystäjien työasemat ovat 64-bittisiä. 32-bittisen kirjaston ongelmana oli se, että se vaati toimiakseen VLC-mediatoistimen käyttämien kirjastojen 32-bittiset versiot. Työasemille oli asennettu 64-bittinen versio VLC-mediatoistimesta, jolloin myös sen käyttämät kirjastot olivat myös 64-bittisinä versioina. Toisena vaihtoehtona olisi ollut lisätä T-LOIK-ohjelmiston asennuspakettiin VLC:n 32-bittiset kirjastot, mutta se olisi johtanut asennuspaketin koon huomattavaan kasvuun. Työasemilta löytyvien 64-bittisten versioiden hyödyntämättä jättäminen olisi ollut tässä suhteessa epäoptimaalinen ratkaisu. Tämän lisäksi saatavilla olevat 32-bittiset kirjastot olivat käännetty Meta.VLC:n versiosta 16.04, kun uusin saatavilla oleva lähdekoodi oli versiota 17.06. Loppuen lopuksi käytettävä Meta.VLC-kirjasto käännettiin itse lähdekoodin viimeisimmästä versiosta 64-bittiseen muotoon. Etuna tässä oli se, että koodiin voitiin samalla tehdä pieniä muutoksia. Esimerkiksi kirjaston tarjoama suoratoiston play-toiminto ei palauttanut mitään paluuarvoa kyseisen toiminnon onnistumiseen liittyen. VLC-kirjaston play-toiminto palauttaa boolean-tyyppisen arvon siitä, onnistuiko median toisto vai ei. Tämä korjattiin kääntämisen yhteydessä, jotta kameratyökalun koodissa pystyttiin tekemään tarvittavat virhetarkastelut.

Toteutettaessa Meta.VLC:n tarjoamaa WPF-elementtiä kameratyökaluun huomattiin nopeasti, että muitakin muutoksia lähdekoodiin olisi voitu tehdä. Kirjasto nimittäin tarjosi ulospäin vain WPF-elementin eikä elementin takana olevaa mediatoistimen toteutusta. Mikäli kirjasto olisi tarjonnut mediatoistimen erillään WPF-elementistä, olisi mediatoistin voitu alustaa erikseen ja jälkeenpäin datansidonnalla liittää WPF-elementtiin. Kyseisen muutoksen tekeminen olisi ollut suuritöinen, joten siihen ei ryhdytty, vaan tyydyttiin käyttämään WPF-elementtiä hieman epätarkoituksenmukaisella tavalla. Yleensä käyttöliittymäelementtejä käsitellään vain XAML-koodin puolella, ei suoranaisesti muun C#-koodin mukana. Nyt elementin käsittelyä jouduttiin siirtämään näkymämallin puolelle itse näkymästä. Syy tähän oli se, että elementille täytyi alustuksen yhteydessä kertoa, mitä

VLC-asetuksia sen tuli käyttää sekä syöttää hakemistopolku, josta työasemalle asennetut VLC-kirjastot löytyvät. Tämä olisi tietenkin voitu toteuttaa kovakoodaamalla hakemistopolku ja asetusparametrit suoraan XAML-koodiin, mutta silloin olisi pitänyt varmistua siitä, että jokaisella työasemalla VLC olisi asennettuna täsmälleen samaan tiedostopolkuun. Tällaista varmuutta ei tietenkään ollut ja se olisi ollut varsin ikävä rajoite muutenkin. Myöskään datansidonta ei ollut tässä kohtaa toimiva ratkaisu. Näkymässä olevat käyttöliittymäelementit nimittäin alustetaan ennen kuin näkymämalli sidotaan näkymään. Ratkaisu tähän ongelmaan on luoda elementti näkymämallissa (tarkalleen ottaen suoratoistokomponentissa) ja jälkeenpäin dynaamisesti lisätä elementti näkymään. Kamerakuvan avaamisen yhteydessä näkymä jätettiin siis alkuun tyhjäksi ja elementti luotiin samaan aikaan näkymämallin kanssa. Tällöin näkymämallissa voitiin kartoittaa tarvittavat VLC-asetusparametrit ja hakea oikea hakemistopolku VLC-kirjastoihin. Kun VLC-mediatoistin asennetaan työasemalle, sen asennushakemisto tallentuu Windows-käyttöjärjestelmän rekisteriin. Näkymämallin koodissa tämän rekisterikentän arvo voitiin lukea, millä saatiin selville oikea hakemistopolku. Näin ei syntynyt rajoitteita VLC-kirjastojen sijaintiin liittyen. Kun parametrit olivat selvillä ja VLC-elementti oli luotu, se lisättiin tyhjään näkymään eksplisiittisesti. Normaalisti WPF:ssä näkymän kaikki elementit alustetaan samaan aikaan ilman suoraa riippuvuutta näkymämalliin.

5.2.2 VLC-mediatoistimen asetusparametrit

VLC-mediatoistimelle piti syöttää myös tarvittavat asetusparametrit, joiden avulla suoratoiston laatua ja esimerkiksi dekoodauksen suorituskykyä voitiin säätää. VLC:n dokumentaatio listaa kaikki saatavilla olevat parametrit, mutta muoto, jossa ne pitää VLC-mediatoistimelle syöttää, on jokseenkin epäselvä. Dokumentaatiossa kaikilla parametreillä on etuliitteenään kaksi väliviivaa. Tätä etuliitettä käytettäessä joillakin parametreilla ei tuntunut olevan minkäänlaista vaikutusta VLC-mediatoistimen toimintaan. Selvitystyön jälkeen kävi ilmi, että toinen mahdollinen etuliite parametreille oli kaksoispiste. Monen kokeilun jälkeen tultiin johtopäätökseen, että kummatkin etuliitteet ovat valideja, mutta eri konteksteissa. Kahden viivan parametrit ovat itse toistimelle välitettäviä parametreja. Kaksoispisteellä syötetyt parametrit ovat taas toistimella soitettavan median parametreja. VLC-kirjaston dokumentaation seassa mainitaan kyllä molemmat etuliitteet sekä niiden ero. Dokumentaation tutkimisen jälkeen selvisi, että kahta väliviivaa käytetään, kun mediatoistin alustetaan. Kaksoispistettä käytetään, kun halutaan asettaa parametreja toistettavalle medialle eli tässä tapauksessa suoratoiston lähteelle. Dokumentaatiossa tämä mainitaan parametrilistan seassa melko epäselvästi, joten sen huomaaminen vaatii melko tarkkaa syventymistä parametrilistaan. Lisäksi osa parametreista toimii vain toistettavan median kanssa, mikä aiheutti ylimääräistä hämmennystä.

Parametrien formaatin selvittelyn jälkeen vuorossa oli tarvittavien parametrien selvittäminen. Tärkein parametri oli dekoodausprosessin suorittimelta näytönohjaimelle siirtävä

parametri: *avcodec-hw=dxva2*. Kyseinen parametri sallii dekoodauksen käyttävän laitteistokiihdytystä hyödykseen eli käytännössä hyödyntää näytönohjaajan laskentatehoa. Windows-ympäristöissä on käytössä rajapinta nimeltään DirectX Video Acceleration eli DXVA, jonka avulla videon dekoodaamiseen voidaan suorittimen lisäksi käyttää muuta laitteistoa hyödyksi. Avcodec taas on VLC:n käyttämä kirjasto, joka sisältää useita erilaisia koodaus- ja dekoodaustoteuksia mukaan lukien MPEG-4 Part 2- ja Part 10 -koodekkien dekooderit. Avcodec hoitaa täysin kameratyökalun suoratoiston videodatan käsittelyn, joten omaa logiikkaa dekoodaukselle ei tarvitse tehdä. Edellä mainitun parametrin ensimmäinen osa *avcodec-hw* on siis Avcodec-kirjastolle syötettävä parametrin nimi, jonka arvo kertoo Avcodec-kirjaston dekooderille, mitä rajapintaa voidaan hyödyntää dekoodauksessa. Parametrin arvo *dxva2* tarkoittaa DXVA-rajapinnan versiota 2.0. DXVA määrittelee erilaisia toimintoja, joiden suorittamisessa voidaan hyödyntää laitteistoa. Näytönohjaimen ajurin tehtäväksi jää toteuttaa DXVA-rajapinnan mukaiset toiminnot. Jos toiminnot on toteutettu rajapinnan mukaisella tavalla, videon dekoodausprosessissa kutsutaan näitä toimintoja, jolloin suoritus siirtyy joiltakin osin näytönohjaajalle.

Muitakin median alustukseen liittyviä parametreja selvitettiin ja otettiin käyttöön, kuten esimerkiksi suoratoiston puskurointiin liittyviä parametreja sekä erilaisten suodattimien kytkemiseiä pois päältä. Näiden parametrien tarkempi avaaminen ei ole kuitenkaan tämän työn puitteissa oleellista.

5.2.3 Suoratoistolähteen haku

Kun Meta.VLC-kirjaston videontoistoelementti oli saatu toteutettua kameratyökaluun sekä tarvittavat mediantoistoon liittyvät parametrit selvitettyä, oli seuraavana vuorossa suoratoiston lähteen hakutoimintojen toteuttamisen vuoro. Tässä kohtaa VMX-palvelimen toteuttama REST-rajapinta tarjosi jo toiminnon, jota kutsuttaessa VMX-kameran tunnisteella saatiin vastauksena kyseisen kameran suoratoiston lähdetiedot. Suoratoistotietojen välittäminen kameratyökaluun asti vaati HTTP-asiakaskomponentin lisäämisen kamerapalvelimeen (kuvan 5 VMX Rest client sivulla 27), joka suorittaa pyynnöt VMX-palvelimen REST-rajapintaa vasten. Lisäksi kamerapalvelimen tuli lisätä omaan REST-rajapintaansa samanlainen suoratoistokysely, jota kameratyökalu voi käyttää. Kun kamerapalvelin vastaanottaa suoratoistokyselyn, se käyttää omaa REST-asiakastaan välittämään kyselyn VMX-palvelimelle, jonka vastaus välitetään aina kameratyökaluun asti.

Kaikki uusi VMX-kameraintegraatioon liittyvä toiminnallisuus päätettiin jakaa omiin moduuleihinsa, jolloin rajattaisiin regression syntymistä kamerapalvelimen sekä kameratyökalun vanhaan toiminnallisuuteen. Kameratyökaluun tehtiin siis oma VMX-kameroihin liittyvän HTTP-kommunikoinnin hoitava komponentti (kuvan 5 VMX HTTP Gateway sivulla 27). Samaan tapaan kamerapalvelimen VMX-toiminnallisuudet eli tässä kohtaa VMX REST -rajapinta ja REST-asiakas kapseloitiin omiin moduuleihinsa.

Ensimmäisen version toteutuksen jälkeen huomattiin, että suoratoistotietojen välitys kameratyökaluun asti oli varsin hidas, jolloin katsoja joutui odottamaan tovin ennen kuin videokuvaa pysyttiin oikeasti esittämään. Tähän ratkaisuksi saatiin välimuistin käyttö kamerapalvelimessa. Kun kamerapalvelin hakee VMX-palvelimelta suoratoistotiedot, suoratoistotiedot lisätään tietovarastoon, jossa datan oletetaan olevan validia vuorokauden verran. Kun kameratyökalu haluaa kysellä suoratoistotietoja uudelleen, niitä ei tarvitse hakea VMX-palvelimelta asti, vaan omasta välimuistista. Kamerapalvelimen välimuisti on käytännössä hashmap-tyyppinen tietorakenne, jonka avaimena toimii VMX-kameran tunniste. Välimuistirakenteen toteutuksen jälkeen suoratoistotiedot joudutaan hakemaan VMX-palvelimelta vain, jos välimuistissa olevat tiedot ovat yli vuorokauden vanhat tai niitä ei sieltä löydy ollenkaan. Näin ollen vain ensimmäinen haku on hidas.

Kokonaisuudessaan VMX-kameraintegraation kamerakuvan suoratoiston toteuttaminen osoittautui varsin pitkäkestoiseksi prosessiksi. Meta.VLC-kirjaston rajoite olla paljastamatta VLC-mediatoistimen instanssia ulospäin pakotti tekemään muutamia erikoisempia ratkaisuja, kuten UI-elementin käsittelyä XAML-koodin ulkopuolella. Myös VLC-parametrien tutkiminen suorituskyvyn parantamiseksi oli pitkälti kokeilujen varassa heikon parametreihin liittyvän dokumentaation takia. Suoratoisto kuitenkin saatiin tehtyä ja siihen liittyvät vaatimukset täytettyä.

5.3 Liikennekameroiden ohjaaminen

Toinen toteutettava päätoiminnallisuus oli kameroiden ohjaaminen. Tieliikennepäivystäjien tuli pystyä kääntämään kameraa sekä suurentamaan ja loitontamaan kuvaa. Lisäksi päivystäjien tuli pystyä lukitsemaan kamera sen hetkiseen asentoonsa tietyksi ajaksi.

5.3.1 Suunta- ja lukituspäivitykset

Liikennekameroiden ohjaamisen toteutus aloitettiin kameroiden suunnan ja lukituksen tilatietojen haun toteuttamisella. VMX-palvelin tarjosi REST-rajapinnassaan toiminnallisuuden tilauksien tekemiseen. Tilaus suunta- ja lukitustietojen päivittymisestä pystyttiin asettamaan lähettämällä tilauspyyntö tilattavien kameroiden tunnisteiden kanssa VMX-palvelimelle. Kyseisen tilauspyynnön tuli sisältää myös takaisinkutsu-osoite, jonne VMX-palvelin lähettäisi HTTP-protokollan avulla suunta- ja lukituspäivityksiä kameroiden suunnan- tai lukituksen muutoksista. Tätä varten kamerapalvelimelle oli rakennettava päivityksiä vastaanottava REST-rajapinta, jonka osoite välitettäisiin VMX-palvelimelle tilauspyynnön mukana. Tämän REST-rajapinnan implementointi oli varsin triviaalia.

Seuraava vaihe oli välittää vastaanotetut VMX-palvelimen lähettämät päivitykset kameratyökalulle asti. Suunta- ja lukituspäivitykset tuli välittää kaikille kameratyökaluinstansseille, joten REST-rajapinnan käyttö tässä kohtaa olisi varsin ongelmallista, kun vastaanottajia on useita. Java Messaging Service -viestiväylä (JMS) ja erityisesti JMS-aiheen käyttö oli optimaalinen ratkaisu tällaisiin tilanteisiin. Kamerapalvelimen vastaanottamat

viestit ohjattiin omaan VMX-päivityksille tarkoitettuun JMS-aiheeseen, josta jokaiseen kameratyökaluinstanssin JMS-kuuntelija pystyy lukemaan päivitysviestit. Kun JMS-kuuntelija vastaanottaa esimerkiksi suuntaviestin, se sisältää VMX-kameran tunnisteen, jonka avulla kameratyökalussa auki olevan kameran oikea näkymämalli löydetään. Sitten tämän näkymämallin suuntatieto päivitetään näkymämallin ja WPF:n datansidonnan ansiosta, jolloin näkymämallin päivitys heijastuu automaattisesti näkymämallin näkymään eli kameratyökalun käyttöliittymän kameraruudukossa olevaan näkymään, jossa suunta esitetään.

Päivitysten tilaus suoritetaan samaan tapaan kuin suoratoistotietojen haku. Kun kamera avataan kameratyökalussa, kameratyökalu lähettää tilauspyynnön ensin kamerapalvelimen VMX-REST-rajapintaan, ja kamerapalvelin lähettää tilauspyynnön edelleen REST-asiakkaan avulla VMX-palvelimelle. Kun päivitysten tilaus on asetettu, VMX-palvelin tietää lähettää kyseisen kameran tilapäivitykset kamerapalvelimelle suunnan tai lukituksen muutoksen tapahtuessa.

Päivitysten välityksen toteuttamisen jälkeen alkoi itse lukituksen ja ohjaamisen toteuttaminen. Vaatimuksena oli, että päivystäjä pystyy lukitsemaan kameran erikseen tietyksi ajaksi, ja kameran olisi lukituttava automaattisesti kahdeksi minuutiksi, kun sitä yritetään kääntää. Lukitus toteutettiin samalla kaavalla kuin suoratoistotietojen haku. Kameratyökalu tekee lukituspyynnön kamerapalvelimelle, joka välittää pyynnön VMX-palvelimelle. Käytännössä tämä oli vain olemassa olevien REST-rajapintojen laajentamista eikä sen monimutkaisempia ratkaisuja tarvittu.

5.3.2 Ohjaaminen esiasennoilla

Kameran ohjaaminen jaettiin kahteen osaan: esiasentoihin suuntaukseen ja kameran manuaaliseen ohjaamiseen. Kameratyökalun suuntakomponentti on vastuussa esiasennoista. Suuntakomponentti sisältää kameran esiasennot, joihin kamera voidaan kääntää. Tämä esiasentolista sidottiin näkymässä olevaan alasveto-listaan, josta päivystäjä pystyy valitsemaan esiasennon. Valinnan muutos lähettää pyynnön kamerapalvelimen kautta VMX-palvelimelle, joka kääntää kameraa valittuun esiasentoon. Ennen kuin kamerapalvelin välittää kääntöpyynnön, se varmistaa, että kamera on lukossa. Ensimmäisenä se siis lähettää erillisen pyynnön lukita kamera, jonka jälkeen tehdään itse kääntöpyyntö. Tässä kohtaa kahden erillisen pyynnön tekeminen VMX-palvelimelle ei oleellisesti näyttänyt vaikuttavan loppukäyttäjän käyttökokemukseen. Ainoa väliaikainen ongelma esiasentojen kanssa oli alkuun se, että esiasentojen ollessa merkkijonoja niiden merkistökoodaukset poikkesivat toisistaan, joten merkkijonojen vertailu ja esitys hajosivat joissakin tilanteissa. Tämä saatiin kuitenkin korjattua yhdenmukaistamalla merkistökoodaus UTF-8-tyyppiseksi.

5.3.3 Manuaaliohjaus hiiri- ja näppäimistösyötteillä

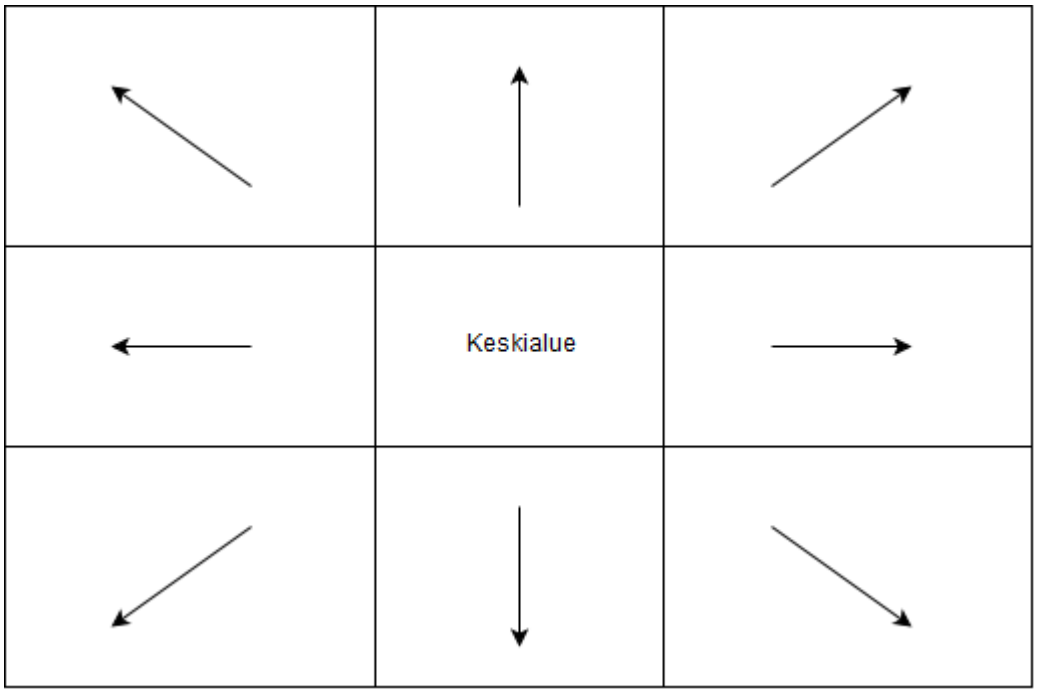
Kameran manuaalinen kääntäminen on esiasennoista poikkeavaa kameran kääntämistä, joka tapahtuu päivystäjän tekemien hiiren tai näppäimistön syötteiden välityksellä. VMX-palvelimen REST-rajapinta määrittelee muodon, jossa kääntämisen parametrit tulee välittää. Kääntäminen jaettiin kolmeen eri parametriin: pan, tilt ja zoom. Pan tarkoittaa kameran horisontaalista liikettä, toisin sanoen vertikaalisen akselin ympäri tapahtuvaa liikettä. Tilt taas kuvaa kameran pystysuuntaista liikettä. Zoom on kamerakuvan suurentamis- ja loitontamiskerroin. Alkuun parametrien arvot kuvasivat kameran kiertoa asteina.

Toteutus aloitettiin näppäimistökääntämisen toteuttamisella. WPF:ssä näppäimistön ja hiiren syötteistä laukeaa tapahtuma, joka välittää kyseisen syötteen parametreja. Kehittäjä voi sitoa parametrit vastaan ottavan tapahtumakäsittelijän XAML-koodissa kuvattuun UI-elementin tapahtumiin. Näppäimistön näppäintä painettaessa ja sen irti päästämisestä laukeavat KeyUp- ja KeyDown-tapahtumat, joihin voidaan sitoa käsittelijät eli metodit, jotka ottavat tapahtuman parametrit vastaan eli tässä tapauksessa WPF:n määrittelemän KeyEventArgs-tyyppisen olion. KeyEventArgs-olio sisältää tiedon, mikä näppäin on kyseessä ja onko se painettuna vai vapautettuna. Jos kyseessä olisi nuolinäppäin, syöte käsiteltäisiin kääntönä. Zoom-arvoa muokataan plus- tai miinus-näppäimillä.

Näppäinkääntöä aloitettiin tekemään siltä pohjalta, että näppäinkomennot lähettäisivät asteita kuvaavina arvoina kamerapalvelimelle, joka välittäisi ne VMX-palvelimelle REST-rajapinnan yli. VMX-palvelimen rajapinta kuitenkin muuttui kesken toteutuksen asteiden käytöstä nopeuteen, jolloin kääntämistä oli mietittävä uudelleen. Asteiden hylkääminen ja kääntönopeuden käyttöönotto johtui pitkälti siitä, että VMX-kamerajärjestelmän suorittama kääntö ei ollut tarpeeksi luotettava asteita käytettäessä. Kääntönopeuksien käyttö todettiin varmemmaksi ratkaisuksi, joten rajapinta muutettiin sen mukaiseksi.

Näppäin-kääntö toteutettiin loppuen lopuksi niin, että WPF:n KeyDown-tapahtuman käsittelijä kartoittaa, mikä näppäin on kyseessä ja muuttaa sen oikeaksi pan-, tilt- tai zoom-parametriksi. Lopuksi käsittelijä välittää parametrin ohjauskomponentille (kuva 6 sivulla 34), joka muodostaa REST-rajapinnan kuvaaman tietorakenteen, johon asetetaan pan-, tilt-, ja zoom-arvot tapahtumakäsittelijän välittämien parametrien mukaan. Oikeiden arvojen lisäämisen jälkeen kyseinen tietorakenne lähetetään ensin kamerapalvelimelle ja edelleen VMX-palvelimelle. Koska arvot kuvasivat nyt käännön nopeutta, oli näppäimen lähetettävä irti päästettäessä erillinen pysäytyspyyntö, muuten kamera kääntyisi loputtomiin viimeisimmän kääntöpyynnön mukaan. KeyUp-tapahtuman käsittelijä toteutettiin nollaamaan kyseessä olevan näppäimen kääntönopeus, joka välitettiin samaa väylää pitkin VMX-palvelimelle. Nopeuksien käyttäminen ei siis näppäimistökäännön yhteydessä ollut ongelma.

Hiirellä kääntäminen toteutettiin toimimaan samalla tavalla kuin VMX Client. Käyttäjä painaa jostakin kohtaa kameranäkymää ja suunta lasketaan kaksiulotteisena vektorina näkymän keskipisteestä painalluskohtaan. Vektorin x- ja y-arvot kartoitetaan pan- ja tilt-arvoiksi, jotka lähetetään eteenpäin. Vektorin pituus määrittää kääntönopeuden lopullisen arvon eli painamiskohdan ollessa näkymän reunalla kamera kääntyy nopeammin. Lisäksi näkymän keskellä on pieni kynnysalue (kuva 7), jonka alueella olevat hiirisyötteet eivät käännä kameraa, jolloin ohjaaminen helpompaa. Hiiren napin pysyessä pohjassa hiiren liikettä luetaan ja kääntöparametrien arvoja muutetaan sen mukaa. Tällä tapaa nopeutta ja suuntaa voidaan liikkeiden avulla muuttaa suoraan.



Kuva 7: Kameranäkymän jako kääntöalueisiin

WPF tarjoaa mahdollisuuden käsitellä myös MousePress-, MouseRelease- ja MouseMove-tapahtumilla. Samaan tapaan kuin näppäimistöohjauksessa, MousePress-tapahtuma aloittaa käännön laskemalla edellä mainitun vektorin suunnan ja pituuden sekä välittämällä tämän tiedon ohjauskomponentille. Vektori voidaan laskea hakemalla ensin näkymän keskipiste puolittamalla elementin pituus ja leveys. Klikkauskohdan koordinaatit saadaan taas MouseEventArgs-tyyppisen parametrin tiedoista, jonka koordinaatit on suhteutettu näkymän koordinaatistoon. Kummatkin arvot ovat tässä kohtaa elementin koordinaattiavaruudessa, joten niitä voidaan käyttää sellaisenaan. Lopullinen suuntavektori saadaan vähentämällä klikkauskohdan koordinaateista näkymän keskipisteen koordinaatit.

Kun MouseRelease-tapahtuma saadaan, lähetetään pysäytyskomento samaan tapaan kuin KeyUp-tapahtuman yhteydessä. Hiiren liikuttaminen hiiren näppäin pohjassa käsitellään MouseMove-tapahtuman käsittelijässä. MouseMove-tapahtuman käsittelijässä luetaan hiiren kohdistimen sijainti näkymässä uudelleen ja lasketaan uusi vektori. Ongelmana on,

että MouseMove-tapahtuma laukeaa aina, kun kohdistinta siirretään pikselinkin verran pois nykyisestä sijainnistaan. Tämä johtaa siihen, että nopeasta ja pitkästä hiiren liikkeestä laukeaa lyhyen ajan sisään useita tapahtumia. Jokaisen pikselipoikkeaman lähettäminen kamerapalvelimelle REST-rajapinnan kautta ei ole järkevää, sillä se aiheuttaa paljon pyyntöjen käsittelyä niin kameratyökalussa, kamerapalvelimella ja VMX-palvelimella. Ratkaisuna tähän hiiren liikkeestä syntyvien tapahtumien käsittelyä suodatettiin ajallisesti niin, että monesta peräkkäisestä lyhyen ajan sisään syntyvästä tapahtumasta vain viimeisin tapahtuma käsitellään loppuun. Teknisestä näkökulmasta kyseinen toiminto saatiin aikaan käyttämällä .NET-sovelluskehykselle suunnattua Rx.NET-kirjastoa, joka toteuttaa ohjelmistosuunnittelumallin, jossa tapahtuman tarkkailija tiedottaa tapahtuman tilaajille muutoksista. Rx.NET tarjoaa joukon erilaisia operaatioita, joita tässä tarkkailija-tilaaja-mallissa voidaan hyödyntää. Yksi näistä on Throttle-operaatio, joka ottaa syötteenään aikaikkunan, jonka sisällä tapahtuvista muutoksista vain viimeisin välitetään eteenpäin tilaajille ajan päätyttyä. MouseMove-tapahtumassa hyödynnettiin tätä operaatiota lisäämällä tapahtumille aikaikkunaksi 20 millisekuntia, jolloin lähteviä REST-pyyntöjä saatiin vähennettyjä. Tämäntapainen suodatus ei ole lopputuloksen kannalta ongelma, sillä muutaman pikselin muutos ei kameran liikkeeseen vaikuta juuri ollenkaan.

Toiminnallisissa vaatimuksissa kamerakuvan suurentaminen ja loitontaminen oli spesifioitu käytettäväksi hiiren rullan avulla plus- ja miinus-näppäinten lisäksi. Tämä oli varsin haastava toiminto toteuttaa sen jälkeen, kun VMX-palvelimen rajapinta muuttui asteista nopeuksiin. WPF tarjoaa tavan käsitellä rullan liikkeestä syntyviä tapahtumia samaan tapaan kuin muita edellä mainittuja tapahtumia. Ongelmaksi muodostuu kuitenkin se, että käsittelijään saapuva WPF:n määrittelemä MouseWheelEventArgs-tyyppinen parametri sisältää vain rullan deltamääräisen liikkeen. Yksi rullan naksahdus luo uuden tapahtuman, jonka MouseWheelEventArgs-olio sisältää kokonaislukutyyppisen Delta-kentän, joka kertoo ainoastaan mihin suuntaan rullaan on siirretty. Samankaltaista operaation lopetustapahtumaa, kuten näppäimistön KeyUp tai hiiren MouseRelease, ei siis ole olemassa. Tällöin ei siis tiedetä, milloin rullaaminen on lopetettu eikä liikkeen lopetuskutsua voida lähettää. Ainoa tapa ratkaista tämä puute oli asettaa jokin kiinteä aika. VMX-palvelimen rajapintaan oli suunnitteilla time-to-live-arvon lisääminen lähetettävään kääntöparametrien tietorakenteeseen, jonka arvo määrittäisi käännön keston. Tätä ei kuitenkaan ollut toteutushetkellä saatavilla vielä, joten kameratyökalun vastuulle jäi lähettää itse pysäytyskutsu tietyn ajan jälkeen. Ohjauskomponenttiin lisättiin ajastin, joka lähettää sekunnin päästä viimeisestä rullan siirrosta zoomin pysäytyspyynnön. Ratkaisu ei ole kovin optimaalinen, mutta tekee tehtävänsä väliaikaisena ratkaisuna, kunnes tarvittava rajapintamuutos käännön kestolle tulee saatavilla.

5.3.4 Kameraohjaamisen responsiivisuus

Manuaalisen kameraohjaamisen jälkeen huomattiin nopeasti, että esimerkiksi hiirellä kääntäminen oli erittäin hidasta. Hiirellä tehty kääntö näkyi kameratyökalun videokuvassa vasta 2-3 sekunnin kuluessa käännöstä. Toisin sanoen kameran kääntäminen kameratyökalusta käsin oli loppukäyttäjän kannalta varsin vaivalloista, ellei jopa käyttökelvotonta. Joitakin muutoksia oli tarve tehdä, jotta integraation ei-toiminnalliset vaatimukset saataisiin tältä osin täytettyä.

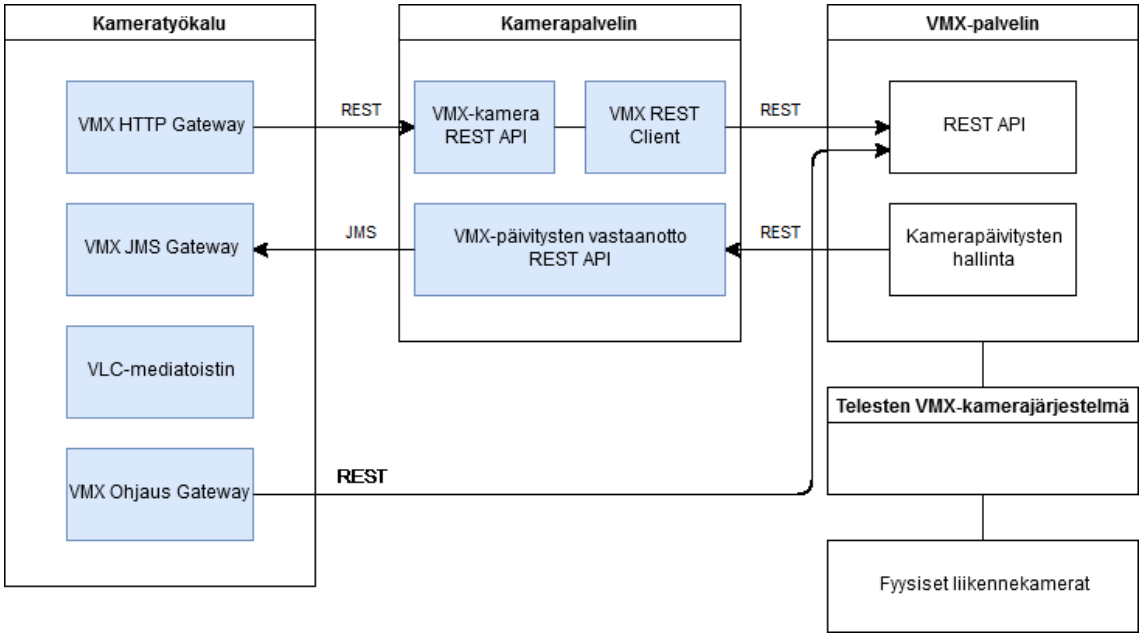
VMX-kameraintegraation arkkitehtuurimallia muutettiin pudottamalla kamerapalvelin pois kameraohjauksesta. Sen sijaan, että ohjauspyynnöt lähetettiin ensin kamerapalvelimelle, josta edelleen VMX-palvelimelle, kameratyökalu lähettäisi pyynnöt suoraan VMX-palvelimelle. Tämä muutos rikkoi alkuperäistä arkkitehtuurimallia, jossa kaikki liikenne kulkisi kamerapalvelimen kautta, jolloin VMX-palvelin rajapintoineen pysyisi tuntemattomana kameratyökalulle. Kameraohjaamisen käytön kannalta muutos oli kuitenkin välttämätön. Kameratyökaluun toteutettiin vielä yksi kommunikointikomponentti VMX Ohjaus Gateway. Kyseiselle komponentille paljastettiin VMX-palvelimen rajapintakuvaus, jonka avulla kääntöparametrit sarjallistettiin suoraan kameratyökalussa VMX-palvelimen vaatimaan muotoon. Eniten työtä aiheutti se, että aiemmin kaikki REST-pyynnöt olivat kameratyökalussa sarjallistettu XML-muotoon, jota kamerapalvelin pääasiassa käyttää. VMX-palvelin taas käyttää JSON-muotoa HTTP-viestien sisällön kuvaamiseen. Suoraohjaus kameratyökalusta VMX-palvelimelle aiheutti siis tarpeen tuoda uutta toiminnallisuutta myös HTTP-viestien sisällön sarjallistamiseen. Muulta osin ohjauksen kääntäminen suoraan VMX-palvelinta vasten oli ongelmaton.

Lopputuloksena kameran kääntäminen näkyi videokuvassa alle sekunnin viiveellä. Pyyntö välittyi VMX-palvelimelle nopeasti, sillä nyt VMX-palvelimen ja kameratyökalun välillä ei ollut eri verkossa sijaitsevaa kamerapalvelinta tai palomuureja hidastamassa liikennettä. Käännön ja videokuvan välillä pystyi tosin vielä tämän muutoksen jälkeenkin havaitsemaan pienen viiveen, johtuen siitä, että videokuva ei ole täysin reaaliaikaista kameratyökalun ja suoratoistopalvelimen välillä. Suurin syy tähän viiveeseen on se, että VLC-mediatoistimen on puskuroitava dataa noin minimissään 120 millisekunnin verran, ennen kuin videokuvaa esitetään käyttöliittymässä. Jos puskuria pienennetään, videokuva alkaa tökkimään tai pysähtyy kokonaan. Viive oli kuitenkin sen verran pieni, että kameran ohjaustoiminnon käytön kannalta se ei enää ollut ongelma.

5.4 VMX-kameraintegraation toteutuksen arviointi

VMX-kameraintegraatio saatiin lopulta päätökseen vaatimusten ja reunaehtojen puitteissa. Alkuperäisestä suunnitelmasta erottiin matkan varrella hieman, mutta vain kameroiden ohjaamisen osalta. Kameran ohjaaminen jouduttiin viemään käyttöliittymään ja näin ollen VMX-palvelimen REST-rajapinnan käyttö piti levittää kamerapalvelimen ulkopuolelle. Kuva 8 esittää entiteettien suhdetta VMX-kameraintegraation jälkeen, jossa

kameroiden ohjaus suoraan käyttöliittymästä on toteutettu. Tähän oli kuitenkin varauduttu jo suunnitteluvaiheessa.



Kuva 8: *Entiteettikaavio VMX-kameraintegraation toteutuksesta*

Muita suunnitelmasta poikkeavia muutoksia kameroiden ohjaamisen toteuttaminen ei tehty. Kameran ohjaaminen REST-rajapinnan kautta osoittautui yllättävänkin toimivaksi ratkaisuksi. Alun epäily HTTP-protokollan käytöstä kameran ohjaamiseen osoittautui tarpeettomaksi lopputuloksen kannalta.

Videon suoratoiston esittämiseen liittyen kameratyökalu muuttui muistinkäytön kannalta melko raskaaksi sovellukseksi. Ennen integraatiota kameratyökalu käytti muistia muutamia satoja megatavuja, nyt muistinkäyttö voi nousta jopa yli gigatavuun. Syynä tähän on se, että jokaiselle suoratoistolähteelle on luotava oma VLC-mediatoistin, joka käyttää muistia suhteellisen paljon.

Jälkeenpäin ajateltuna Meta.VLC:n tarjoaman WPF-elementin taustalle avattuna VLC-mediatoistin olisi voitu tuoda kirjaston rajapintaan, jotta sitä olisi voitu käsitellä erillään itse käyttöliittymäelementistä. Suoratoiston toteutus olisi näin ollen ollut hieman siistimpi ja käyttöliittymäelementin alustusta ei olisi tarvinnut tuoda näkymämalliin. Lopputuloksen kannalta se olisi vain miellyttänyt enemmänkin kehittäjää kuin varsinaista loppukäyttäjää.

Testaamisen osalta VMX-kameraintegraation toteutuksen virheettömän toiminnan varmentaminen oli varsin haastavaa johtuen siitä, että varsinaista testiympäristöä ei ollut. Toteutuksen testaamiseen oli varattu yksi suljetun tieosuuden kamera, joka oli kytketty samaan tuotantoympäristön järjestelmään kuin kaikki muutkin liikennekamerat. Testaaminen oli rajattu tähän kyseiseen kameraan sen takia, ettei tieliikennepäivystäjien työtä

häirittäisi ylimääräisillä kameroiden kääntelyillä. Haittapuoleksi muodostui se, että liikennekamerat voivat olla eri mallia, jolloin ne voivat käyttäytyä eri tavalla. Esimerkiksi, jotkut tietynmalliset kamerat värisevät hieman, jolloin niiden asentojen arvot (pan, tilt ja zoom) muuttuvat jatkuvasti. Tämä johtaa siihen, että ne lähettävät suuntapäivityksiä jatkuvasti kamerapalvelimelle. Ilman kattavampaa testiympäristöä tällaisten yllättävien seikkojen havaitseminen ja niihin varautuminen on mahdotonta. Tuotantoympäristön käyttö muutenkin kehitys- ja testialustana on vastoin ohjelmistotuotannon yleisiä käytäntöjä, mutta muita vaihtoehtoja ei ollut.

VMX-kameraintegraation toteutuksen kokonaisuutta arvioitaessa suunnitellut toiminnallisuudet toimivat suunnitellulla tavalla ei-toiminnallisten vaatimusten ja reunaehtojen puitteissa. Kameratyökalu ei käytä usean suoratoiston esitykseen liikaa työaseman resursseja ja virhesietoisuutta on lisätty esimerkiksi suoratoiston tilan monitoroinnilla ja tarvittavilla uudelleenyrityksillä virhetilanteen sattuessa. Myös kaikkien spesifioitujen koodekkien (MPEG-4 Part 2, Part 10 ja MJPEG) avulla koodattujen suoratoistojen dekoodaus on tuettu kameratyökalussa. Lisäksi kameroiden kääntäminen ja kuvan suurentaminen toimii sekä hiiren että näppäimistön avulla. Lopputulos toimii kaikin puolin suunnitelulla tavalla. Tämä johtopäätös pohjautuu kuitenkin varsin rajattuun testaukseen, kuten edellisessä kappaleessa on selostettu, joten kokonaisuuden luotettavuuden arviointi ilman laajempaa käyttöä on vaikeaa.

VMX-kameraintegraatio on kuitenkin valmis vain tähän työhön rajatulta osalta. Integraatiota on tarkoitus kehittää vielä pitkään. Kameratyökaluun on suunniteltu lisättävän muitakin suoratoistoon liittyviä toiminnallisuuksia, kuten tallennettujen suoratoistojen katselu. Tieliikennepäivystäjän on visioitu pystyvän katselemaan tallenteita samaan tapaan kuin normaalia videota katseltaessa eli tallennetta voisi muun muassa kelata edestakaisin ja jopa leikata pienempiin pätkiin. Näitä ominaisuuksia täytyy kuitenkin vielä suunnitella tarkemmin ja selvittää niihin liittyviä ongelmia.

6. YHTEENVETO

Tämän diplomityön tavoitteena oli toteuttaa liikennekameroiden keskitetyn VMX-kamerajärjestelmän integraatio liikennekeskuksissa käytössä olevaan T-LOIK-liikenneohjausjärjestelmään. Tarkoituksena oli tuoda liikennekameroiden tuottama videokuva VMX-kamerajärjestelmästä tieliikennepäivystäjien käyttämän kameratyökalun esitettäväksi. Lisäksi tarvittiin keino ohjata liikennekameroita, jotta kameralla voitiin kuvata moneen eri suuntaan. Työ kuvaa näiden kahden toiminnallisuuden toteutusprosessia.

Videodatan lähettämistä verkon yli lähettäjältä vastaanottajalle tavalla, jossa vastaanottaja voi katsoa videokuvaa sitä mukaa, kun dataa saapuu, kutsutaan suoratoistoksi. Suoratoiston etuna on se, että vastaanottajan ei tarvitse odottaa, että kaikki data on vastaanotettu ennen kuin sitä voi jo käsitellä. Haasteeksi tässä muodostuu datan lähettäminen tarpeeksi nopeasti, jotta videokuvaa voidaan esittää vastaanottajan päässä sujuvasti. Lisäksi videokuvan laatu muuttuu riippuvaiseksi verkon laadusta. Videodatan lähettämistä nopeutetaan pakkaamalla dataa lähettäjän päässä kooderin avulla. Tämän jälkeen pakattu data siirretään verkon yli vastaanottajalle, joka purkaa datan dekooderin avulla ja yrittää rakentaa mahdollisimman tarkan arvion alkuperäisestä kuvasta. Kooderin ja dekooderin muodostamasta paristä käytetään nimitystä koodekki. Videon pakkaamisesta on olemassa useita eri standardeja ja formaatteja, joista VMX-kameraintegraation kannalta olennaisimmat ovat MPEG-4 Part 2, MPEG-4 Part 10 ja MJPEG.

Kameraintegraatio suunniteltiin arkkitehtuuriltaan sellaiseksi, että kamerapalvelin olisi T-LOIK-järjestelmän ainoa osa, joka kommunikoisi VMX-kamerajärjestelmän päälle toteutetun VMX-palvelimen kanssa yhden REST-rajapinnan kautta. Kamerapalvelin kyselisi tarvittavat suoratoistoon liittyvät tiedot VMX-palvelimelta ja välittäisi nämä tiedot kameratyökalulle, joka taas vastaisi suoratoiston avaamisesta ja videokuvan esittämisestä. Videokuvan esittämiseen kokeiltiin prototyyppien avulla useita erilaisia videon toistoon liittyviä kirjastoja, joista valittiin lopulta VLC-mediatoistinta käyttävä Meta.VLC-kirjasto. Lisäksi suunniteltiin, että kameratyökalu pystyisi tieliikennepäivystäjän syötteiden avulla lähettämään kameroille ohjauspyyntöjä REST-rajapinnan kautta, joilla kuvattavaa suuntaa voitiin hallita.

Videon suoratoisto toteutettiin Meta.VLC-kirjaston avulla, mutta kirjasto vaati varsin epäjohdonmukaisen lähestymistavan itse implementaatioon. Videokuvaa esittävää WPF-käyttöliittymäelementtiä jouduttiin alustamaan näkymästä erillään näkymämallin puolella.

Kameran ohjaaminen toteutettiin lopulta siirtämällä ohjauspyyntöjen lähetys kamerapalvelimelta suoraan kameratyökaluun. Syynä tähän oli se, että kierrätettäessä pyyntöjä kamerapalvelimen kautta ohjaamisessa oli havaittavissa turhan paljon viivettä. Pyyntöjen

lähetys kameratyökalusta suoraan VMX-palvelimelle nopeutti ohjaamista siinä määrin, että kyseisessä ratkaisussa pysyttiin.

VMX-kamerajärjestelmän integraatio TLOIK-liikenneohjausjärjestelmään saatiin lopulta valmiiksi. Lopputulos täytti suunnitellut toiminnalliset vaatimukset käyttöympäristön asettamien rajoitteiden puitteissa. Usea samanaikainen, suoratoistettu video saatiin näkymään kameratyökalussa ja kameroita voitiin kääntää käyttäjän antamien syötteiden mukaisesti.

LÄHTEET

[1] .NET Framework Guide, verkkosivu. Saatavissa (viitattu 28.12.2017): https://docs.microsoft.com/en-us/dotnet/opbuildpdf/toc.pdf.

[2] Getting started with C#, verkkosivu. Saatavissa (viitattu 28.12.2017): https://docs.microsoft.com/en-us/dotnet/opbuildpdf/csharp/getting-started/toc.pdf.

[3] Getting started with WPF, verkkosivu. Saatavissa (viitattu 28.12.2017): https://docs.microsoft.com/en-us/visualstudio/opbuildpdf/designers/TOC.pdf?branch=live.

[4] Data Binding Overview, verkkosivu. Saatavissa (viitattu 28.12.2018): https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview.

[5] The MVVM Pattern, verkkosivu. Saatavissa (viitattu 29.12.2017): https://msdn.microsoft.com/en-us/library/hh848246.aspx.

[6] Flux, verkkosivu. Saatavissa (viitattu 5.5.2018): http://www.flir.co.uk/traffic/display/?id=61996.

[7] H. Lee, C. Tseng, A software framework for Java message service based Internet messaging system, Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, IEEE, pp. 161-165.

[8] S. Tiainen, Streaming tester software for mobile systems, Tampere University of Technology, 2004, 66 s.

[9] I.E.G. Richardson, H.264 and MPEG-4 Video compression : video coding for next-generation multimedia, Wiley, Chichester, 2003, 281 s.

[10] What are the different profiles supported by MPEG-4 Video? verkkosivu. Saatavissa (viitattu 14.01.2018): https://mpeg.chiariglione.org/faq/what-are-different-profiles-supported-mpeg-4-video.

[11] E. Schonfeld, H.264 Already Won, TechCrunch, Vol. 2010, 2010, Saatavissa (viitattu 25.01.2018): https://techcrunch.com/2010/05/01/h-264-66-percent-web-video/.

[12] International Telecommunication Union JCT-VC - Joint Collaborative Team on Video Coding, verkkosivu. Saatavissa (viitattu 20.01.2017): https://www.itu.int/en/ITU-T/studygroups/2017-2020/16/Pages/video/jctvc.aspx.

[13] L. Chen, N. Shashidhar, Q. Liu, Scalable Secure MJPEG Video Streaming, 2012 26th International Conference on Advanced Information Networking and Applications Workshops, pp. 111-115.

[14] S. Kamath, J. R. Jackson, Low-bit rate motion JPEG using differential encoding, Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004., pp. 1726 Vol.2.

[15] Internet Engineering Task Force, (IETF) Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, verkkosivu. Saatavissa (viitattu 27.01.2018): https://tools.ietf.org/html/rfc7230.

[16] C. Liu, Rate adaptation in media streaming, Tampere University of Technology, 2013, 50 s. Saatavissa: https://tut.finna.fi/Record/tutcat.235509.

[17] IETF SDP: Session Description Protocol
, verkkosivu. Saatavissa (viitattu 27.01.2018): https://tools.ietf.org/html/rfc2327.

[18] I. Haikala, T. Mikkonen, I. Haikala, Ohjelmistotuotannon käytännöt, 12. uud. p. ed. Talentum, Helsinki, 2011, 242 s.

[19] J. Paakki Ohjelmistojen vaatimusmäärittely, verkkosivu. Saatavissa (viitattu 04.01.2018): https://www.cs.helsinki.fi/u/paakki/Vaatimus-11-Luentokalvot-1.pdf.

[20] Java Virtual Machine Spesification, verkkosivu. Saatavissa (viitattu 04.01.2018): https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html.

[21] D. Crockford The application/json Media Type for JavaScript Object Notation (JSON), verkkosivu. Saatavissa (viitattu 04.01.2018): http://www.ietf.org/rfc/rfc4627.txt.

[22] Introduction to DirectShow, verkkosivu. Saatavissa (viitattu 05.01.2018): https://msdn.microsoft.com/en-us/library/windows/desktop/dd390351(v=vs.85).aspx.

[23] WPF Architecture, verkkosivu. Saatavissa (viitattu 06.01.2018): https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/wpf-architecture.

[24] Graphics and Drawing in Windows Forms, verkkosivu. Saatavissa (viitattu 06.01.2018): https://docs.microsoft.com/en-us/dotnet/framework/winforms/advanced/graphics-and-drawing-in-windows-forms.

[25] Technology Regions Overview, verkkosivu. Saatavissa (viitattu 06.01.2018): https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/technology-regions-overview.