jamk.fi

Audiostreamien puskurointi ja tallennus verkkoympäristössä

Janne Hakala

Opinnäytetyö
Syyskuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Hakala, Janne	Opinnäytetyö, AMK	Syyskuu 2017
	Sivumäärä 68	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Audiostreamien puskurointi ja tallennus verkkoympäristössä

Tutkinto-ohjelma

Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)

Ari Rantala

Toimeksiantaja(t)

Combitech Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa Combitech Oy:lle opinnäytetyön aiheesta toimiva prototyyppiratkaisu, josta olisi mahdollisesti hyötyä tulevissa toimeksiantajan asiakasprojekteissa. Tehtävänä oli toteuttaa audiotallennin, joka pystyisi luomaan audiopuskureita, jotka pystyisivät vastaanottamaan verkossa olevilta audiolähteiltä audiovirtaa. Audiopuskureiden tehtävänä oli puskuroida vastaanotettu audiovirta, normalisoida eri audiolähteiden mahdollinen aikareferenssiero sekä tallentaa puskuroidut audiovirrat kiintolevylle.

Opinnäytetyöprojekti toteutettiin käyttäen ketterän kehityksen menetelmää Scrum. Projekti tehtiin sprinteittäin aluksi viikon mittaisina ja myöhemmin kahden viikon sprintteinä. Sprintit suunniteltiin käyttämällä projektinhallintatyökalua Jira. Sprintin päätteeksi pidettiin sprinttikatselmointi, jossa kerrattiin toimeksiantajan muille työntekijöille opinnäytetyön eteneminen ja toiminnallisuudet, jotka oli saatu valmiiksi sprintin aikana.

Audiotallennin toteutettiin käyttäen Qt Creator -ohjelmointiympäristöä ja C++-ohjelmointikieltä. Suunnitteluvaiheessa määritettiin kehykset sille, mitä toiminnallisuuksia audiotallentimelle oli tarkoitus toteuttaa. Aluksi audiotallentimelle toteutettiin ohjelmistorunko sekä tarvittavat ohjelmistoluokat datan vastaanottamista ja tallentamista varten. Tämän jälkeen keskityttiin ydinongelman ratkaisemiseen ja tutkimustyöhön siihen liittyen. Tuloksia varten muistipuskurin koon kehittymisestä tallennettiin metadataa tekstitiedostoon, josta voitiin valitulla Calc-taulukkolaskentaohjelmalla myöhemmin luoda havainnollistavat kuvaajat eri tapauksiin liittyen.

Opinnäytetyön lopputuloksena saatiin toteutettua toimiva prototyyppiratkaisu toimeksiantajaa varten, mikä vastasi vaatimusmäärittelyssä määriteltyjä vaatimuksia ja näin ollen ratkaisi tutkittavana olleen ongelman.

Avainsanat (asiasanat)

Audio, Aikareferenssi, C++, Puskurointi, Normalisointi, Näytteenottotaajuus, TCP, Qt Creator

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Hakala, Janne	Bachelor's thesis	September 2017
		Language of publication: Finnish
	Number of pages 68	Permission for web publication: x

Title of publication

Buffering and recording audio streams in network environment

Degree programme

Software Engineering

Supervisor(s)

Rantala Ari

Assigned by

Combitech Oy

Abstract

The goal of the thesis was to implement a working prototype of the thesis topic given by the Combitech Oy which would help the client in the future with potential future customer projects. The task was to create an audio recorder that could create audio buffers which could receive audio stream from the audio sources on the web. The audio buffer was to be able to buffer the audio, normalize the time reference difference between different audio sources and store the buffered audio streams on hard disk.

The thesis project was carried out with sprints used in agile methodologies, firstly with one-week sprints and later, with two-week sprints. The sprints were planned using the project management tool Jira. At the end of each sprint, a sprint review was held where the thesis progress and functionality made within the sprint was revealed to the other employees in the company.

The audio recorder was implemented with the integrated development environment Qt Creator with using the programming language C++. At the planning stage, the frames were defined for what functionalities were to be implemented for the audio recorder. At first, the basic frame for audio recorder and needed programming classes for receiving the data and storing it were programmed. After that, the focus was on solving the core problem of the thesis. To present the results it was necessary to save metadata about the memory buffer size in a text file and then with Calc-software it was possible to create the needed graphs from that metadata.

The result of the thesis was a working prototype solution for the client that met the requirements set in the requirements specification and thus, it solved the investigated problem.

Keywords/tags (subjects)

Audio, C++, Buffering, Normalization, Sampling rate, TCP, Time reference, Qt Creator

Miscellaneous

Sisältö

Kuviot

Sanasto

Abstract class

Abstrakti luokka on ohjelmistoluokka, jossa ei voi luoda funktiolle toiminnallisuutta, vaan se täytyy toteuttaa perityssä aliluokassa.

Agile software development

Agile software development eli ketterä ohjelmistokehitys tarkoittaa tapaa, jolla ohjelmistokehitystä toteutetaan. Tavoitteena on reagoida nopeasti tuleviin muutoksiin ja olla suorassa yhteydessä asiakkaaseen. Ketterään ohjelmistokehitykseen kuuluvat vahvana osana sprintit (ks. Sprint).

Audio

Audio on digitaalinen tallenne analogisesta äänisignaalista, joka voidaan tarvittaessa muuntaa takaisin ääneksi äänikortin avulla ja toistaa kaiuttimien kautta.

Audion normalisointi

Audion normalisoinnilla tarkoitetaan tässä kontekstissa vastaanotettujen datapakettien muuttamista siten, että muistipuskurin koko saadaan tasattua kohti haluttua arvoa.

Audiostream

Audiostream eli audiovirta on tietotekniikassa audion jatkuvaa lähettämistä Internetin kautta.

C++

C++ on olio-ohjelmointikieli, jonka avulla voidaan ohjelmoida laajassa mittakaavassa erilaisia ohjelmistoja aina sulautetuista järjestelmistä asiakas-palvelintyyppisiin ohjelmistoratkaisuihin.

Class

Class eli ohjelmistoluokka on olio-ohjelmoinnissa (ks. OOP) malli siitä, minkälaisia metodeita ja muuttujia kyseessä olevasta luokasta luodulla oliolla on käytössään.

Cross-platform

Cross-platform eli alustariippumattomuus tarkoittaa ohjelmistoa tai järjestelmää, joka toimii useilla eri alustoilla ja käyttöjärjestelmillä.

CSV-tiedostomuoto

CSV eli comma-separated-values on tiedostomuoto, jota käytetään yleisesti taulukkomuotoisen datan varastointiin. Tiedostossa olevat arvot erotetaan pilkuilla, jotta ne voidaan taulukkolaskentaohjelman avulla erotella.

Git

Git on hajautettu versionhallintajärjestelmä, joka on tarkoitettu lähdekoodin varastointi varten.

IDE

Integrated development environment eli ohjelmointiympäristö on ohjelmisto, jolla ohjelmoija toteuttaa ohjelmistoja.

INI-tiedostomuoto

INI-tiedostomuotoa käytetään konfiguraatiotietojen tallentamiseen tekstitiedostoon, jossa on yksinkertainen rakenne. Rakenteeseen kuuluu osa-alue, ominaisuus ja sen arvo.

OOP

Object-oriented programming eli olio-ohjelmointi on ohjelmistokehityksen malli, joka keskittyy ohjelmistoluokista (ks. Class) luotujen olioiden avulla toiminnallisuuden toteuttamiseen.

Product Owner

Product Owner eli tuoteomistaja on henkilö, joka on yhteydessä asiakkaan ja kehitystiimin välillä ja vastaa siitä, mitä ominaisuuksia tulevaan ohjelmistoon halutaan.

Sample Rate

Sample rate eli näytteenottotaajuus tarkoittaa näytteiden määrää audiossa sekunnin aikana. Mitä suurempi näytteenottotaajuus on, sitä tarkempi digitaalinen tallenne se on.

Scrum

Scrum on ketterän ohjelmistokehityksen (ks. Agile software development) toimintaprosessi, jolla kehitetään monimutkaisia ohjelmistoja tehokkaasti kehitystiimin sisällä. Scrumin perusperiaatteena on toimittaa toimiva ohjelmisto asiakkaalle vähitellen reagoiden nopeasti muuttuviin tilanteisiin.

Sprint

Sprintillä kuvataan Scrumissa (ks. Scrum) kehitysjaksoa, joka kestää tavallisesti 1-4 viikkoa, minkä aikana ohjelmistoon toteutetaan sprinttisuunnittelussa määritetyt seuraavat toiminnallisuudet.

Qt Creator

Qt Creator on ohjelmointiympäristö (ks. IDE), joka on alustariippumaton (ks. Cross-platform) sovelluskehitin, jolla ohjelmoijat voivat kehittää työpöytä - ja mobiilisovelluksia useille käyttöjärjestelmille.

XML

XML on kuvauskieli, jolla kuvataan dataa. XML-tiedostosta muodostuu rakenteeltaan syvenevä, jossa voi olla useita alatasoja. Tämä helpottaa datan jäsentelyä. XML-tiedostoon voidaan tallentaa esimerkiksi metatietoa.

1 Audio osana arkipäivää

Viimeisten vuosikymmenten aikana kehitys digitaalisessa tekniikassa on vaikuttanut merkittävästi ihmisten arkipäiväisiin toimintamalleihin ja rutiineihin. Nykyään peruskäyttäjällä on mahdollisuudet hankkia kuluttajahintaan hyvin monenlaisia teknisiä laitteita työ- ja viihdekäyttöön. Alati kehittyvä tekniikka tuo digitaaliset välineet lähemmäs ihmisten arkipäiväistä elämää. Tekniikka alkaakin olla yhä enemmän osana tavallisissa arkisissa asioissa.

Tekniikan kehittymisessä on paljon hyviä puolia. Tekniset laitteet auttavat ihmisiä jokapäiväisissä asioissa. Voidaan pitää yhteyttä henkilöön tuhansien kilometrien päähän, tarkistaa nopeasti onko lähikauppa vielä auki, saada tärkeät viestit rannekelloon, paikantaa kadonnut henkilö hätäpuhelun avulla ja kuulolaitteen avulla voidaan heikentynyttä kuuloa vahvistaa. Tämä lista on loputon, eikä ole tarkoituksenmukaistakaan luetella tässä kontekstissa kaikkea, minkä tekniikka on mahdollistanut. Tärkeää on kuitenkin havainnoida, että ääni on isossa roolissa teknisissä laitteissa. Tästä varmaan parhaana esimerkkinä ovat päivittäin ympäri maailmaa soitetut miljardit puhelut, joissa ääni on ratkaisevassa osassa.

Ääni ja audio ovat käsitteitä, jotka saatetaan puhekielessä helposti sekoittaa samaksi asiaksi. Ne ovat kuitenkin eri asioita, vaikka ovatkin vahvasti sidoksissa toisiinsa. Ääni tarkoittaa mekaanista aaltoliikettä, jonka ihminen kuuloaistinsa avulla muodostaa musiikiksi, puheeksi tai meluksi. Ääni ei etene tyhjiössä, vaan tarvitsee aina väliaineen edetäkseen. Olomuoto voi olla nestettä, kaasua tai kiinteää. Tavallisesti väliaine on ilma eli kaasu. Audiolla sen sijaan tarkoitetaan digitaaliseen eli tietokoneen ymmärtämään muotoon muutettua tallennetta ääniaalloista. (Difference between Sound and Audio n.d.)

Musiikkia suoratoistetaan ympäri maailmaa valtavia määriä. Nykypäivänä monet työpöytä- ja mobiilisovellukset mahdollistavat tämän erittäin helposti. Loppukäyttäjä vastaanottaa päätelaitteeseensa audiovirtaa Internetin kautta. Audiovirta muutetaan sen jälkeen digitaalisesta muodosta analogiseksi. Tämä toteutetaan käyttämällä DAC-konvertteria. DAC (digital-to-analog converter) muuttaa digitaalisen signaalin analogiseksi. Audio muutetaan aina analogiseksi signaaliksi ennen kuin se siirretään vahvistimelle, josta signaali johdetaan esimerkiksi kaiuttimille tai kuulokkeille. Tämän jälkeen

voidaan havainnoida ilmassa liikkuvat ääniaallot kuuloaistin avulla. (What is a DAC? n.d.) Tässä opinnäytetyössä ei ollut tarkoituksena tutkia sitä, miten DAC-konvertteri toimii, vaan keskittyä siihen, mitä tapahtuu ennen kuin audio päätyy DAC-konvertterille.

Audion siirrossa Internetin kautta voi tapahtua erilaisia ongelmia, jotka vastaanottavan päätelaitteen tulisi havainnoida ja reagoida näihin poikkeamiin tarkoituksenmukaisilla tavoilla. Saattaa olla esimerkiksi tilanteita, joissa Internet-yhteys on hidas ja toiminnassa esiintyy katkoksia tai jumiutumista. Tällöin audiota suoratoistavan sovelluksen tulisi huomioida mahdolliset yhteyskatkokset ja tilanteet, joissa audiota vastaanotetaan katkoksen jälkeen isompana massana.

Esimerkiksi nettiradioissa lähetetään suorana lähetyksenä audiovirtaa, jota päätelaitteet toistavat ihmisille. Näissä tilanteissa pitää huomioida verkon katkokset sen kannalta, mitä tehdään yhteyden palautuessa. Onko järkevää jatkaa lähetystä siitä kohdasta, mitä reaaliaikaisesti lähetetään vai toistaa katkoksen aikana lähetyksessä tapahtuneet asiat. Jälkimmäinen tapaus johtaa siis siihen, että vastaanottava päätelaite on jäljessä radiolähetystä.

Tässä opinnäytetyössä tutkittiin, miten audiovirtaa tulisi oikeaoppisesti käsitellä verkkoympäristössä. Tarkoituksena oli toteuttaa audiotallennin, joka vastaanottaa Internet-yhteyden kautta lähetettyä audiovirtaa. Vastaanotettua digitaalista audiota käsiteltiin eli puskuroitiin tietokoneen muistissa, ennen kuin se tallennettiin kiintolevylle myöhempää käyttöä varten. Päätavoitteena oli havainnoida lähetyksen aikana mahdollisesti tapahtuvia ongelmia ja poikkeustilanteita.

Audiolähteet lähettivät digitaalista audiovirtaa tietyillä näytteenottotaajuuksilla. Audiolähteillä voi olla hieman eri käsitys siitä, kuinka pitkä aika esimerkiksi sekunti on. Tämä tarkoitti käytännössä sitä, että vastaanottava sovellus oletti saavansa sekunnissa audiovirtaa tietyn verran, vaikka todellisuudessa sitä tulikin enemmän tai vähemmän. Toteutetun sovelluksen tuli reagoida tähän ongelmaan puskuroinnin yhteydessä ja korjata vastaanotettua audiota. Muutoin aika-akseli vääristyi, eikä varastoitu audio ollut enää luotettavaa kuluvaan aikaan nähden. Palattaessa tiettyyn ajankohtaan audiossa ei voitu enää varmasti sanoa, millä ajanhetkellä se oli oikeasti tapahtunut.

2 Työn lähtökohdat

2.1 Taustat

Alkuvuodesta 2017 toimeksiantaja Combitech Oy:n edustajat tulivat käymään Jyväskylän ammattikorkeakoululla rekrytoimassa opiskelijoita. Sen yhteydessä opinnäytetyöntekijä kävi keskustelemassa toimeksiantajan edustajien kanssa. Combitech Oy oli kiinnostunut siitä, että opinnäytetyön tekijällä opinnot olivat hyvässä vaiheessa, eikä opintoja kevään loppupuolella puuttuisi enää muuta kuin itse opinnäytetyö.

Opinnäytetyön tekijä oli yhteydessä toimeksiantajaan tapaamisen jälkeen, ja he sopivat uuden tapaamisajan, jolloin keskusteltiin mahdollisesta lopputyön aiheesta. Toimeksiantaja esitteli opinnäytetyön tekijälle muutamia opinnäytetöiden aiheita, joiden tutkimisesta ja ratkaisemisesta olisi tulevaisuudessa hyötyä yritykselle. Opinnäytetyön tekijä valitsi aiheen näiden aiheiden joukosta. Valittua aihetta oli toimeksiantajan taholta aiemminkin käsitelty ja pohdittu, mutta siitä ei oltu toteutettu mitään varsinaista ratkaisua.

2.2 Toimeksiantaja ja sidosryhmät

Opinnäytetyön toimeksiantajana toimi Combitech Oy, jonka tiloissa työtä alettiin toteuttaa huhtikuun alussa 2017. Combitech-konserni on erikoistunut tekniseen konsultointiin ja on itsenäinen yritys, mutta kuuluu osaksi puolustus- ja turvallisuusyritystä Saab AB:ta. Combitech-konsernin asiakkaat ovat pääsääntöisesti Ruotsissa, Suomessa ja Norjassa, mutta heillä on myös kansainvälisiä asiakkaita. Näissä pohjoismaissa Combitechilla on jo yli 1900 pätevää työntekijää. (About Us n.d.)

Suomessa toiminut Saab Ab:n omistama Saab Systems Oy tuli osaksi Combitech-konsernia vuoden 2014 alussa, tällöin myös nimi vaihdettiin Combitech Oy:ksi. Combitech Oy:llä on Suomessa neljä toimipistettä Espoossa, Tampereella, Jyväskylässä ja Säkylässä. Näissä toimipisteissä on yhteensä noin 80 työntekijää. Combitech Oy toimittaa puolustusvoimille teknisiä ratkaisuja ja palveluita, mutta myös yrityksille, teollisuudelle ja julkiselle sektorille. (Tietoja Combitechistä 2014.)

Opinnäytetyön ohjelmointityö toteutettiin käyttäen ketterän ohjelmistokehityksen eli Agile software developmentin menetelmää Scrum. Scrum-viitekehykseen perinteisesti kuuluvien kehitysjaksojen eli sprinttien hengessä ohjelmointityötä toteutettiin aluksi viikon mittaisina sprintteinä ja myöhemmin kahden viikon mittaisina sprintteinä. Combitechiltä opinnäytetyön ohjaajana toimi Timo Mustonen. Hän oli pääohjaajana koko ohjelmointityön ajan ja ohjasi työtä haluttuun suuntaan sekä toimi täten projektin tuoteomistajana eli Product Ownerin roolissa.

Sidosryhmänä opinnäytetyössä toimi opinnäytetyön tekijän koulu JAMK eli Jyväskylän ammattikorkeakoulu. JAMK vastasi opinnäytetyön ohjauksesta, tarkistuksesta, arvioinnista ja julkaisusta. Toisena sidosryhmänä voidaan mainita Combitech Oy:n Jyväskylän toimipisteen muut työntekijät. Jokaisen sprintin päätteeksi pidettiin yhteenvetona sprinttikatselmus, jossa esiteltiin muille työntekijöille sprintin tavoitteet, toteutetut toiminnallisuudet ja tulevat suunnitelmat ja työntekijöillä oli mahdollisuus esittää tarkentavia kysymyksiä.

2.3 Tavoitteet

2.3.1 Toimeksiantajalle

Toimeksiantaja kokeili heille uutta lähestymistapaa rekrytoinnissa, ja he päättivät tulla JAMKille rekrytoimaan opiskelijoita. Tavoitteena oli rekrytoida nuoria valmistumisen kynnyksellä olevia opiskelijoita. Combitech Oy antoi opinnäytetyön tekijälle opinnäytetyön aiheen, joka voi tulevaisuudessa liittyä epäsuorasti toimeksiantajan mahdolliseen asiakasprojektiin. Toimeksiantajan tavoitteena oli antaa tekijälle sellainen opinnäytetyön aihe, josta luotua prototyyppiä he voisivat tulevaisuudessa hyödyntää. Tavoitteena oli myös saada tekijälle opinnäytetyön ohella muita töitä, jotka liittyivät oikeaan asiakasprojektiin. Tämän avulla tekijä haluttiin saada jo aikaisessa vaiheessa mukaan yrityksen projektityöhön, jotta opinnäytetyön päätyttyä olisi helpompaa ja luontevampaa jatkaa projektitöissä.

2.3.2 Tekijälle

Tekijälle opinnäytetyön tavoitteet olivat korkealla. Tavoitteena oli asettaa opinnäytetyö muiden opintojen kanssa arvosanaltaan samaan linjaan eli ylimmät arvosanat olivat alusta alkaen kiikarissa. Ohjelmointityön tavoitteena oli oppia aiemmista virheistä, joita oli ollut aiemmilla opintojaksoilla eritoten harjoitustöiden parissa. Näissä suurimpana puutteena useimmiten oli havaittavissa hyvän suunnittelun puute. Harjoitustöissä käytännössä mentiin suoraan asiaan, alettiin toteuttaa ohjelmaa ja jätettiin suunnittelu täysin muistin varaisiksi ideoiksi.

Opinnäytetyön luonteenkin mukaisesti raportointi on suuressa roolissa, joten tämän osa-alueen parantaminen oli isoimpina tavoitteina jo aikaisessa vaiheessa. Raportointiosuudessa on ollut parannettavaa viitaten aikaisempien opintojaksojen harjoitustöihin. Tavoitteena oli oppia raportoimaan tehtyä työtä isommasta näkökulmasta, joka vähitellen pienenee kohti ratkaistua ongelmaa. Tämän tarkoituksena oli saada luotua ehjä raportti alusta loppuun saakka, jotta onnistuneesti ratkaistun tutkimuksen tulokset saadaan lukijalle raportoitua selkeästi ja johdonmukaisesti.

2.3.3 Muille tahoille

Muuna tahona tässä osiossa voidaan mainita opinnäytetyön tekijän oppilaitos eli JAMK. JAMKin tavoitteina opinnäytetöiden osalta on pääsääntöisesti se, että niissä tutkimuksen osa-alue on 100 %. JAMKin tavoitteena on se, että tutkittua aihetta oppilaitos voisi myöhemmin käyttää hyödyksi opetuksessa. Tämän takia opinnäytetöiden kaikki materiaali pyritään pitämään julkisena.

Tavoitteena on myös, että opinnäytetyön aihe täytyy olla riittävän laaja ja haastava, että siitä saadaan ammennettua oikeanlainen opinnäytetyö. Aihepiirin tulisi olla hyvin sidoksissa omaan alaan ja täten nimenomaan todistetaan vuosien aikana opitut taidot viimeisen työn muodossa.

3 Työn kuvaus

3.1 Yleistä

Verkossa on olemassa useita audiolähteitä, jotka tuottavat digitaalista audiovirtaa. Näistä audiolähteistä vastaanotettua audiovirtaa täytyi käsitellä ja havainnoida lähetyksessä mahdollisesti tapahtuvia poikkeamia, minkä jälkeen audio tuli tallentaa kiintolevylle myöhempää tarkastelua varten. Poikkeamia lähetyksessä olivat yhteyden katkeaminen ja yhteyden pätkiminen, eli audiovirtaa ei koko ajan vastaanotettu tasaisesti, vaan isompina purskeina. Nämä molemmat poikkeamat olivat osana tämän opinnäytetyön ydinongelman ratkaisussa eli eri lähteiden välisen aikareferenssieron hallinnassa.

Eri audiolähteiden välillä voi tavallisesti olla hyvin pieniä eroja siinä, kuinka nopeasti aika etenee niiden käsityksen mukaan, eli sekunti ei ole atomikellon tarkkuudella määritetty näissä audiolähteissä. Tätä aikareferenssieroa tuli siis hallinnoida vastaanottavassa päässä, koska ennen pitkää, jos kaksi lähdettä tuottaa samalla näytteenottotaajuudella audiovirtaa, niiden datamäärän koko tulee eroamaan kuluvan ajan funktiona. Audiolähteiden aikareferenssi tuli normalisoida yhteiseen aikareferenssiin, joka oli vastaanottavan tallentimen oma aikareferenssi. Tätä aikaa pidettiin tässä tapauksessa oikeana ja audiolähteiden audiovirta tarvittaessa normalisoitiin vastaamaan sitä.

Tallentimessa audiovirran normalisointi tuli ratkaista eri tapausten mukaisesti. Jos audiolähde lähetti dataa liikaa eli audiolähteen käsitys sekunnista oli pitempi kuin tallentimen, aikareferenssin korjaamiseksi audiovirtaa tuli muokata siten, että tämä ylimääräinen data saatiin poistettua. Mikäli tilanne oli toisinpäin eli dataa vastaanotettiin liian vähän, tällöin tallentimeen vastaanotetussa audiovirrassa nämä tyhjät kohdat tuli hallita, koska aika kului koko ajan eteenpäin.

Yhteenvetona voidaan todeta, että audiovirtaa lähetettiin tahallisesti enemmän, kuin tallentimen aikakäsitys oletti, jotta ongelma saatiin ilmaantumaan. Tämä ongelma tuli hallinnoida tallentimen muistissa, ennen kuin audiovirta tallennettiin kiintolevylle. Tästä muodostui opinnäytetyön tärkein osa-alue, ja muut rakennelmat tallentimessa olivat esityötä ydinongelman ratkaisun tukemiseksi.

3.2 Työn rajaus

Toimeksiantajan antaman aihekokonaisuuden koko olisi ollut opinnäytetyöksi liian iso, joten siitä rajattiin opinnäytetyön tekijälle kohtuullisen kokoinen osa-alue. Tämän opinnäytetyön osalta keskityttiin audion vastaanottamiseen verkosta, audion puskurointiin tallentimessa sekä sen tallentamiseen kiintolevylle. Näistä osa-alueista tärkeimpänä oli audion puskurointi tallentimen muistissa.

Toimeksiantajan tiloissa toteutettiin myös toinen opinnäytetyö. Nämä opinnäytetyöt liittyivät epäsuorasti toisiinsa, mutta toimivat kuitenkin omina kokonaisuuksinaan. Opinnäytetyöt toteutettiin pienenä projektina, jossa tuoteomistajana toimi toimeksiantajan taholta Timo Mustonen. Jokaisen sprintin jälkeen opinnäytetöiden tekijät esittelivät omat tuotoksensa toimeksiantajan tiloissa järjestetyssä yhteisessä katselmoinnissa. Tämän tavoitteena oli pitää Combitech Oy:n Jyväskylän toimipisteen muut työntekijät tietoisina opinnäytetyön etenemisen vaiheista sekä kartuttaa opinnäytetyön tekijälle kokemusta toteutetun työn esittelystä.

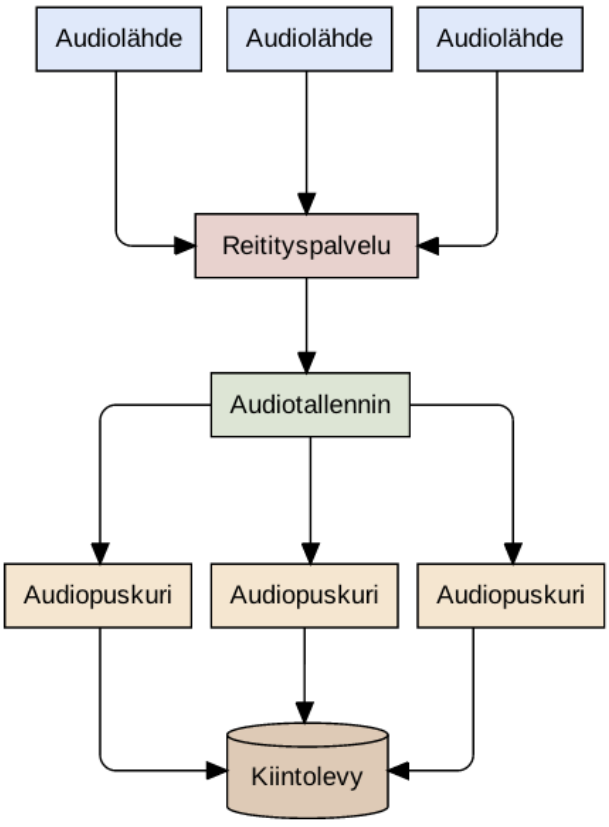
Työtä rajattiin myös siihen suuntaan, että Mustonen toteutti opinnäytetyön tekijän työn tueksi AudioSource-nimisen ohjelman C++-ohjelmointikielellä. Tämän ohjelman avulla voitiin lähettää audiovirtaa verkkoympäristössä tallentimelle. Tämä edesauttoi sitä, että opinnäytetyön tekijä pystyi keskittymään opinnäytetyön aiheeseen, eikä näin ollen tarvinnut toteuttaa aiheen ulkopuolella olevaa toiminnallisuutta. Tallentimen toiminnan testauksessa AudioSouce oli merkittävässä roolissa, koska se tarjosi audiovirran tallentimelle puskuroitavaksi.

Opinnäytetyössä oli tarkoituksena luoda tallentimen runko alusta saakka kohti haluttua päämäärää, joka parhaiten edesauttaisi varsinaisen ongelman ratkaisussa. Alkuvaiheessa toteutettiin audion vastaanottamista varten tarvittavat toiminnallisuudet sekä audion tallentamista. Tämän jälkeen voitiin kohdistaa pääpainopistettä audion käsittelyyn ja sen puskurointiin muistissa.

3.3 Työn käyttötarkoitus

Tässä projektissa toteutetut kaksi opinnäytetyötä liittyivät isompaan ohjelmistokokonaisuuteen. Opinnäytetöissä luotiin prototyyppiratkaisut eri osa-alueiden ongelmien ratkaisuun. Audiotallentimen luomien audiopuskurien tuli siis voida vastaanottaa verkossa olevilta lähteiltä audiovirtaa, normalisoida ja tallentaa se kiintolevylle. Audiopuskurin luonnin yhteydessä sille määritettiin audioformaatti, jota sen oli tarkoitus vastaanottaa.

Toisen opinnäytetyön aihepiirinä oli näiden audiovirtojen reitittäminen oikeanlaiselle audiotallentimen audiopuskurille. Kokonaiskuvana järjestelmän ideana on se, että reitityspalvelu tietää olemassa olevat audiolähteet ja niiden audioformaatit. Siihen tietoon perustuen reitityspalvelu käskee audiotallentimen luoda oikeanlainen audiopuskuri, johon kyseisen lähteen audio voidaan ohjata. Tämän jälkeen audiopuskuri hoitaa audion lopun käsittelyn. Nämä kaksi kokonaisuutta olivat siis erilliset aihealueet ja osaset tässä järjestelmässä, mutta yhdessä muodostivat isomman ohjelmistokokonaisuuden (ks. kuvio 1).



Kuvio 1. Järjestelmän kokonaiskuvaus

3.4 Vaatimusmäärittely

3.4.1 Yleistä

Vaatimusmäärittelyllä tarkoitetaan dokumenttia, joka kuvaa ja selventää toteutettavalle ohjelmistolle asetetut vaatimukset ja tavoitteet. Se on pääsääntöisesti aina osana ohjelmistoprojektin suunnitteluvaihetta. Vaatimusmäärittely kertoo lukijalle, mitä toiminnallisia sekä ei-toiminnallisia vaatimuksia suunnitellun järjestelmän tulee toteuttaa. Vaatimuksia on monenlaisia, eikä kaikki niistä ole varsinaista toiminnallisuutta, vaan ne voivat olla myös erilaisia rajoitteita ohjelmiston toiminnalle. (Taina 2010, 2-3.)

Vaatimusmäärittely on runko toteutettavan ohjelmiston suunnittelussa. Sen avulla pystytään selkeyttämään ajattelumallia siitä, mitä oikeasti halutaan toteuttaa ja mihin toteutettua lopputuotosta tulevaisuudessa tullaan käyttämään. Vaatimusmäärittely voidaan mieltää eräänlaiseksi muistilistaksi suunnitteluvaiheesta sen takia, että toteutusvaiheessa voidaan palata tarkistamaan ennalta määritetyt vaatimukset ja näin ollen voidaan paremmin ohjata työtä alun perin haluttuun suuntaan.

3.4.2 Johdanto

Tässä opinnäytetyössä tarkoituksena oli toteuttaa ohjelmisto, joka vastaa toimeksiantajan antamia tavoitteita ja vaatimuksia mahdollisimman hyvin. Tavoitteena oli, että toimeksiantaja voisi tulevaisuudessa hyödyntää opinnäytetyössä tuotettua ohjelmistoa prototyyppinä vastaavanlaisten ongelmien ratkaisemiseen mahdollisissa asiakasprojekteissa.

Toteutettavalla järjestelmällä ei ole varsinaista käyttäjää, joka voisi omalla toiminnallaan vaikuttaa audiotallentimen toimintaan. Ohjelmistolle ei siis toteuteta erillistä käyttöliittymää, koska se toimii itsenäisenä palveluna. Täten käyttäjätarinatkin ovat hieman erilaisia, koska ne ovat tapahtumia, jotka käyttäjä olettaa audiotallentimen hoitavan automaattisesti. Edempänä luvuissa 3.4.3-3.4.5 tullaan kuvaamaan, mitä vaatimuksia opinnäytetyössä toteutettavalla ohjelmistolla on.

3.4.3 Rakenteelliset vaatimukset

Audiotallentimen rakenne tulee olla alusta alkaen järkevästi suunniteltu, jotta sitä voidaan tulevaisuudessa helpommin hyödyntää muihin tarpeisiin. Järkevä rakenteen suunnittelu mahdollistaa myös sen, että ulkopuolinen henkilö pystyy paremmin sisäistämään ohjelman toiminnan. Tämä korostuu etenkin silloin, jos tulevaisuudessa toimeksiantajan taholta päätetään hyödyntää tässä opinnäytetyössä toteutettua ohjelmistoa.

Ohjelmiston eri toiminnallisuudet tulee olla pilkottuina selkeisiin ohjelmistoluokkiin ja -alaluokkiin. Luokkien tulee toteuttaa niille omaksutut ja tarkoitetut asiat, eikä niiden toteutus saa poiketa muiden luokkien vastuualueille. Luokkien nimet tulee olla kuvaavia sen suhteen, mitä toiminnallisuutta niiden on tarkoitus toteuttaa. Tallentimeen toteutetun rakenteen tulisi mahdollisimman hyvin auttaa opinnäytetyön ydinongelman ratkaisussa. Ensisijaisena vaatimuksena onkin saada toteutettua onnistunut ohjelmiston runko, jonka avulla on helpompi toteuttaa ratkaisut opinnäytetyössä tutkittavana oleviin ongelmiin.

3.4.4 Ei-toiminnalliset vaatimukset

Ei-toiminnallisia vaatimuksia on hankalampi tunnistaa kuin toiminnallisia vaatimuksia, mutta ne ovat silti tärkeässä roolissa toteutettavan ohjelmiston näkökulmasta. Ei-toiminnallisia vaatimuksia voivat muun muassa olla laatuvaatimukset, ohjeistusvaatimukset, arkkitehtuurivaatimukset ja kehitystyön vaatimukset. (Taina 2010, 27 ja 29.)

Kehitystyön vaatimuksia toteutettavalle ohjelmistolle on se, että työ tulee toteuttaa käyttäen ketterän ohjelmistokehityksen menetelmää Scrum. Scrumiin olennaisesti liittyvien sprinttien eli kehitysjaksojen aikana toteutetaan ohjelmointityötä. Ketterän kehityksen mukaisesti pyritään sprinttien aikana toteuttamaan toimiva ohjelmistokokonaisuus, reagoimaan haluttuihin muutoksiin nopeasti ja olemaan suorassa viestinnässä eri sidosryhmien kanssa.

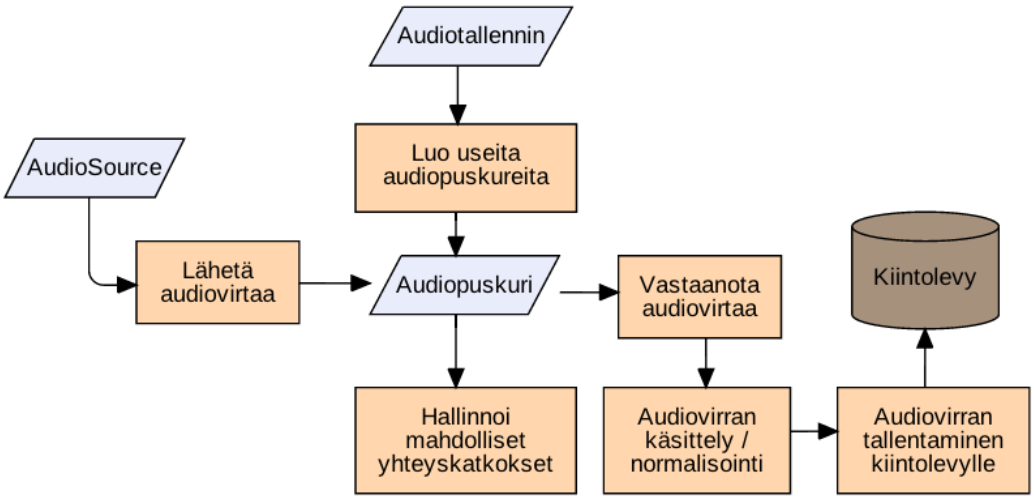
Laadullisia vaatimuksia ohjelmistossa on se, että toteutetaan järkeviä osakokonaisuuksia, joita on helppo tulevaisuudessa päivittää ja mahdollisesti integroida suoraan johonkin toiseen järjestelmään.

3.4.5 Toiminnalliset vaatimukset

Toiminnallisiksi vaatimuksiksi vaatimusmäärittelyssä määritellään toteutettavan ohjelmiston ominaisuudet ja toiminnot, jotka sen tulee hallita. ”Toiminnalliset vaatimukset liittyvät tulevan ohjelmiston tarjoamiin palveluihin” (Taina 2010, 25).

Toiminnallisia vaatimuksia audiotallentimen toiminnalle tulee määritettyä enemmän toteutusvaiheen aikana, mutta seuraavat toiminnallisuudet tulee olla vähintään toteutettuina (ks. kuvio 2). Audiotallentimen tulee pystyä vastaanottamaan AudioSource-ohjelman lähettämää audiovirtaa verkkoympäristössä, ja sen pitää pystyä tallentamaan vastaanotettu data kiintolevylle myöhempää tarkastelua varten. Audiotallentimen tulee pystyä luomaan useita audiopuskureita, jotka vastaanottavat audiovirtaa eri audiolähteiltä, joita tässä kontekstissa ovat AudioSouce-ohjelman avulla luodut audiovirrat.

Tallentimeen vastaanotettua audiota tulee pystyä käsittelemään, ennen kuin se tallennetaan levylle. Tallentimen tulee täten reagoida mahdollisiin aikareferenssieroihin eri audiolähteiden välillä ja mahdollisesti poistaa tai lisätä dataa tilanteiden mukaisesti ja näin ollen normalisoida vastaanotettu audio ennen tallentamista. Tallentimen tulee myös pystyä reagoimaan mahdollisiin verkkoyhteyden katkeamisiin sekä selvitä tilanteista, joissa yhteys on hidas tai jumiutunut, mutta palautuu äkillisesti normaalitilaan.



Kuvio 2. Toiminnalliset vaatimukset kuvattuina

4 Suunnittelu, työkalut ja teknologiavalinnat

4.1 Yleistä

Ohjelmistosuunnittelu on avainasemassa ohjelmistotuotannossa. Suunnittelu nousee esille etenkin, kun toteutetaan hiemankin isompaa kokonaisuutta. Jos ohjelmiston suunnitteluvaihe ohitetaan kokonaan ja aloitetaan suoraan kasata ensimmäisiä ohjelmistokoodin osasia, ennen pitkää huomataan, miten hankalaa sitä on muokata haluttuun suuntaan. Kun ensimmäinen isompi muutos on käsillä, alkaa ohjelmistokoodi muuntautua yhä hankalammin luettavaksi. Käytännössä näissä tilanteissa olisi parempi aloittaa koko työ alusta ja suunnitella kokonaisuus alusta alkaen järkevästi. (Valdarrama 2014.)

Ohjelmistokoodin laatutasoa ei kerro se, toimiiko toteutettu ohjelma tai saatiinko se onnistuneesti toimitettua asiakkaalle. Pikemminkin laatutasoa kuvastaa se, kuinka helppoa toteutetun ohjelmiston lähdekoodia on jälkikäteen ylläpitää, lukea ja muokata. (Valdarrama 2014.)

”Kuka tahansa voi kirjoittaa koodia, jota tietokone ymmärtää. Hyvät ohjelmoijat kirjoittavat koodia, jota ihminen voi ymmärtää.” (Valdarrama 2014.)

4.2 Suunnittelu

Suunnittelussa lähtökohtana oli pohtia teknologiavalintoja vaatimusmäärittelyn pohjalta olevien tietojen perusteella. Kun tietyt vaatimukset tulee toteuttaa, on teknologiavalintojenkin oltava sitä tukevia. Teknologiavalintoja ohjelmistoprojekteissa ovat yleisesti ohjelmointiympäristö, ohjelmointikieli, versionhallinta ja mahdolliset muut työn tekemistä tukevat ohjelmistot. Valitun ohjelmointikielen ja ohjelmointiympäristön tulisi olla hyvin sidoksissa toisiinsa, että työn toteuttaminen onnistuisi mahdollisimman suoraviivaisesti ilman ylimääräisiä sivuraiteita. Kun nämä asiat ovat kunnossa, on työn tekeminenkin huomattavasti mielekkäämpää.

”Suunnitteluun panostaminen asettaa mahdollisuudet onnistuneelle projektille ja takaa sijoitetun pääoman paremman tuoton” (Importance Of Planning In Software Development n.d.).

4.3 Ohjelmointiympäristöt

4.3.1 Yleistä

Ohjelmointiympäristöllä tarkoitetaan ohjelmaa, joka mahdollistaa sovelluskehityksen helpommalla tavalla kuin vielä muutamia vuosikymmeniä takaperin. Ohjelmointiympäristöt tarjoavat tavallisesti ohjelmoijille graafisen käyttöliittymän, johon he voivat aloittaa kirjoittamaan valitsemallaan ja ohjelmointiympäristön tukemalla ohjelmointikielellä ohjelmakoodia (Integrated Development Environment (IDE) n.d.).

Ohjelmointiympäristöistä on nykypäivänä olemassa hyvin moderneja versioita, joiden avulla esimerkiksi käyttöliittymän prototyyppi voidaan rakentaa ohjelmointiympäristössä olevaan näkymään pelkästään raahaamalla valmiita ohjelmistokomponentteja kuten tekstikenttiä ja painikkeita. On myös olemassa eri ohjelmointikielille olevia kääntäjiä, jotka kääntävät ihmisten ymmärtämän tekstimuodossa olevan ohjelmakoodin tietokoneen ymmärtämään binääriseen muotoon. Näin ollen ohjelmointiympäristöt tekevät sovelluskehityksestä paljon nopeampaa kuin aikoinaan.

4.3.2 Vaihtoehdot

Qt Creator

Qt Creator (ks. liite 1) on ohjelmointiympäristö, jolla ohjelmistokehitystä voidaan tehdä alustariippumattomasti. Kehittäjät voivat luoda sovelluksia monille työpöytä -, mobiili- ja sulautetuille laitealustoille. Qt Creator tarjoaa tuen C++-ohjelmointikielelle ja QML-kuvauskielelle. Ohjelmointiympäristössä on myös tuki yleisimmille versionhallintajärjestelmille, mikä edesauttaa ohjelmakoodin nopeaa versionhallintaan varastointia suoraan ohjelmointiympäristön kautta. (The IDE Qt Creator n.d.)

Visual Studio

Visual Studio on ohjelmointiympäristö, jonka avulla voidaan toteuttaa eri tarkoituksiin olevia ohjelmistoja. Visual Studio on saatavilla sekä Windows- että macOS-käyttöjärjestelmille. Molemmilla käyttöjärjestelmillä voidaan toteuttaa mobiili- ja webbisovelluksia, pelejä ja käytettävälle kehityskäyttöjärjestelmälle kohdistettuja sovelluksia. (Visual Studio IDE n.d.)

4.4 Ohjelmointikielet

4.4.1 Yleistä

Ohjelmointikielellä tarkoitetaan tapaa, jolla sovelluskehityksessä kommunikointi tapahtuu ihmisen ja tietokoneen välillä. Ohjelmistot luodaan käyttämällä ohjelmointikieltä, jonka avulla voidaan määrittää ohjelman toiminta. Ohjelmointikielissä on olemassa omat standardit merkintätavat, joiden avulla voidaan alkaa teoriassa rakentaa minkälaista ohjelmistoa tahansa. (Programming Language n.d.)

Ohjelmakoodi voidaan mieltää eräänlaisiksi ohjeiksi tietokoneelle. Ohjelmakoodissa komennetaan tietokonetta tekemään tarvittavat toiminnot, jotta selvitään muun muassa poikkeustilanteista. ”Esimerkiksi tilanteessa, jossa tapahtuu seuraavasti, niin menettele tällä tavalla.” Nämä ovat yksiselitteisiä toimintamalleja, joita tietokone noudattaa pilkun tarkkuudella ja tekee sen joka kerta sillä tavalla, miten se on ohjelmakoodiin määritetty ennen sen kääntämistä.

4.4.2 Vaihtoehdot

C++

C++ on C-ohjelmointikieleen perustuva Bjarne Stroustrupin kehittämä ohjelmointikieli, johon on lisätty olio-ohjelmointiin kuuluvia ominaisuuksia. C++-kieltä voidaan ohjelmoida joko käyttämällä C-tyyliä tai olio-ohjelmointityyliä. C++-kieltä käytetään yleisesti erilaisten järjestelmien, sovellusten, sulautettujen järjestelmien ja ajureiden ohjelmointiin. Kielen etuuksina on se, että käyttäjä voi luoda omia luokkia, joista voidaan luoda rajattomasti uusia olioita eli instansseja. Luokkia voidaan yhdistää käyttämällä olio-ohjelmointiin kuuluvaa perintää, jolloin perivä luokka saa toisen luokan julkiset tiedot käyttöönsä. (C++ Programming Language n.d.)

Java

Java on ohjelmointikieli, jota käytetään hajautetuissa Internet-ympäristöissä. Java on luotu muistuttamaan C++-kieltä ja on näin ollen myös olio-ohjelmointiin orientoitunut ohjelmointikieli. Java on tällä hetkellä yleisin käytetty ohjelmointikieli mobiilikäyttöjärjestelmä Androidin sovelluksille. Javalla voidaan toteuttaa itsenäisiä sovelluksia, mutta

myös asiakas-palvelintyyppisiä ohjelmistoratkaisuja. Java on helppo oppia, jos on kokemusta C++-kielestä, koska Java muistuttaa vahvasti sitä. (Rouse 2016b.)

4.5 Versionhallinta

4.5.1 Yleistä

Versionhallinnalla tarkoitetaan ohjelmistollista aputyökalua, jota käytetään ohjelmistoprojekteissa hallinnoimaan ohjelmakoodin versiointia. Versionhallinta pitää kirjaa siellä oleviin tiedostoihin tapahtuvista muutoksista. Versionhallinnan avulla useampi ohjelmoija voi työskennellä saman projektin eri tiedostojen parissa yhtä aikaa, minkä jälkeen he voivat julkaista muutoksensa versionhallintaan ja molemmilla on toisen tekemät muutokset omassa versiossaan mukana. Mahdollisten inhimillisten tai teknisten virheiden vuoksi versionhallinnasta voidaan aina palauttaa vanha versio projektin ohjelmakoodista. Ilman versionhallintaa kahden henkilön tekemän työn yhdistäminen on tuskallista, koska se täytyy tehdä manuaalisesti siirtämällä tiedostoja toiselle henkilölle ja päinvastoin sekä varmistaa, että molemmilla on nyt viimeisin versio käytössään. Versionhallintaa käyttämällä tällaisista ongelmista ei tarvitse huolehtia. (What is version control n.d.)

4.5.2 Git

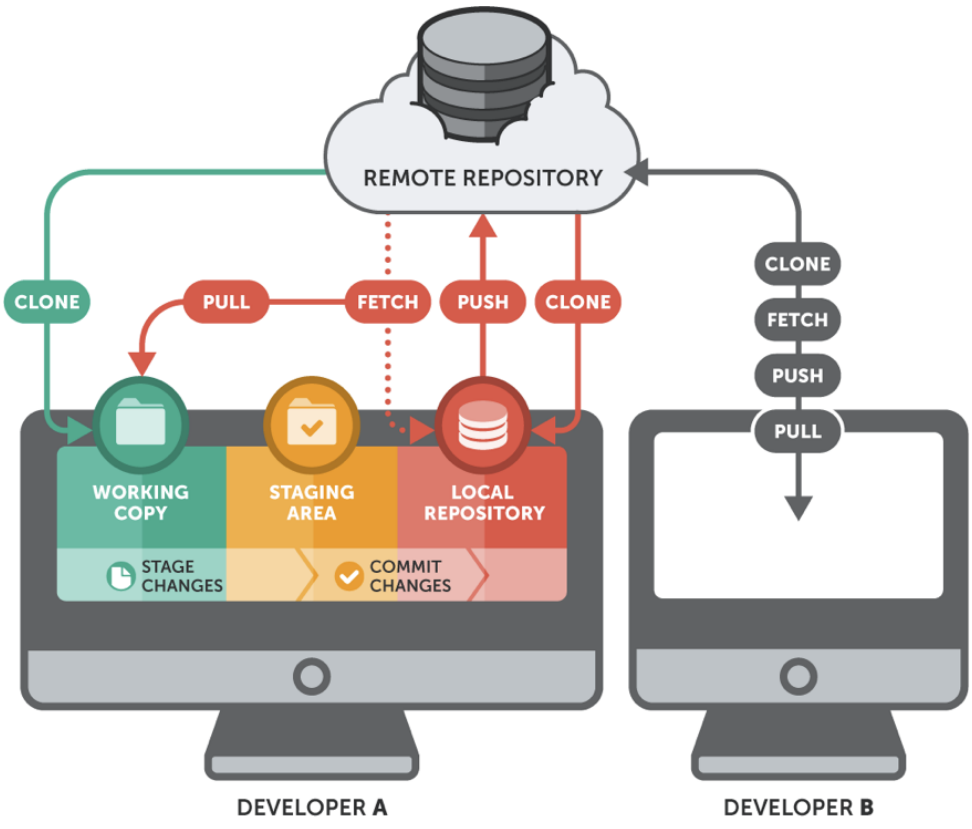
Git on Linus Torvaldsin eli Linux-käyttöjärjestelmän ytimen kehittäjän vuonna 2005 tekemä moderni nykyajan hajautettu versionhallintajärjestelmä. Gitin käyttö on noussut suureen suosioon sovelluskehittäjien keskuudessa viimeisen kymmenen vuoden aikana. Hajautettu versionhallintajärjestelmä tarkoittaa sitä, että jokaisella kehittäjällä voi olla tietokoneellaan oma paikallinen kopio varsinaisesta ohjelmavarastosta eli niin sanotusta repositoriosta, joka sisältää koko versiohistorian. Gitin avulla sovelluskehitystä voidaan tehdä ilman verkkoyhteyttä ja näin ollen siirtää muutokset paikalliseen repositorioon ja vasta myöhemmin etärepositorioon verkkoyhteyden ollessa saatavilla. (What is Git n.d.; Luontola & Paksula 2009, 3,5.)

Tavallisissa versionhallintajärjestelmissä kuten Subversionissa koko versiohistoria on keskitetty yhteen tiettyyn paikkaan eikä käyttäjillä ole mahdollisuutta pitää tietokoneellaan paikallista repositoriota. Tämä tarkoittaa sitä, että tällöin käyttäjällä on pakko

olla verkkoyhteys saatavilla, jotta muutokset voidaan siirtää keskitettyyn etärepositorioon. Tehdyt toiminnot ovat myös useammin hitaampia, kun verkkoyhteys on vaatimuksena, kun taas Gitin avulla toiminnot ovat nopeita, koska ne tapahtuvat pääosin paikallisesti kehittäjän tietokoneella.

Gitin perusperiaatteena on kiertokulku eri työvaihetilojen välillä (ks. kuvio 3). Ohjelmistoprojektin alkaessa luodaan etärepositorio sen tallentamista tarjoavaan palveluun. Tämän jälkeen repositorio voidaan kopioida omalle työtietokoneelle komennolla ”clone”. Kehitystyötä voi tämän jälkeen aloittaa tekemään omalla tietokoneella, josta muutokset siirretään aluksi paikalliseen repositorioon ja siitä aina etärepositorioon saakka komennolla ”push”.

Toinen kehittäjä tekee omalla tietokoneellaan samat toiminnot ja he voivat kummatkin tehdä omia työtehtäviään ja sopivan tilaisuuden tullen aina siirtää muutokset etärepositorioon. Tavallisesti työpäivän alussa ennen omien töiden aloittamista on järkevää päivittää paikalliset tiedostot, mikäli joku on lisännyt uusia muutoksia etärepositorioon. Tämä tapahtuu komennolla ”pull”. Näin ollen varmistetaan se, että tiedostot ovat ajan tasalla.



Kuvio 3. Gitin perustoiminnallisuus (Learn Version Control with Git n.d.)

4.5.3 Vaihtoehdot

GitHub

GitHub on vuonna 2008 julkaistu tallennuspaikka ohjelmistoprojekteille, jotka käyttävät Git-komentorivityökalua. GitHub on graafinen selainkäyttöliittymä Git-repositorioiden hallintaa varten. GitHubissa oleva ohjelmakoodi on ilmaisversiossa kaikille julkisesti nähtävillä. GitHubin perimmäisenä ideana onkin olla eräänlainen sosiaalinen verkosto sovelluskehittäjälle, ja näin ollen se luo ympärilleen omanlaisen yhteisönsä, joka voi kommentoida ja vapaasti tarkastella muiden tuotoksia. GitHubista on myös olemassa maksullinen versio, jonne voi luoda yksityisiä repositorioita. (Rouse 2016a.)

GitLab

GitLab on GitHubin kaltainen vuonna 2011 julkaistu selainkäyttöliittymä Git-repositoiden hallintaan. GitLab on avoimen lähdekoodin ohjelmistoprojekti. Se tarkoittaa, että kuka tahansa voi tarkastella, muokata ja julkaista omia muutoksiaan projektiin. GitLabista on olemassa tarjolla asennettava versio ilmaisena sekä maksullisena. GitLab asennetaan tällöin omalle palvelimelle. GitLabista on myös olemassa selainversio samaan tapaan kuin GitHubista, johon käyttäjät voivat luoda repositorioita ilman ylimääräistä asennustyötä ja oman palvelimen ylläpitämistä. (GitLab Inc n.d.)

4.6 Muut työkalut

Audacity

Audacity (ks. liite 1) on ilmainen avoimen lähdekoodin sovellus, joka on tarkoitettu audion muokkaamiseen, nauhoittamiseen ja toistamiseen. Audacity on saatavilla Windows-, Linux- ja macOS-käyttöjärjestelmille. Audacityn avulla voidaan muokata useita eri audioformaatteja sekä tuoda audiota sovellukseen useissa eri formaateissa. Audacity tarjoaa graafisen käyttöliittymän, jossa käyttäjä voi nähdä audion aaltoliikeen muodossa. (About Audacity n.d.)

Tämä sovellus oli erittäin tärkeässä roolissa koko opinnäytetyöprojektin onnistumisen kannalta. Audiotallentimen audiopuskurin tallentamat audiovirrat voitiin tuoda suoraan Audacity-ohjelmaan halutussa formaatissa. Audiovirran käyrästä voitiin havain-

noida mahdollisia poikkeamia ja ongelmatilanteita audiossa. Tämän avulla voitiin varmistaa audiotallentimen oikeaoppinen toiminnallisuus, kun tarkasteltiin graafista audiovirtaa ja peilattiin tilanteeseen, jossa esimerkiksi yhteys oli katkennut ja myöhemmin palautunut. Audiotallentimessa muokatusta audiosta voitiin Audacityssä selvästi havaita audiota toistettaessa, mistä kohtaa audiota oli muokattu. Tämä korostui, jos audiota oli muokattu väärästä kohdasta. Kuuloaistille nämä välittyivät äänen pätkimisenä ja visuaalisesti audiovirran kuvaajassa olleina piikkikohtina.

HipChat

HipChat on australialaisen Atlassian yrityksen vuonna 2012 ostama yrityksille ja työyhteisöille tarkoitettu pikaviestintäpalvelu. Atlassian osti HipChatin kolmelta kehittäjältä ja palkkasi heidät jatkamaan kehitystyötä heidän toimistoonsa San Franciscoon. HipChatissa voi luoda työyhteisölle omia yksityisiä ryhmäkeskusteluita sekä lähettää valitulle henkilölle yksityisviestejä. (Team chat that's actually built for business 2017; We've been acquired by Atlassian! 2012.)

Tämä palvelu päätettiin ottaa mukaan tähän opinnäytetyöprojektiin pikaviestittelyä varten. Toimeksiantajalla oli ollut käytössään kyseinen sovellus muissakin projekteissa, joten sen käyttöönotto oli vaivatonta.

Jira

Jira (ks. liite 1) on Atlassianin kehittämä tehtävienhallintaohjelmisto, joka on tarkoitettu ohjelmistoihin liittyvien erilaisten ohjelmistovirheiden sekä ohjelmistoprojektien hallintaan. Jiraa käytetään eri aihealueisiin kohdistuvien tapausten raportoinnissa. Näitä tapauksia tavallisesti ovat projektin työtehtävät ja ohjelmistovirheet. Jirassa sovelluskehitystä voi tehdä ketterän kehityksen teemoin. Jirasta löytyvät Scrumille olennaiset sprintit, näkymät sprinttien suunnitteluun sekä Scrumin Board-näkymä, jossa voidaan työtehtävien tilaa hallita - käytännössä jaottelemalla tehtävät sen mukaisesti, mitkä ovat vielä tekemättä, työn alla ja tehtynä. (JIRA Software n.d.)

Jira koettiin tarpeelliseksi opinnäytetyöprojektin luonteen kannalta, koska työ päätettiin toteuttaa ketterän kehityksen menetelmin Scrumin ja sprinttien muodossa. Jira oli oiva työkalu suunnittelutyössä sekä projektinhallinnassa. Toimeksiantaja oli aiemmin-

kin käyttänyt Jiraa muissa projektitöissä, joten käytännöt olivat tuttuja jo entuudestaan. Tällaisen oikean projektinhallintatyökalun ottaminen mukaan opinnäytetyöprojektiin oli hyvä harjoitus opinnäytetyön tekijälle. Varsinaisissa asiakasprojekteissa projektinhallintaan tarkoitetut työkalut ovat isossa roolissa ja niitä tulee osata käyttää tarkoituksensa mukaisesti.

LibreOffice

LibreOffice on ilmainen avoimen lähdekoodin ohjelmisto, johon kuuluvat Microsoft Office -ohjelmiston kaltaiset sovellukset. Microsoft Excel -taulukkolaskentaohjelman kaltainen versio LibreOfficessa on Calc. Calc-sovelluksesta (ks. liite 1) löytyvät Microsoft Excelin tapaan perinteiset laskutoiminnallisuudet ja kuvaajat. (What is LibreOffice? n.d.)

LibreOffice valittiin opinnäytetyöprojektiin Microsoft Excelin korvaajaksi, koska LibreOffice on ilmainen ohjelmisto. LibreOfficen Calc sovellusta hyödyntämällä saatiin luoduista metadatatiedostoista luotua haluttuja kuvaajia, joista ilmeni audiotallentimen toiminta eri poikkeustilanteissa. Näiden kuvaajien avulla voitiin todistaa toteutetun audiotallentimen haluttu toiminnallisuus ja perustelemaan eri tapauskohtaiset tilanteet.

4.7 Teknologiavalinnat ja niiden perustelut

Käyttöjärjestelmä

Käyttöjärjestelmäksi tähän opinnäytetyöprojektiin valittiin Microsoft Windows 10 käytännössä yksinomaan sen takia, että kehitystyötä toteutettiin toimeksiantajan tietokoneella, jossa oli käytössä kyseinen käyttöjärjestelmä.

Ohjelmointiympäristö

Ohjelmointiympäristön valinnan osalta päädyttiin valitsemaan Qt Creator. Qt Creatorissa etuna oli hyvä tuki C++-ohjelmointikielelle, ja siinä oli lukuisia hyödyllisiä ohjelmistoluokkia valmiiksi asennettuna. Näitä valmiita luokkia hyödynnettiin merkittävissä määrin ohjelmiston toteutuksessa. Qt Creatorista oli saatavilla avoimen lähdekoodin ilmaisversio, joka sopi hyvin tähän opinnäytetyöprojektiin.

Visual Studio oli myös vaihtoehtona kehitysympäristön valinnassa. Visual Studiossa oli myös hyvä tuki C++-kielelle. Visual Studion Community-ilmaisversiossa ei ollut kuitenkaan samoja ominaisuuksia kuin maksullisissa versioissa. Tarvittavaa ohjelmistokoodin kääntäjää ei saatu enää toimimaan Visual Studion kanssa muutettujen ominaisuuksien johdosta ilmaisversioon, joten potentiaaliseksi vaihtoehdoksi jäi Qt Creator.

Ohjelmointikieli

Ohjelmointikieleksi opinnäytetyöprojektiin valittiin C++. Siinä koettiin olevan enemmän hyödyllisiä ominaisuuksia kuin Javassa. Valittua kieltä tuki se, että Qt Creatorissa oli hyvä yhteensopivuus C++:n kanssa ja Qt:n tarjoamat ohjelmistoluokat oli kirjoitettu C++-kielellä. Combitech Oy:n Timo Mustosen toteuttama AudioSource-ohjelma oli myös toteutettu C++-kielelle, joten C++ katsottiin olevan etuna tulevan ohjelmiston rakentamisessa.

Versionhallinta

Hajautetuista versionhallintaohjelmistoista valittiin opinnäytetyöprojektiin GitLab ohjelmakoodin varastointia ja versiointia varten. Tärkeimpinä eroina GitHubin ja GitLabin välillä on se, että GitHub on enemmän panostanut yhteisölliseen ohjelmistokehitykseen, kun taas GitLab on keskittynyt kehittämistyössään enemmän käytännön asioihin versionhallinnassa sekä tapauksien ja havaittujen ohjelmistovirheiden helpompaan käsittelyyn. Jokaiseen tapaukseen voidaan GitLabissa esimerkiksi lisätä haluttu liitetiedosto ja käyttöoikeuksia voidaan jakaa koskemaan vain tiettyä osa-aluetta. Tärkeänä erona ovat myös hinnoitteluerot ohjelmistojen välillä. GitHubissa maksulliset versiot ovat huomattavasti kalliimpia kuin GitLabissa. (GitLab vs GitHub n.d.)

GitHubiin verrattuna GitLabin ilmaisessa versiossa oli mahdollista luoda yksityisiä repositorioita. Tämä oli merkittävimpänä syynä valinnassa. Muutoin GitHub olisi ollut aivan yhtä pätevä versionhallintaohjelmisto tähän projektiin, koska versionhallintaan kuuluvia kaikkia hienouksia ei tässä projektissa edes käytetty. Ohjelmakoodin yksityinen versiointi ja varastointi olivat ainoat prioriteetit. Lähdekoodin hajautettu versionhallinta koettiin olevan paras ja helpoin tapa tällaisen pienen opinnäytetyöprojektin hallintaan ja läpivientiin. Hajautettu versionhallinta ja sen käyttö olivat jo entuudestaan tuttuja asioita, joten niiden opiskeluun tai harjoitteluun ei tarvittu käyttää juurikaan resursseja.

5 Toteutus

5.1 Yleistä

Toteutusvaiheessa suunniteltu ohjelmisto toteutetaan yleensä käyttämällä valittuja teknologioita. Toteutettavaan ohjelmistoon toteutetaan vaatimusmäärittelyssä määritetyt toiminnallisuudet ja muut vaatimukset. Vaatimusten määrittely tarkentuu kooditasolla kehitystyön edetessä ja niitä jaotellaan yhä pienempiin osa-alueisiin. Toteutusvaihe on pääsääntöisesti pitkäkestoisin prosessi ohjelmistoprojekteissa. Poikkeuksena tähän voidaan mainita ohjelmistojen mahdollinen vuosia kestävä ylläpito asiakkaan tarpeisiin vastaten sekä ohjelmistovirheiden ja -haavoittuvaisuuksien korjaaminen. Tällainen toimenkuva on hyvin tavallista ohjelmistoalan projekteissa, joissa toteutetaan asiakkaalle järjestelmä, jota ylläpidetään säännöllisesti. Asiakasprojekteissa tuotto ei koostu toteutetun ohjelmiston valmistuskustannuksista, vaan ohjelmiston ylläpitokustannuksista.

Toteutusvaiheessa yleensä keskitytään vaatimusmäärittelyn sanelemiin kehyksiin ja toteutetaan ohjelmisto mahdollisimman hyvin sitä kohti. Peilataan suunniteltuja vaatimuksia siihen, mitä lopullinen tuotteen käyttäjä oikeasti haluaa ohjelmiston mahdollistavan ja miten hän sitä käyttää. Useimmiten toteutusvaiheessa keskustellaan useaan otteeseen mahdollisen asiakkaan sekä projektiin kuuluvien sidosryhmien kanssa siitä, mihin suuntaan projekti on etenemässä ja ollaanko menossa oikeaan suuntaan. Tämä edesauttaa sitä, että tarvittaessa voidaan kehityslinjaa oikaista, ennen kuin isompaa kurssin muutosta kehityskaaressa on tapahtunut. (Rouse 2015.)

Nopea muutoksiin reagoiminen ja suora viestintä eri sidosryhmien välillä ovat keskeisimmät tavoitteet ketterässä ohjelmistokehityksessä. Ketterässä ohjelmistokehityksessä jatkuva suunnittelu ja testaus ovat koko ajan läsnä, koska saadun palautteen perusteella suunnitelmat voivat muuttua äkisti. Nopea päätöksien tekokyky ja niiden toteuttaminen ovat suuressa roolissa ketterässä kehityksessä. Ketterässä kehittämisessä pyritään saamaan nopeasti valmista aikaan, mikä voi toisinaan johtaa huonoihin arkkitehtuurillisiin ratkaisuihin, joita joudutaan myöhemmin korjaamaan. Periaatteena on olla suorassa yhteydessä asiakkaaseen ja saada tuotettua aina esittelyä varten toimiva versio toteutettavasta ohjelmistosta. (What Is Agile Software Development? n.d.)

5.2 Lisätyt toiminnallisuudet

Toteutusvaiheen aikana opinnäytetyölle asetetut vaatimukset syventyivät ja niitä tuli lisää sen mukaisesti, mihin suuntaan työ eteni. Havaittiin, että joistain lisäyksistä olisi myöhemmin hyötyä tulosten todistamisessa ja analysoinnissa. Tallentimesta haluttiin myös tehdä dynaamisempi tarvittavien konfiguraatioiden avulla. Näitä lisättyjä toiminnallisuuksia pohdittiin etenkin jokaisen sprintin päätyttyä olleissa sprinttisuunnitteluissa, joissa luotiin tarvittavat käyttäjätarinat tulevaa sprinttiä varten.

Kehitystyö eteni suunnitellusti ja määrätietoisesti kohti haluttua päämäärää. Ohjelmointityön luonteeseen kuuluu hieman muokkaantua toteutustyön aikana ja vähitellen rakentua kohti lopullista päämäärää. Täten ei ole tavatonta, että uusia ideoita tulee kehitystyön aikana, kun kokonaiskuva alkaa hahmottua paremmin. Onkin liki mahdotonta pystyä ennalta suunnittelemaan haluttuja toiminallisuuksia niin tarkasti, ettei toteutuksessa tarvitsisi juurikaan mitään lisätä. Seuraavissa kahdessa kappaleessa nämä halutut lisätoiminnallisuudet ovat kuvattuna tarkemmin.

Audiopuskurin luonnin yhteydessä tuli pystyä muuttamaan puskurin konfiguraatiotietoja oman konfiguraatiotiedoston avulla, minkä takia audiotallentimelle tuli toteuttaa toiminnallisuus, jotta tiedostosta voitiin lukea puskurin käynnistymisen yhteydessä tarvittavat parametritiedot. Audiopuskurin luonnin yhteydessä luotiin myös audiopuskurin metatietoja varten oma tiedosto, johon tallennettiin audiopuskurin asetustiedot. Käytännössä nämä tiedot kertoivat puskurille, minkälaista audiovirtaa puskurille haluttiin lähettää. Näihin tietoihin perustuen testauksessa voitiin lähettää tietylle audiopuskurille audiovirtaa nimenomaan halutussa formaatissa.

Vastaanotetusta audiovirrasta ja puskuroinnin aikana tapahtuvista muutoksista haluttiin tallentaa tiedostoon metatietoa, josta voitiin myöhemmin mallintaa kuvaajia LibreOfficen Calc-talukkolaskentaohjelman avulla. Näistä kuvaajista voitiin havainnollistaa ja todeta audiotallentimen oikeaoppinen toiminnallisuus. Luoduista kuvaajista ja tallennetun audiovirran graafisesta mallista Audacity-ohjelmassa voitiin vetää johtopäätöksiä eri tapahtumiin vedoten ja näin ollen todistaa toteutettu toiminnallisuus. Esimerkiksi yhteyden katkeamistilanne ja - kohta olivat selvästi nähtävissä sekä audiossa että metatiedostossa. Näiden vertailuiden ja kuvaajien avulla pystyttiin selvittämään ja perustelemaan tutkimustyön tulokset, havainnot ja johtopäätökset.

5.3 Audiotallennin

Audiotallentimen toiminnallisuus jakautuu moneen eri osa-alueeseen, jotka hoitavat niille tarkoitetut toiminnallisuudet ja vastuualueet. Nämä osa-alueet ja eri tilanteiden hallinta tullaan käymään edempänä yksityiskohtaisesti ja yksitellen läpi. Nämä eri osa-alueet ovat audiopuskuri, audioformaatti ja konfiguraatiot, audion vastaanottaminen, audion tallentaminen, audion normalisointi, audiopuskurin metadata ja kaaviot.

Audiotallennin toteutettiin itsenäiseksi palveluksi, joka oli valmiudessa vastaanottamaan luomiinsa audipuskureihin audiovirtaa verkosta. Näissä audiopuskureissa audio käsiteltiin ja siihen tehtiin tarvittavat korjaukset, minkä jälkeen audio tallennettiin kiintolevylle. Samalla audiopuskuriin kerättiin tallentamisen aikana koko ajan uutta audiovirtaa käsittelyä varten.

Audion kirjoittamisen yhteydessä kirjoitettiin myös toiseen tiedostoon metatietoa audiopuskurin tilasta jokaisen käsitellyn datapaketin jälkeen. Audiotallentimen käynnistyksen yhteydessä luotiin määrätty määrä audiopuskureita asetetuilla asetus - ja konfiguraatiotiedoilla. Jokaiselle puskurille luotiin oma palvelin, jonka avulla verkosta voitiin vastaanottaa audiovirtaa audiopuskurille käsiteltäväksi.

Audiotallentimeen toteutettiin myös yhteydenmuodostus toiminnallisuus Reitityspalveluun, jotta voitiin testimielessä kokeilla koko järjestelmää yhtenä isona kokonaisuutena. Oikeassa toimenkuvassa Reitityspalvelulla olisi tieto siitä, että minkälaisessa formaatissa olevia audiolähteitä on olemassa ja sen mukaan käskisi audiotallentimen luoda oikeanlainen audiopuskuri, jonne audiota voitaisiin lähettää, mikäli sellaista ei olisi vielä valmiiksi olemassa.

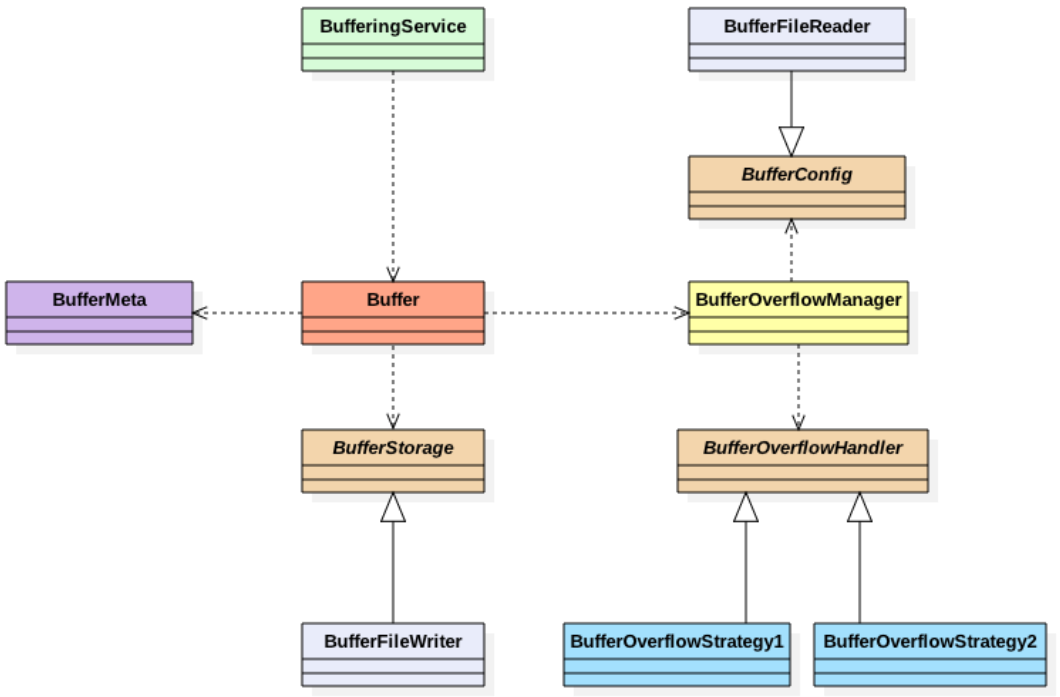
Tässä projektissa kuitenkaan kokonaisen järjestelmän testaaminen ei ollut pääpainopisteenä, vaan ennemminkin saada toteutettua kaksi opinnäytetyötä, jotka toimisivat omina kokonaisuuksinaan. Tämän johdosta testauksessa audiotallennin loi satunnaisesti eri formaattitiedoilla olevia audiopuskureita 5 kappaletta. Näitä formaattitietoja käytettiin testaamisessa apuna, kun AudioSource-ohjelmalla lähettiin tietylle audiopuskurille oikeassa formaatissa olevaa audiovirtaa, jotta mahdollisilta lisäongelmilta vältyttiin ja voitiin keskittyä varsinaisen opinnäytetyön ongelman ratkaisuun.

5.4 Kaaviot

5.4.1 Luokkakaavio

Luokkakaavio on ohjelmistokehityksen ja nimenomaan olio-ohjelmoinnin yksi perinteisimmistä kuvausmalleista, kun kuvataan toteutetun ohjelmiston rakennetta. Luokkakaaviosta (ks. kuvio 4) on nähtävissä tähän opinnäytetyöhön toteutetut ohjelmistoluokat sekä niiden suhteet toisiinsa. BufferingService-luokka vastaa tässä kontekstissa toteutettua audiotallenninta, joka on riippuvainen luomistaan audiopuskureista, jotta dataa voitiin vastaanottaa. Keskellä oleva Buffer-luokka tarvitsee toimiakseen luokkia, joihin se on kytköksissä. Näitä ovat BufferMeta, BufferStorage ja BufferOverflowManager.

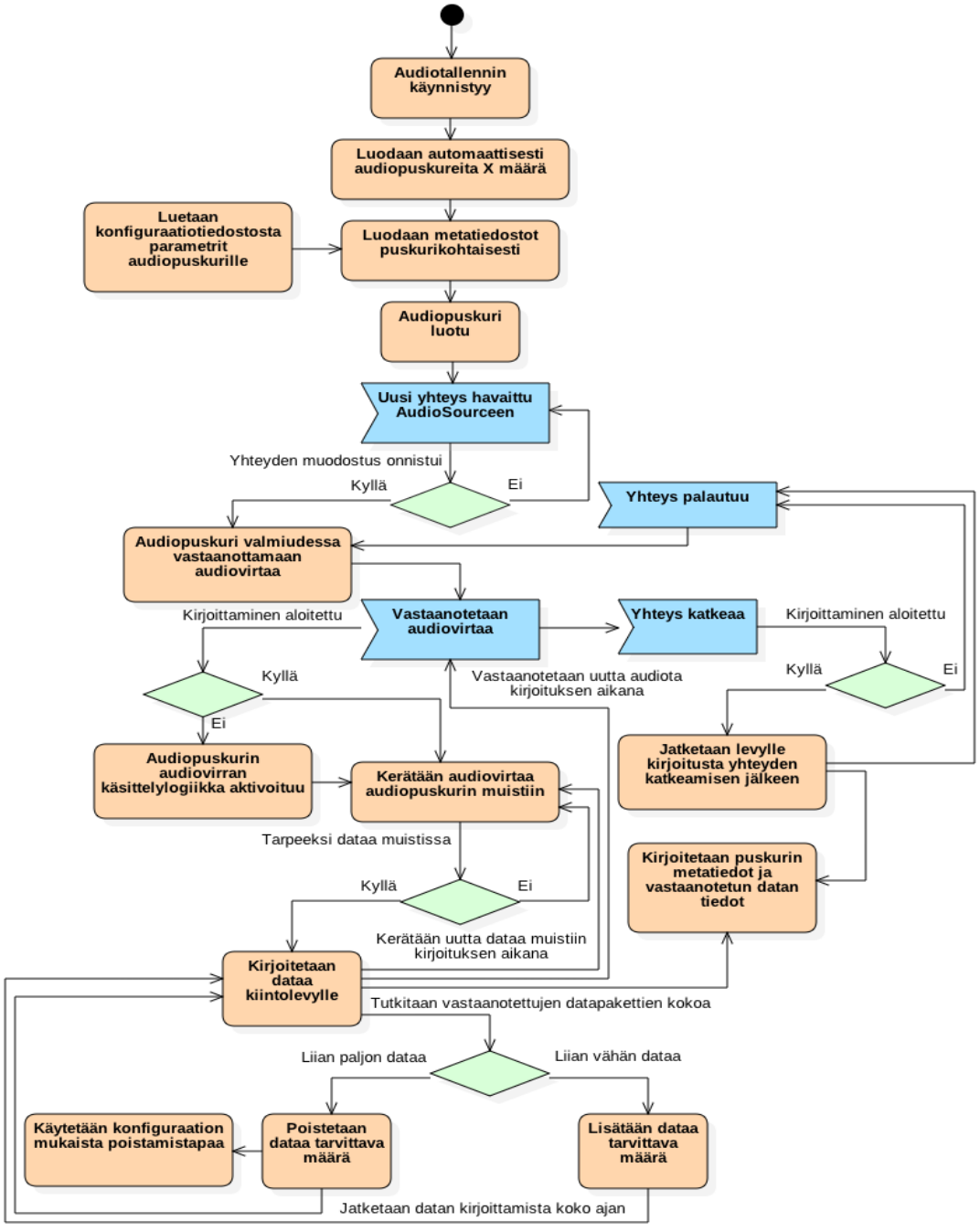
Ruskealla värityksellä olevista abstrakteista luokista on peritty aliluokka/aliluokat, jotka toteuttavat halutun toiminnallisuuden eri tavoin. Abstraktin luokan hyötynä on se, että esimerkiksi BufferStorage-luokasta voitaisiin toteuttaa datan tallentamiseen erilaisia ratkaisuja, eikä tallentaminen olisi sidottu nimenomaan tiedostoon tallentamiseen, vaan esimerkiksi johonkin muuhun formaattiin.



Kuvio 4. Luokkakaavio

5.4.2 Aktiviteettikaavio

Audiotallentimen aktiviteettikaaviosta ilmenee audiotallentimen aktiviteetit eli toiminnallisuudet, jotka se toteutti (ks. kuvio 5). Kuviosta 5 ilmenee, miten audiotallentimen sisäinen toiminnallisuus ja tapahtumien kulku eteni. Pääideana oli se, että audiotallentimen alustusvaiheen jälkeen se oli valmiina vastaanottamaan audiovirtaa, puskuroimaan sitä, normalisoimaan sen, tallentamaan sen kiintolevylle sekä selviämään mahdollisista yhteyden katkeamistilanteista.



Kuvio 5. Aktiviteettikaavio audiotallentimen toiminnasta

5.5 Audiopuskuri

5.5.1 Yleistä

Audiopuskuri toteutettiin audiotallentimen toiminnallisuuden ytimeksi. Se vastasi käytännössä kaikesta toiminnallisuudesta audiotallentimessa käyttäen muita luotuja ohjelmistoluokkia tarvittavien toiminnallisuuksien toteuttamiseen. Audiopuskurista yksinkertaisesti kutsuttiin toisen ohjelmistoluokan funktiokutsua halutun toiminnallisuuden tai ongelmakohdan hallitsemiseen. Tästä esimerkkinä oli opinnäytetyön ydinongelman ratkaiseminen eli audiovirtojen normalisointi ennen tallentamista.

5.5.2 Audiopuskurin palvelimen alustus

Jokaiselle audiopuskurille luotiin puskurin käynnistyksen yhteydessä oma palvelin (ks. kuvio 6) käyttämällä Qt Creatorissa olevaa QTcpServer-ohjelmistoluokkaa. Tämän palvelimen avulla jokainen luotu audiopuskuri oli kykenevä vastaanottamaan dataa ennalta määritetystä tietoliikenneportista käyttäen TCP-yhteyttä. Tämän jälkeen puskuriin vastaanotetut datapaketit tarkistettiin, ennen kuin ne tallennettiin kiintolevylle audiopuskurin luomaan tiedostoon.

Testauksen helpottamiseksi audiovirtaa lähettävä AudioSource-ohjelma ja Audiotallennin olivat käynnissä samalla tietokoneella, joten verkko-osoitteena käytettiin nimeä ”localhost”. Tämä viittasi verkkoympäristössä kyseiseen käytössä olevaan tietokoneeseen. Tämä helpotti sitä, ettei IP-osoitetta tarvinnut erikseen etsiä tai muistaa. Uuden yhteyden tunnistamiseen käytettiin Qt:ssa mukana olevia signaaleja ja niin sanottuja ”slotteja”. Se tarkoitti käytännössä sitä, että kun jokin uusi signaali havaittiin, siirryttiin kuvattuun funktioon hallinnoimaan se.

```
class Server : public QObject{
  Q_OBJECT
public:
    Server(Buffer * b, int32_t port) : QObject(b), buffer(b){
        portnr = port;

        connect(&server, &QTcpServer::newConnection, this, &Server::acceptConnection);

        if (server.listen(QHostAddress::Any, port)){
            qDebug() « "Server started at port: " « port;
        }else{
            qDebug() « "Server could not start";
        }
    }
```

Kuvio 6. Audiopuskurin palvelimen alustus

5.5.3 Audiopuskurin alustus

Audiopuskurille luotiin oma yksityinen ohjelmistoluokka, josta käytettiin muuttujaa ”d”. Tässä luokassa oli tarvittavat muuttujat audiopuskurin toimintaa varten. Yksityisen luokan hyötynä on se, että muut ohjelmistoluokat eivät voi käyttää yksityisen luokan muuttujia. Audiopuskurin alustuksessa (ks. kuvio 7) sille luotiin nimikohtainen kansio määritettyyn sijaintiin kiintolevylle, jonne se myöhemmin loi halutut tiedostot ja kirjoitti niihin dataa käyttäen kirjoittamiseen luotua ohjelmistoluokkaa. Alustuksessa luotiin audiopuskurin sisäisen Server-luokan avulla aiemmin kuvattu palvelin haluttuun tietoliikenneporttiin sekä luotiin audion normalisoinnista huolehtiva manageri.

Audiopuskurit luotiin ennalta määritetyillä asetusarvoilla. Nämä arvot kertoivat siis audioformaatin eli minkälaista audiota audiopuskuriin oli tarkoitus vastaanottaa. Tästä kerrotaan enemmän luvussa 5.6.2. Alustuksessa kirjoitettiin nämä audioformaatin tiedot puskurin luomaan metatiedostoon. Samaan metatiedostoon kirjoitettiin myös konfiguraatiotiedostoon asetettujen muuttujien arvot.

Näiden konfiguraatiotietojen ja valitun audioformaatin perusteella voitiin esimerkiksi laskea, kuinka iso kooltaan tulisi audiopuskurin muistipuskurin olla, jotta se on ajallisesti esimerkiksi sekunnin mittainen. Tämä tarkoitti sitä, että haluttiin tietää, kuinka paljon tietyllä näytteenottotaajuudella tulisi sekunnin mittaisen audiopuskurin koon olla tavuina (1 tavu on 8 bittiä). Tätä tietoa käytettiin hyväksi audion normalisoinnissa, kun tarkkailtiin puskurin kokoa, jotta se ei alkanut kasvaa liian paljon määritellystä sekunnin mittaisesta muistipuskurin koosta.

```
bool Buffer::initBuffer(int32_t port, const QString &bufferName, QAudioFormat &audioFormat){
    this->setBufferName(bufferName);
    d->storageWriter.reset(new BufferFileWriter());
    d->manager.reset(new BufferOverflowManager());

    if (!d->storageWriter->initData(this->getBufferName())){
        d->storageWriter.reset();
        d->manager.reset();
        return false;
    }
    if (!d->manager->initManager(d->storageWriter)){
        d->storageWriter.reset();
        d->manager.reset();
        return false;
    }

    d->server = new Server(this, port);
    d->configReader.reset(new BufferFileReader());
    d->bufferSizeInMilliseconds = std::stoi(d->configReader->readBufferSizeInMilliseconds().toStdString());
    d->bufferMeta.init(audioFormat, d->configReader->readDropStrategy(), d->configReader->readDropPercentage(),
                       QString::number(d->bufferSizeInMilliseconds));
    d->storageWriter->writeBufferInitMetaData(bufferName, d->bufferMeta);

    d->bytesForBufferDuration = d->bufferMeta.getBytesForDuration(d->bufferSizeInMilliseconds * d->conversionRate);

    return true;
}
```

Kuvio 7. Audiopuskurin alustaminen

5.6 Audioformaatti ja konfiguraatiot

5.6.1 Yleistä

Monissa järjestelmissä tarvitaan konfiguraatiotietoja ja asetustietoja myöhempää käyttöä varten. Tässä opinnäytetyössä audiotallentimen luomat audiopuskurit asetettiin tiettyyn audioformaattiin. Näiden audiopuskurien audioformaatti tiedot haluttiin tallentaa myöhempää käyttöä varten omaan tiedostoonsa, että sieltä voitiin tarkistaa arvot, joilla puskuri oli luotu.

Tarkoituksena oli myös, että mikäli audiotallennin tai audiopuskuri lakkaisi toimimasta, olisi mahdollista käynnistää audiopuskuri samoilla asetustiedoilla näihin tallennettuihin metatietoihin pohjautuen. Tämän toiminnallisuuden toteuttaminen ei kuitenkaan katsottu olevan tarpeeksi korkealla prioriteettilistalla, joten sen toteuttaminen jäi ajatuksen tasolle.

5.6.2 Audioformaatti

Audioformaatin avulla kuvataan audiolle asetetut parametrit. Näitä parametriarvoja ovat näytteenottotaajuus, kanavien määrä, näytteen koko, näytteen tyyppi ja tavujärjestys. Nämä tiedot kertovat, miten audiossa data on järjestetty. QAudioFormat luokan avulla nämä tiedot voidaan asettaa halutulla tavalla. (QAudioFormat Class n.d.)

Näiden tietojen varastointiin käytettiin QAudioFormat-ohjelmistoluokkaa, joka tarjosi mahdollisuuden tallentaa ja hallita näitä tietoja ohjelmistoluokasta luodun olion (ohjelmistoluokasta luotu instanssi) avulla. Lähettävässä AudioSource-ohjelmassa audiovirta koodattiin PCM-formaattiin. Pulssikoodimodulaatiolla tarkoitetaan sähköisen äänisignaalin koodaamista digitaaliseen numeeriseen muotoon.

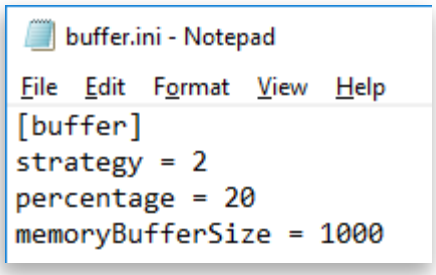
Audiopuskurin luomaan tiedostoon tallennettiin XML-formaatissa (ks. kuvio 8) audiotallentimessa audipuskurille valittu audioformaatti ja konfiguraatiotiedostossa olevat parametritiedot sekä audiopuskurin tarkka ajankohta, jolloin se oli vastaanottanut ensimmäisen datapaketin verkosta. Aloitusajankohta laskettiin käyttämällä Unix epoch -aikaa. Unix epoch -aika on sekunteina kulunut aika määritellystä ajanhetkestä nolla, joka on 1.1.1970 keskiyöllä (What is epoch time? n.d.).

```
<buffer1>
 <BufferSize>1000</BufferSize>
 <ByteOrder>1</ByteOrder>
 <ChannelCount>1</ChannelCount>
 <Codec>audio/pcm</Codec>
 <Percentage>20</Percentage>
 <SampleRate>48000</SampleRate>
 <SampleSize>32</SampleSize>
 <SampleType>1</SampleType>
 <StartTime>1496302850760</StartTime>
 <Strategy>2</Strategy>
</buffer1>
```

Kuvio 8. Audiopuskurin audioformaatti ja asetustiedot

5.6.3 Konfiguraatiotiedosto

Konfiguraatiotiedosto luotiin manuaalisesti INI-tiedostoformaattissa (ks. kuvio 9). Tästä tiedostosta voitiin audiopuskurin käynnistyksen yhteydessä lukea (ks. kuvio 10) tarvittavat tiedot. Kuviosta 9 voidaan havainnoida, kuinka muistipuskurin koko luetaan INI-tiedostosta käyttäen Windowsin Windows.h-nimisessä tiedostossa olevaa C++-kielelle ohjelmoitua GetPrivateProfileString-funktiota, jonka tarkoituksena on lukea parametritietoja INI-päätteisistä asetustiedostoista. Konfiguraatiotiedostoon tallennettiin haluttu muistipuskurin koko millisekunneissa. Sinne tallennettiin myös normalisointia varten tieto valitusta strategiasta ja prosenttiosuus, jonka jälkeen strategia aktivoitiin. Normalisoinnista ja strategioista kerrotaan enemmän luvussa 5.10.



Kuvio 9. Audiopuskurin konfiguraatiotiedosto

```
QString BufferFileReader::readBufferSizeInMilliseconds(){
    char memoryBufferSize [100];
    GetPrivateProfileString("buffer", "memoryBufferSize ", "default",
                            memoryBufferSize , 100, "C:\\buffers\\buffer.ini");

    QString str(memoryBufferSize);

    return str;
}
```

Kuvio 10. Konfiguraatiotiedostosta lukeminen

5.7 AudioSource

5.7.1 Yleistä

AudioSource-ohjelma oli testikäyttöön tarkoitettu sovellus, jonka Timo Mustonen toteutti C++-ohjelmointikielellä. Ohjelman tarkoituksena oli toimia lähettävänä osapuolena, joka lähetti valittuun verkko-osoitteeseen ja tietoliikenneporttiin audiovirtaa datapaketteina valitussa audioformaatissa. Lähetetyt audiot olivat MP3-formaatissa olevia ääniraitoja. Nämä lähetetyt audiot vastaanotettiin audiopuskuriin käsittelyä varten. Audiopuskurit luotiin tarkoituksellisesti ennalta määritetyssä audioformaatissa, jotta AudioSource-ohjelmalla voitiin lähettää audiovirtaa samassa formaatissa.

5.7.2 Käyttö testauksessa

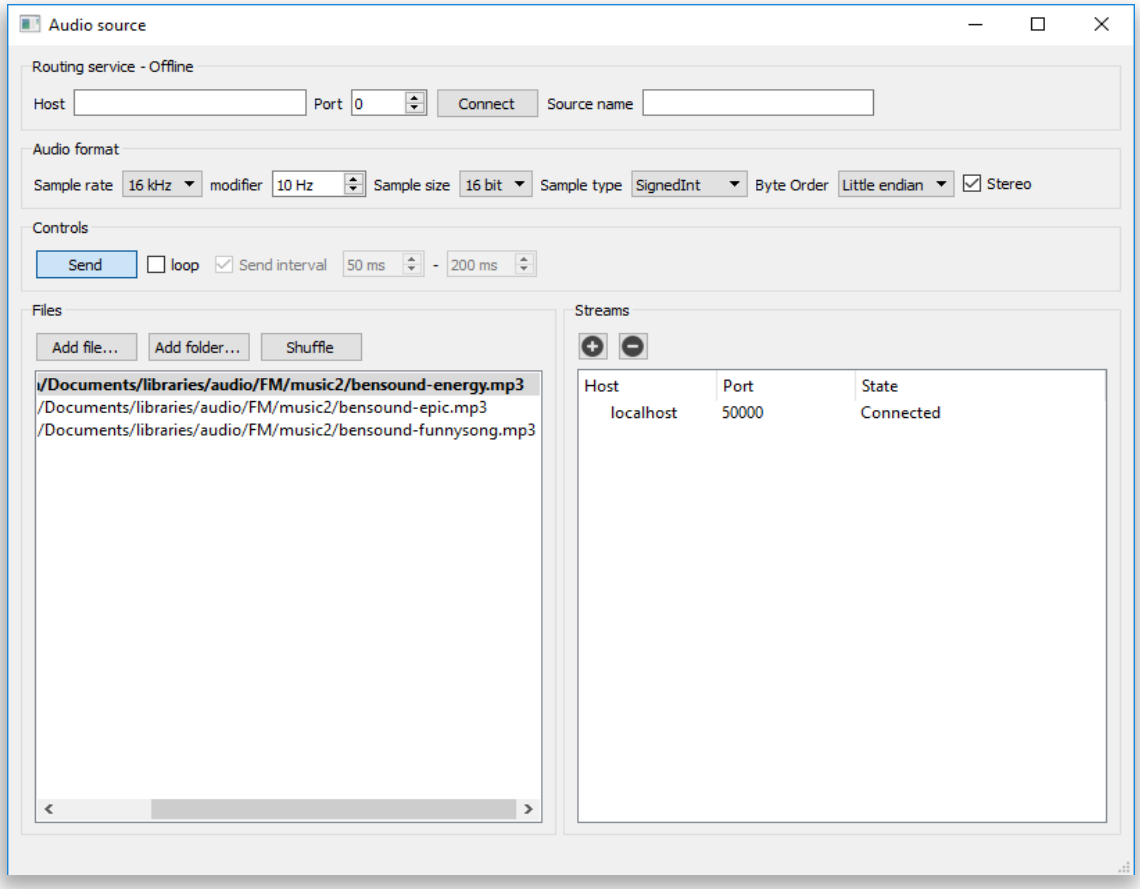
AudioSource-ohjelma (ks. kuvio 11) oli merkittävässä roolissa koko opinnäytetyön ohjelmointiosuuden ajan. Sen avulla voitiin tuoda esiin ongelmia, joita opinnäytetyössä oli tarkoituksena ratkaista. Esimerkiksi yhteyden katkeamistilanne oli helposti havainnollistettavissa AudioSourcen avulla. Lähetettävä audiovirta poistettiin listasta ja tämä tapaus saatiin havainnollistettua.

Aluksi testauksessa ei otettu mukaan muita lisähienouksia, joita ohjelmassa oli mahdollisuus lisätä. Pääpainopiste oli ensin saada toimimaan yhteyden muodostaminen audiotallentimen ja AudioSourcen välille sekä onnistua vastaanottamaan dataa halutusta verkkoportista. Kun tämä perustoiminnallisuus ja runko oli saatu kasattua, voitiin keskittyä varsinaisen ongelman ratkaisemiseen.

AudioSource-ohjelmasta oli mahdollista lähettää tahallisesti audiovirtaa suuremmalla tai pienemmällä näytteenottotaajuudella, kuin audiopuskuri oletti. Esimerkiksi 16 000 hertzin taajuudella lähetetty audiovirta voitiin vaihtaa, että sitä lähetetäänkin 16 010 hertzin taajuudella. Vastaanottavassa audiopuskurissa tämä johti ennen pitkää ongelmiin, koska vastaanottava audiopuskuri vastaanotti laskennallisesti enemmän dataa tietyssä aikayksikössä, kuin sen oli tarkoitus. Ongelma saatiin paremmin nostettua esille, kun taajuuden määrää kasvatettiin huomattavasti liioitelluksi.

Tämä esimerkki kuvastaa alun perin määriteltyä ydinongelmaa, jossa kaksi ohjelmistoa eivät ole aikasynkronisesti keskenään samassa aikakäsityksessä eli niiden käsitys esimerkiksi sekunnista on hieman eri mittainen. Taajuus kuvaa jonkin toistuvan tapahtuman määrää aikayksikköä kohti. Tässä tapauksessa audionäytteitä on sekunnissa esimerkiksi 16 000 kappaletta. Esimerkiksi lähteen A aikakäsitys sekunnista on aikasynkronisesti ja atomin tarkasti tasan sekunti, mutta lähteen B käsitys samasta sekunnista on 5 % pitempi kuin A:n. Tässä tapauksessa B lähettää 16 800 näytettä audiopuskurin olettaman sekunnin aikana. Tämä johtaa siihen, että ajan kuluessa audiopuskurin koko alkaa kasvaa lineaarisesti ylöspäin, koska puskuriin tulee uutta dataa enemmän, kuin sieltä kirjoitetaan pois.

AudioSource-ohjelmassa oli mahdollista havainnollistaa myös tapaus, jossa muistipuskuri ehdittiin kirjoittamaan tyhjäksi. Tällainen tilanne tapahtuisi, jos verkko olisi jumiutunut tai kovin hidas. Tällöin datapaketit saapuisivat isoina purskeina. Ohjelmasta voitiin määrittää, minkälaisin väliajoin audiota lähetettiin. Tässä tarkoituksena oli hallita audiopuskurissa tilanne, jossa puskuri oli tyhjä, mutta täyttyi nopeasti.



Kuvio 11. AudioSource-ohjelma

5.8 Audion vastaanottaminen

5.8.1 Yleistä

Audion vastaanottaminen toteutettiin käyttämällä QTcpSocket-ohjelmistoluokkaa. Sen avulla voitiin vastaanottaa verkosta dataa. Luokkaan kuuluvalla signaalifunktiolla readyRead (ks. kuvio 12) saatiin signaali tapahtumasta, jolloin havaittiin tietoliikennettä tulevan portista. Tämän signaalin käsittely voitiin sen jälkeen hallita määritellyssä erillisessä funktiossa, jossa vastaanotettiin kaikki TCP-yhteyden välityksellä lähetetyt datapaketit, jotka tässä tapauksessa olivat audiota.

Internetiin kytkettyjen tietokoneiden välillä voidaan lähettää eräänlaisia viestejä, jotka liikkuvat verkossa paketteina. Nämä paketit voidaan välittää TCP-yhteydellä tietokoneesta toiseen. TCP-protokolla on yhteyden ylläpitämiseen ja pakettien perille saattamiseen orientoitunut protokolla. Sen takia se pyrkii ylläpitämään yhteyden, kunnes kaikki paketit ovat saapuneet perille päätelaitteisiin. (Rouse 2014.)

5.8.2 Yhteyden muodostaminen

Yhteyden muodostuksessa (ks. kuvio 12) tarkistettiin aluksi, onko yhteys jo olemassa. Mikäli yhteys oli jo muodostettu ja portti oli käytössä, ei uutta yhteyttä enää sallittu, vaan se suljettiin. Muussa tapauksessa otettiin uusi yhteys vastaan ja luotiin sille tarvittavat signaalien käsittelyfunktiot connect-funktiota käyttämällä. Nämä funktiot vastasivat datapakettien saapumisesta sekä yhteyden katkeamistilanteista.

```
void acceptConnection(){
    if (client){
        QTcpSocket * tmpclient = server.nextPendingConnection();
        if (tmpclient){
            tmpclient->close();
        }
        qDebug() << "Port already in use.";
        return;
    }

    client = server.nextPendingConnection();

    if (!client)
        return;

    connect(client, &QTcpSocket::readyRead, buffer, &Buffer::readSocket);
    connect(client, &QTcpSocket::disconnected, this, &Server::on_client_disconnected);
    buffer->clientConnection = true;
}
```

Kuvio 12. Yhteyden muodostaminen AudioSource-ohjelmaan

5.8.3 Datan vastaanottaminen

Uuden datapaketin saapuessa QTcpSocket-luokan signaalifunktio aktivoitui, jolloin kutsuttiin readSocket-funktiota (ks. kuvio 13). Tässä funktiossa otettiin jokainen saapunut datapaketti vastaan QByteArray-luokan avulla. Kun ensimmäisen kerran vastaanotettiin dataa, alustettiin audiopuskurin aloitusaika ja käynnistettiin puskurin metadatalle tarkoitettu oma aikalaskuri. Samalla käynnistettiin myös ajastin, jonka kesto perustui konfiguraatiotiedostossa määriteltyyn audiopuskurin kokoon millisekunneissa. Kun tämä aika oli kulunut umpeen, aloitettiin audiopuskurin levylle kirjoituslogiikka. Tämän avulla audiopuskurin muistipuskuriin saatiin kerättyä esimerkiksi sekunnin verran dataa, jotta kirjoituksen alkaessa puskuri ei olisi heti tyhjä.

Tästä toiminnallisuudesta oli apua myös myöhemmässä vaiheessa, kun testattiin AudioSource-ohjelman avulla verkon jumiutumista eli viivettä datapakettien saapumisen välillä. Esimerkiksi audiota lähetettiin 200 ms:n välein, jolloin puskurin koko vaihteli määritellystä sekunnin koosta siten, että muistipuskurista ehdittiin tuon 200 ms:n viiveen aikana kirjoittaa pois 200 ms dataa. Täten muistipuskurin koko oli enää 800 ms. Tämän viivevaiheen jälkeen puskurin koko palautui lähelle haluttua muistipuskurin kokoa. Audiopuskurin muistissa tuli siis olla aina hieman dataa, että se selvisi verkossa tapahtuvista yhteysongelmista.

```
void Buffer::readSocket(){
    QByteArray data = d->server->readData();
    if (!data.isEmpty() && !d->firstDataReceived){
        d->bufferMeta.initStartTime(QDateTime::currentMSecsSinceEpoch());
        d->firstDataReceived = true;
        d->metaDataElapsedTimer.start();
        int32_t timerLength = d->bufferSizeInMilliseconds - d->bufferMeta.getDurationForBytes(data.size()) / d->conversionRate;
        if (timerLength < 0){
            timerLength = 0;
        }
        QTimer::singleShot(timerLength, Qt::PreciseTimer, this, &Buffer::startWriteLogic);
    }

    d->dataReceived = data.size();

    d->storageWriter->writeMetaDataFromReceivedData("Received", getDurationForBytesInMilliseconds(d->dataReceived),
                                                    d->metaDataElapsedTimer.elapsed(), getMemoryBufferSizeInMilliseconds());

    if (d->started){
        d->manager->onNewData(data, d->memoryBuffer.size(), d->bytesForBufferDuration, d->bufferMeta,
                                                                d->metaDataElapsedTimer.elapsed());
    }
    d->memoryBuffer.append(data);

    if (!d->memoryBufferWriteReady){
        if (d->bufferMeta.getDurationForBytes(d->memoryBuffer.size() / d->conversionRate) > d->bufferSizeInMilliseconds){
            d->memoryBufferWriteReady = true;
        }
    }
}
```

Kuvio 13. Datan vastaanottaminen QTcpSocketista

5.8.4 Yhteyden katkeaminen

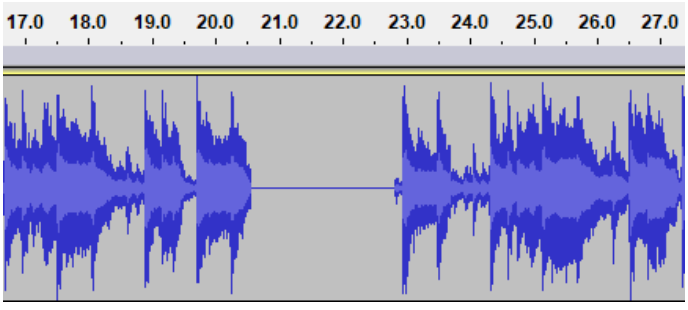
Yhteyden katkeamistilanteessa (ks. kuvio 14) audiopuskurin palvelin sai signaalin yhteyden katkeamisesta, minkä jälkeen se kutsui audiopuskurin yhteyden katkeamistilannetta käsittelevää funktiota, joka asetti muistipuskurin kirjoituslogiikan pois päältä. Yhteyden katketessa kirjoitettiin ensin olemassa olevan muistipuskurin sisältö tyhjäksi, minkä jälkeen tiedostoon kirjoitettiin nollaa eli tyhjää dataa. Yhteyden palautuessa audiopuskuri odotti jälleen, että muistipuskuriin kerääntyi haluttu määrä dataa, minkä jälkeen audion kirjoitusta voitiin jatkaa normaalisti.

```
void on_client_disconnected(){
    delete client;
    client = 0;
    buffer->clientConnection = false;
    buffer->onClientDisconnected();
    qDebug() << "Client disconnected from port: " << portnr;
}

void Buffer::onClientDisconnected(){
    d->memoryBufferWriteReady = false;
}
```

Kuvio 14. Yhteyden katkeaminen

Nollan kirjoittaminen näkyi audion kuvaajassa Audacity-ohjelmassa (ks. kuvio 15) tyhjänä kohtana ja kuunneltaessa vain hiljaisuutena. Aika kului taustalla koko ajan eteenpäin, mutta tyhjän datan kirjoittamisen avulla audiopuskurin aika-akseli ei vääristynyt, vaikkei uutta dataa vastaanotettukaan. Tämä tapahtuma on nähtävissä kuviossa 15, jossa ääniaallon keskikohdassa on vain ohut viiva, koska tällöin audioon oli kirjoitettu tyhjää dataa yhteyden katketessa.



Kuvio 15. Tyhjä kohta audiovirran kuvaajassa

5.9 Audion tallentaminen

5.9.1 Yleistä

Audion tallentaminen toteutettiin käyttämällä ajastettua kelloa, joka tässä tapauksessa oli 50 ms:n mittainen ajastin. Jokaisen 50 ms:n ajan hetken jälkeen kiintolevyllä olevaan tiedostoon kirjoitettiin dataa. Valittuun audioformaattiin perustuen tuli tätä varten laskea se, kuinka paljon dataa tulee kirjoittaa levylle tavuina, jos aikaa on kulunut esimerkiksi 50 ms ja audion näytteenottotaajuus on 16 000 hertziä. Tähän oli apuna QAudioFormat-luokan metodi, joka palautti tavumäärän perustuen formaatille määriteltyyn näytteenottotaajuuteen sekä metodille parametrina annettuun aika-arvoon.

Kirjoitettujen tavujen määrän avulla voitiin kirjoituksen jälkeen poistaa muistipuskurista haluttu määrä tavuja ja näin ollen siirtyä kirjoituksessa seuraavaan kohtaan puskurissa, kun samalla puskurin loppupäähän lisättiin uutta dataa.

5.9.2 Kirjoittamislogiikka

Kun ensimmäinen datapaketti oli vastaanotettu, aloitettiin ajastin, jonka aikana muistipuskuriin kerättiin haluttu määrä dataa. Tämän jälkeen levylle kirjoittamislogiikka aktivoitui ja kutsuttiin funktiota startWriteLogic (ks. kuvio 16). Tässä funktiossa luotiin Qt:n oma tarkka ajastin, jonka käsittelyfunktiota kutsuttiin 50 ms:n välein. Samalla alustettiin myös toinen ajastin aktivoitumaan 100 ms:n välein. Tämä ajastin oli tarkoitettu audion normalisoinnin tarkkailua varten, josta kerrotaan lisää luvussa 5.10.5. Samassa funktiossa käynnistettiin myös kuluvan ajan laskuri, jolta voitiin kysyä tarkka aika, millä tahansa hetkellä. Puskurin metatiedostoon tallennettiin audiopuskurin aloitusajankohta, joka tässä tapauksessa määritettiin tilanteeseen, jolloin vastaanotettiin ensimmäinen datapaketti.

```
void Buffer::startWriteLogic(){
    d->fileWriteTimerId = startTimer(50, Qt::PreciseTimer);
    d->continousDataCheckElapsedTimerId = startTimer(100, Qt::PreciseTimer);
    d->elapsedTimer.start();
    d->started = true;
    d->storageWriter->writeBufferInitMetaData(this->getBufferName(), d->bufferMeta);
}
```

Kuvio 16. Kirjoittamislogiikan aloittaminen

Kirjoitettavien tavujen määrä laskettiin 50 ms:n välein perustuen kuluvan ajan laskurin antamaan arvoon sekä audioformaattiin perustuvaan näytteenottotaajuuteen (ks. kuvio 17). Kuviosta 17 voidaan havaita, miten aluksi muistiin otettiin kulunut aika ajastimen aloittamisesta. Tämän jälkeen audiopuskurin metatietoa hyväksikäyttäen laskettiin kuluneeseen aikaan suhteutettuna, montako tavua tulisi kirjoittaa. Lopuksi vähennettiin tuosta arvosta jo siihen mennessä kirjoitettujen tavujen määrä, jolloin saatiin määrä, joka tuli kirjoittaa seuraavaksi.

```
int32_t Buffer::calculateBytesToWrite(){
    int64_t elapsedTime = d->elapsedTimer.elapsed();
    int32_t bytesToWrite = d->bufferMeta.getBytesForDuration(elapsedTime * 1000);
    bytesToWrite = bytesToWrite - d->writtenBytes;
    return bytesToWrite;
}
```

Kuvio 17. Kirjoitettavien tavujen määrän laskeminen

Qt:n ajastimien tapahtumien käsittely voitiin hallinnoida timerEvent-funktiossa (ks. kuvio 18), jossa tapahtumat käsiteltiin ajastimelle määritellyn Id:n perusteella. Kirjoituslogiikan ajastimessa tarkistettiin, että onko yhteys voimassa, minkä jälkeen varmistettiin se, että muistipuskuri oli kirjoitusvalmis eli siellä oli tarpeeksi dataa valmiina. Tämän jälkeen kirjoitettiin data levylle muistipuskurista funktiolla writeBufferData, jolle annettiin parametrina aiemmin laskettujen tavujen määrä.

Jos yhteyttä ei ollut, muistipuskuri kirjoitettiin tyhjäksi, jonka jälkeen jatkettiin kirjoittamalla tyhjää, kunnes yhteys palautui. Toisen ajastimen käsittelystä kerrotaan luvussa 5.10.5. Siinä tarkistettiin 100 ms:n välein, vastaanotetaanko liikaa dataa.

```
void Buffer::timerEvent(QTimerEvent * e) {
    if (e->timerId() == d->fileWriteTimerId){
        if (clientConnection) {
            if (d->memoryBufferWriteReady){
                writeBufferData(calculateBytesToWrite());
            }else{
                d->writtenBytes = d->writtenBytes + calculateBytesToWrite();
            }
        }else{
            writeBufferData(calculateBytesToWrite());
        }
    }

    if (e->timerId() == d->continousDataCheckElapsedTimerId){
        d->manager->onContinousDataCheck(d->memoryBuffer, d->bytesForBufferDuration,
            d->bufferMeta, d->metaDataElapsedTimer.elapsed());
    }
}
```

Kuvio 18. Ajastinfunktio säännölliseen kiintolevylle kirjoittamiseen

5.9.3 Kiintolevylle kirjoittaminen

Kiintolevylle kirjoitettaessa jokaiselle audiopuskurille luotiin oma RAW-tiedosto, johon audio tallennettiin. Raw tarkoittaa tiedostomuotoa, jossa dataa ei ole pakattu tai käsitelty mitenkään, vaan se on alkuperäisessä muodossaan tallennettu levylle.

Audiopuskurin alustuksen yhteydessä kutsuttiin funktiota initData (ks. kuvio 19), jossa luotiin puskuria varten oma kansio, jonne luotiin audiotiedosto sekä metatiedon tallennusta varten tekstitiedosto. Nämä tiedostot avattiin kirjoitusta varten audiopuskurille valmiiksi, eikä niitä suljettu erikseen kirjoituksen aikana, koska tämä olisi kuormittanut kiintolevyä turhaan, kun avattaisiin ja suljettaisiin sama tiedosto peräjälkeen lyhyen ajanjakson aikana.

```
bool BufferFileWriter::initData(const QString &bufferName){
    if (!QDir(path).exists()){
        QDir().mkdir(path);
    }
    if (!QDir(path + bufferName).exists()){
        QDir().mkdir(path + bufferName);
    }
    audioFile.reset(new QFile(path + bufferName + "\\audio.raw"));
    receivedMetaDataFile.setFileName(path + bufferName + "\\receivedMetaData.csv");


    if (!audioFile.isOpen()){
        if (!audioFile->open(QIODevice::Append)){
            return false;
        }
    }
    if (!receivedMetaDataFile.isOpen()){
        if (!receivedMetaDataFile.open(QIODevice::Append)){
            return false;
        }
    }

    return true;
}
```

Kuvio 19. Tiedostojen alustaminen kirjoittamista varten

Varsinaisessa audion levylle kirjoitusfunktiossa writeData (ks. kuvio 20) tarkistettiin, onko muistipuskurin koko pienempi kuin kirjoitettavien tavujen määrä. Jos näin oli, laskettiin, kuinka paljon pitää kirjoittaa tyhjää dataa, jotta levylle saadaan kuitenkin tarvittava määrä dataa. Tämän jälkeen luotiin QByteArray-luokan avulla tarvittava määrä tyhjää dataa ja kirjoitettiin muistipuskurissa oleva data ensin pois, minkä jälkeen sen perään kirjoitettiin tarvittu määrä tyhjää dataa eli hiljaisuutta.

Normaalissa tapauksessa kirjoituslogiikan 50 ms:n kierroksen aikana laskettu kirjoitettavan datan määrä oli huomattavasti pienempi kuin muistipuskurin koko, jolloin voitiin edetä tapaukseen, jossa kirjoitettiin laskettu määrä dataa muistipuskurista pois. Näin ollen kiertokulku jatkui ja dataa tuli muistipuskurin loppupäähän koko ajan lisää ja alkupäästä sitä kirjoitettiin pois. Lopuksi funktio palautti vielä kutsuvalle audiopuskurille kirjoitettujen tavujen määrän, jotta audiopuskuri pystyi pitämään kirjaa kirjoitetun datan määrästä. Tätä tietoa se käytti kirjoitettavien tavujen laskemiseen.

```
int32_t BufferFileWriter::writeData(QByteArray &memoryBuffer, int32_t bytesToWrite){

    int32_t bytesToWriteEmpty = 0;
    int32_t writtenBytes = 0;

    if (memoryBuffer.size() < bytesToWrite){
        bytesToWriteEmpty = bytesToWrite - memoryBuffer.size();
        QByteArray empty(bytesToWriteEmpty, 0);
        writtenBytes = audioFile->write(memoryBuffer);
        writtenBytes += audioFile->write(empty);
    }else{
        writtenBytes = audioFile->write(memoryBuffer.data(), bytesToWrite);
    }

    return writtenBytes;
}
```

Kuvio 20. Audion kirjoittaminen muistipuskurista kiintolevylle

Audiopuskurin kirjoituslogiikassa jokaisen 50 ms:n aikasyklin aikana kutsuttiin funktiota writeBufferData (ks. kuvio 21), jossa kutsuttiin aiemmin kuvattua datan kirjoitusfunktiota. Tässä funktiossa hallittiin kirjoituksen jälkeinen tilanne, jossa muistipuskuri tuli kokonaan tyhjätä, mikäli muistipuskurin koko oli pienempi kuin kirjoitettujen tavujen määrä. Normaalissa tapahtumakierrossa muistipuskurin alkukohdasta poistettiin dataa kirjoitettujen tavujen verran. Funktiossa myös tallennettiin metadataa puskurin tilasta datan kirjoituksen jälkeen, josta kerrotaan enemmän luvussa 5.11. Funktion lopussa kirjoitettujen tavujen kirjanpitoa vielä päivitettiin.

```
void Buffer::writeBufferData(int32_t bytesToWrite){
    int32_t writtenBytes =  d->storageWriter->writeData(d->memoryBuffer, bytesToWrite);
    if (writtenBytes > d->memoryBuffer.size()){
        d->memoryBuffer.clear();
    }else{
        d->memoryBuffer.remove(0, writtenBytes);
    }

    d->storageWriter->writeMetaDataFromReceivedData("Written",
        getDurationForBytesInMilliseconds(writtenBytes),
        d->metaDataElapsedTimer.elapsed(), getMemoryBufferSizeInMilliseconds());
    d->writtenBytes = d->writtenBytes + writtenBytes;
}
```

Kuvio 21. Audiopuskurin kiintolevylle kirjoitusfunktio

5.10 Audion normalisointi audiopuskurin aikakäsitykseen

5.10.1 Yleistä

Tämän opinnäytetyön ydinongelman ratkaiseminen ja päätutkimuksen aiheena oli oikeaoppinen audion normalisointi ja reagoiminen siihen, mikäli audiolähteiden aikakäsitys erosi vastaanottavan audiopuskurin aikakäsityksestä. Normalisoinnilla tässä kontekstissa tarkoitettiin audiovirtojen aikakäsityksen normalisointia vastaamaan audiopuskurin aikakäsitystä. Tämä toteutettiin käytännössä muokkaamalla vastaanotettua audiota. Aiemmin kuvatut rakenteet ja toteutetut toiminnallisuudet audiotallentimessa ja audiopuskurissa olivat opinnäytetyön vaatimusmäärittelyssä määritelty runko, jonka päälle oli helpompi toteuttaa varsinainen prototyyppiratkaisu tutkittavana olleeseen ongelmaan.

AudioSource-ohjelmassa laskettiin lähetettävään näytteenottotaajuuteen perustuen, paljonko dataa tuli lähettää 50 ms:n aikajakson aikana. Vastaanottavassa audiopuskurissa toteutettiin kirjoituslogiikka vastaamaan tuota 50 ms:a. Dataa siis kirjoitettiin juuri se määrä pois muistipuskurista, mitä sitä oli vastaanotettukin. Tällöin kiertokulku toimi siten, että dataa vastaanotettiin kyseisessä aikayksikössä aina saman verran, kun sitä kirjoitettiin pois. Tämä johti tilanteeseen, jolloin muistipuskurin koko pysyi lähes samana koko ajan. Reaalimaailman verkkoympäristössä tähän vaikuttaisivat tietysti verkon mahdolliset jumiutumiset ja pätkimiset. Tätä ilmiötä havainnollistettiin siten, että AudioSource-ohjelma lähetti satunnaisella viiveellä dataa, eikä säännöllisesti ja kellontarkasti.

Edellä kuvattuun tapaukseen haluttiin saada muutosta siten, että audiolähteen mahdollinen ero aikakäsityksessä audiopuskuriin voitiin havainnollistaa. Tämä tapahtui kasvattamalla lähetettävän audion näytteenottotaajuuden arvoa. Oikeassa toimintamallissa audiolähteen käsitys sekunnista voisi esimerkiksi olla aavistuksen pitempi kuin audiopuskurin. Tässä tilanteessa audiolähde lähettäisi tuon olettamansa 50 ms:n aikajakson aikana hieman enemmän dataa tietyllä näytteenottotaajuudella, kuin audiotallennin olettaisi saavansa. Tämän ongelman ratkaisemiseksi tuli toteuttaa muutama ratkaisuvaihtoehto, joiden pohjalta voitiin havainnoida ja perustella toteutetun toiminnallisuuden ja opinnäytetyön tutkimustyön tulokset.

Ongelman ratkaisussa tuli myös huomioida se, ettei normalisointilogiikkaa voinut aktivoida liian herkästi, koska tämä johti osassa tapauksista siihen, että verkko oli ajoittain hieman hitaampi, jolloin muistipuskuriin vastaanotettiinkin isompi purske dataa, jolloin normalisointilogiikka aktivoitui virheelliseen tietoon perustuen. Ongelman ratkaisemiseksi tuli logiikan aktivointiin perustuvaa raja-arvoa kasvattaa suuremmaksi, jotta verkon aiheuttamat yhteysongelmat eivät sotkeneet normalisointia.

5.10.2 GStreamer

GStreamer on ohjelmistokirjasto, jolla voidaan luoda audio ja video suoratoistosovelluksia. Yleensä GStreameriä käytetään mediasoittimien luomiseen, ja näin ollen siinä onkin laaja tuki eri audio - ja videoformaattien käsittelyyn. Tärkeimpänä lisäyksenä ovat siihen liitettävät liitännäiset, joiden avulla voidaan hallita monimutkaisempia toiminnallisuuksia ja luoda monipuolisempia sovelluksia. (What is GStreamer? n.d.)

GStreamer tarjoaa ratkaisua ongelmaan, jossa lähettävän osapuolen ja vastaanottajan välinen aikaero halutaan synkronoida samaksi. GStreamerissa on aikaeron korjaamiseksi olemassa kaksi tapaa. Ensimmäinen on niin sanottu näytteenottotaajuuden uudelleen näytteistäminen. Tämä tapahtuu siten, että tarvittaessa näytteitä lisätään tai niitä poistetaan. Toisessa tapauksessa näytteitä siirretään eteenpäin ja tämän avulla yritetään synkronoida aikaeroa. (Lauri & Malmgren 2015, 30-31.)

GStreamer kirjaston tarjoamia ominaisuuksia ja hyötyjä mietittiin, kun lähdettiin toteuttamaan audion normalisointia. GStreamer ei kuitenkaan tarjonnut ratkaistavaan ongelmaan suoraan apua, vaan sen käytön lisäksi olisi täytynyt toteuttaa lisätoiminnallisuutta ja integroida se sitten GStreamerin valmiiden funktioiden kanssa.

Laurin ym. (2015, 30-31) mukaan GStreamerin ensimmäinen tapa korjata ongelmaa on liian aggressiivinen halutun toiminnallisuuden toteuttamiseen, mikä olisi ollut kehitystyössä enemmän haitaksi. Toisessa tavassa näytteiden hyppääminen yli olisi saattanut johtaa tilanteeseen, jossa eri lähteiden aikaero olisi saatu normalisoitua, mutta jotain olennaista olisi luultavasti jäänyt pois tallennetusta audiosta. Näiden seikkojen vuoksi GStreamerin käyttö koettiin johtavan hieman sivuraiteille, joten audion käsittely päätettiin toteuttaa ilman ulkoisia apukirjastoja. Käsittelylogiikka perustui tutkimustyön

aikana saatuihin faktoihin siitä, että audiota muokattaessa siitä tulee poistaa kokonaisia näytteitä. Muutoin audiosta tulee epäselvää ja säröilevää.

5.10.3 Normalisointi

Audion normalisoinnin ratkaisemiseen yhteiseen aikakäsitykseen ei ole olemassa yhtä oikeaa tapaa, ja siksi tässä opinnäytetyössä tarkoituksena olikin tutkia vaihtoehtoja ja tuottaa muutama tapa ratkaista kyseinen ongelma ja mahdollisesti kehittää niistä paras vaihtoehto jatkokehitykseen. Opinnäytetyössä etsittiin valmiita ratkaisuvaihtoehtoja, joita kuitenkaan juurikaan ei löytynyt, tai ne eivät olisi auttaneet ratkaistavaan ongelmaan. Näin ollen ratkaisut ongelmiin toteutettiin perustuen tutkimukseen audion käsittelystä ja näytteiden muokkaamisesta. Testauksessa huomattujen virheiden johdosta voitiin päätellä, oltiinko audiota muokattu oikein ja siten voitiin muuttaa audiotallentimen toiminnallisuutta oikeaan suuntaan. Esimerkiksi tilanteessa, jossa yhteys katkesi, muistipuskurissa ollut data tuli kirjoittaa sieltä heti pois, eikä vasta yhteyden palautuessa, jolloin aika eteni oikeassa suhteessa vastaanotettuun dataan nähden.

5.10.4 Audion normalisointilogiikan aktivointi

Audiopuskurin alustuksen yhteydessä sille alustettiin manageri, joka tarvittaessa aktivoi audion normalisointilogiikan, jos tietty raja-arvo ylitettiin. Tämä prosenttiarvo asetettiin manuaalisesti konfiguraatiotiedostoon. Tähän arvoon perustuen audiopuskuri käynnistettiin, ja manageri tarvittaessa aktivoi valitun normalisointistrategian. Funktiossa triggerOverFlowLogic (ks. kuvio 22) laskettiin, kuinka paljon muistipuskurin koko eroaa halutusta muistipuskurin koosta. Tähän arvoon perustuen laskettiin prosenttiosuus, jolla se ylitti halutun muistipuskurin koon, ja tätä arvoa verrattiin konfiguraatiotiedostossa olleeseen arvoon. Jos tämä arvo ylitettiin, aktivoitiin valittu strategia.

```
bool BufferOverflowManager::triggerOverFlowLogic(int32_t memoryBufferSize, int32_t bytesForBufferDuration){
    float difference = memoryBufferSize - bytesForBufferDuration;
    float result = difference / memoryBufferSize * 100;

    return result >= d->overflowPercentage;
}
```

Kuvio 22. Normalisointilogiikan aktivointi

5.10.5 Strategiat audion normalisointiin

Audion normalisointiin toteutettiin kaksi strategiaa, joilla pystyttiin hallitsemaan tilannetta, jossa audiolähde lähetti dataa liikaa, eli sillä oli eri aikakäsitys. Käytännössä tämä ilmeni siten, että vastaanotettujen datapakettien koko oli suurempi kuin normaalisti. Tämä taas heijastui siihen, että muistipuskuriin kerääntyi dataa enemmän, kuin sieltä ehdittiin poistaa. Ajan kuluessa tämä johti tilanteeseen, jossa muistipuskurin koko kasvoi lineaarisesti ylöspäin halutusta puskurin koosta. Tämä havainnollistettiin lähettämällä tahallisesti enemmän dataa AudioSource-ohjelman avulla käyttämällä suurempaa näytteenottotaajuutta, kuin audiopuskuri oletti.

Ensimmäinen strategia tarkkaili 100 ms:n välein, miten puskurin koko kasvaa. Jos valittu prosenttiarvo ylitettiin, aktivoitiin datan pudotusstrategia (ks. kuvio 23). Funktiossa onContinousDataCheck laskettiin ylimääräisten tavujen määrä muistipuskurissa eli määrä, joka tulisi poistaa puskurista, jotta muistipuskurin koko saataisiin korjattua kohti haluttua määrää. Tämän jälkeen tuli selvittää valittuun audioformaattiin perustuen, montako tavua on yksi näyte audiossa. Audio koostuu näytteistä, jotka esittävät digitaalista audiota sekunnin aikana, jolloin puhutaan näytteenottotaajuudesta. Mitä suurempi näytteiden määrä sekunnissa on, sitä tarkempi digitaalinen tallenne se on audiosta.

Esimerkiksi audion yhden näytteen koko oli 16 bittiä, jolloin yksi näyte tavuina oli 2 tavua. Kun dataa jouduttiin poistamaan, luonnollisesti jotain olennaista saattoi kadota audiosta. Jos yhdestä kohdasta poistettiin merkittävä määrä dataa, ei voitu välttämättä olla varmoja siitä, mitä lähettävän audiolähteen audio tarkalleen sisälsi. Tämä ongelma nousi esille, kun audiosta poistettiin epämääräinen näytteen kokoon perustumaton määrä dataa. Tällöin Audacity-ohjelmassa tarkastellussa audiossa oli selvästi kuultavissa pätkimistä ja säröilyä. Tämän ongelman minimoimiseksi tuli dataa pudottaa kokonaisina näytteinä ja monesta eri kohdasta, ettei audio hajoaisi niin selkeästi, että se on ihmiskorvalla mahdollista havainnoida.

Strategiassa laskettiin, montako kertaa tuo yhden näytteen kokoinen tavumäärä tuli poistaa muistipuskurista, jotta kaikki ylimääräinen data on hävitetty. Laskettiin myös siirtymän pituus pudotuskohtien välille, jotta datan poistaminen jaoteltiin tasaisesti

koko muistipuskurin matkalle. Tätä silmukkaa käytiin läpi niin kauan, kunnes koko muistipuskuri oli käyty läpi ja tarvittu datamäärä oli poistettu puskurista.

```
void BufferOverflowStrategy1::onContinousDataCheck(QByteArray &memoryBuffer, int32_t bytesForBufferDuration,
                                                   const BufferMeta &bufferMeta){
    int32_t dataToDrop = memoryBuffer.size() - bytesForBufferDuration;

    int32_t oneSampleInBytes = bufferMeta.getSampleSize() / 8;

    int32_t timesToDropData = dataToDrop / oneSampleInBytes;
    int32_t dataToMoveForwardOnBuffer = memoryBuffer.size() / timesToDropData;
    for (int i = 0; i <= memoryBuffer.size();){

        if (dataToMoveForwardOnBuffer % oneSampleInBytes != 0){
            int32_t remainder = dataToMoveForwardOnBuffer % oneSampleInBytes;
            dataToMoveForwardOnBuffer = dataToMoveForwardOnBuffer + remainder;
        }
        memoryBuffer.remove(i, oneSampleInBytes);

        i = i + dataToMoveForwardOnBuffer;

    }
}
```

Kuvio 23. Audion normalisointistrategia: Jatkuva tarkastelu

Toisessa strategiassa tarkoituksena oli aktivoida normalisointilogiikka samaan tapaan, kuin edempänä kuvattu tapaus sillä poikkeuksella, että vastaanotettuihin datapaketteihin reagoitiin välittömästi, joko poistamalla koko datapaketti (ks. kuvio 24) tai osa siitä. Tämä tarkoitti käytännössä sitä, että muistipuskuriin ei lisätty ollenkaan vastaanotettua datapakettia, vaan se käytännössä heitettiin heti pois.

Tässä strategiassa ei ollut yhtä paljon logiikkaa kuin edellisessä. Siihen voisi jatkokehityksessä lisätä esimerkiksi mahdollisuuden siihen, että ympärillä oleva logiikka pudottaisi saapuvia datapaketteja pitemmältä aikajaksolta, jolloin datan puuttuminen audiossa ei olisi niin selvästi kuultavissa. Molemmat toteutetut strategiat toteuttavat halutun päämäärän eli ne tarkkailevat muistipuskurin tilaa, eikä muistipuskuri pääse näin ollen holtittomasti kasvamaan. Eri tilannetapauksista luvassa tuloksia luvussa 6.2 kuvaajien muodossa.

```
void BufferOverflowStrategy2::onNewData(QByteArray &data){
    data.remove(0, data.size());
}
```

Kuvio 24. Audion normalisointistrategia: Saapuneen datapaketin tarkastelu

5.10.6 Yhteenveto

Lopputuloksena toteutetusta audion normalisoinnista voidaan sanoa se, että haluttu toiminnallisuus ja ongelmaan toteutetut ratkaisut toimivat halutulla tavalla. Strategioiden hienosäätö ja muokkaaminen yhä älykkäämpään suuntaan olisivat jatkokehitykseen kohdistettavia asioita. Hyvin suunnitellun ja suunnittelua kohti toteutetun audiotallentimen rungon päälle oli huomattavasti helpompi aloittaa toteuttamaan ja tutkimaan sitä, miten audiota tulisi oikeaoppisesti käsitellä. Käytössä ollut audioformaatti ja QAudioFormat-luokasta saatujen metodien avulla laskutoimituksien toteuttaminen oli suoraviivaista ja helppoa.

Testausvaiheessa toteutettiin useita erilaisia kokeiluja datan pudotuslogiikan suhteen, kunnes saatiin toteutettua järkevä kokonaisuus, joka toimi oikealla tavalla. Toteutetuista strategioista on selvää, että ensimmäinen versio oli parempi ja monipuolisempi, koska se keskittyi audion käsittelemiseen pidemmällä aikavälillä. Audiota muokattaessa väärästä kohdasta, jossa audionäyte jakautui kahtia, oli mahdollista havaita audiovirran kuvaajassa esiintynyt paksu pystyviiva. Kuunneltaessa audiota pystyviivan kohta havainnoitiin särisevänä äänenä. Tähän ongelmaan törmättiin useasti, kunnes pudotuslogiikan laskutoimitukset saatiin toteutettua siten, että näytteitä poistettiin oikeista kohdista kokonaisina kappaleina.

5.11 Audiopuskurin metadata

5.11.1 Yleistä

Audiopuskuriin vastaanotetusta datasta ja siihen tapahtuvista muutoksista haluttiin tallentaa metatietoa, josta myöhemmin pystyttiin luomaan tarvittavat kuvaajat eri tilanteisiin liittyen. Näiden kuvaajien avulla pystyttiin visuaalisesti todistamaan havaitut tulokset, joista luvassa tarkemmin luvussa 6.2. Metadataa kerättiin seuraavissa tilanteissa: datan vastaanottaminen, datan tallentaminen kiintolevylle ja audion normalisointi. Näistä tilanteista haluttiin saada metadataa tallennettua tiedostoon, jotta siitä voitiin myöhemmin tarkistaa audiopuskurin toiminnallisuus eri tilanteissa. Tämä tapahtui vertaamalla tallennettuja arvoja siihen, mitä eri tilanteissa haluttiin tapahtuvan.

5.11.2 Metadatan tallentaminen

Metadatan tallennus toteutettiin tallentamalla data CSV-tiedostoon. Tämä tiedostomuoto valittiin, koska sen avulla oli helppo tuoda arvot taulukkolaskentaohjelmaan, joka tässä projektissa oli LibreOfficen Calc. Tiedostoon talletetut arvot erotettiin toisistaan pilkuilla, minkä johdosta Calc osasi erotella arvot omiin soluihinsa allekkain taulukon muotoon.

Tallentaminen toteutettiin käyttämällä Qt:n QTextStream-luokkaa, jonka avulla voitiin tallentaa tämän tyyppistä tekstidataa kätevästi (ks. kuvio 25). Tiedostoon tallennettiin tieto kyseessä olleesta operaatiosta, aikalaskurin sen hetkinen aika, vastaanotetun datapaketin koko ja muistipuskurin sen hetkinen koko. Koot tavuina muutettiin vastaamaan näytteenottotaajuuteen perustuvaa aikaa millisekunteina kyseisestä datasta. Tämän avulla voitiin helpommin verrata ja havainnoida dataa, kun kaikki arvot olivat samassa yksikössä. Kuvioista tuli myös helpommin ymmärrettäviä, kun käsiteltiin sekuntiarvoja eikä tavuarvoja.

```
void BufferFileWriter::writeMetaDataFromReceivedData(QString operation, int32_t durationForDataSize,
                                                     int32_t elapsedTime, int32_t durationForMemoryBufferSize){
    if (!receivedMetaDataFile.isOpen()){
        return;
    }

    QTextStream out(&receivedMetaDataFile);
    if (receivedMetaDataFile.size() == 0){
        out << "Operation" << "," << "Elapsed time" << "," << "Data duration" << "," << "Memorybuffer duration\n";
    }

    out << operation << "," << elapsedTime  << "," << durationForDataSize << "," << durationForMemoryBufferSize << "\n";
}
```

Kuvio 25. Metadatan tallentaminen tiedostoon

Tekstitiedostosta Calc-ohjelman avulla taulukkomuotoon muutetusta datasta on nähtävissä, miten muistipuskurin koko kehittyy suhteessa kuluneeseen aikaan sekä verrattaessa vastaanotettujen datapakettien kokoa (ks. kuvio 26). Datapakettien koko pysyy normaalissa tapauksessa hyvin lähellä haluttua 50 ms, joka kuviosta 26 on nähtävissä. Myös muistipuskurin koko kasvaa hyvin samalla tavalla eli 50 ms:n kasvulla. Jos dataa lähetettiin enemmän tai viiveellä, oli se havaittavissa datapakettien saapumisajoissa sekä muistipuskurin koon isompana vaihteluna ylös ja alas. Nämä tapaukset kuvaajien muodossa ovat nähtävissä edempänä esitettävissä tuloksissa luvussa 6.2.

Operation	Elapsed time	Data duration	Memorybuffer duration
Received	12	50	0
Received	49	50	50
Received	100	50	100
Received	150	50	150
Received	200	50	200
Received	249	49	250
Received	300	51	299
Received	350	50	350
Received	400	50	400
Received	450	50	450
Received	500	49	500
Received	550	51	549
Received	600	50	600
Received	650	50	650
Received	700	50	700
Received	750	50	750
Received	799	49	800
Received	849	50	849
Received	900	51	899
Received	954	50	950
Received	1000	50	1000

Kuvio 26. Tallennettu metadata muistipuskurin tilasta taulukkomuodossa

5.11.3 Johtopäätökset

Metadatan tallennus oli siinä mielessä ratkaisevana tekijänä opinnäytetyön onnistumisen kannalta, että datan avulla saatiin luotua visuaalisia tuloksia eri tilanteista ja ratkaistuista ongelmista. Muutoin tulosten havainnollistaminen ja selventäminen olisi ollut hankalampaa. Datasta oli myös merkittävää apua testauksessa, kun tutkittiin logiikan toimivuutta ja tarkasteltiin vastaanotettujen datapakettien ja muistipuskurin koon vaihtelua. Voitiin esimerkiksi havainnoida tilanne, jossa data muistipuskurissa alkoi kasvaa lineaarisesti. Tallennetuissa arvoissa oli nähtävissä, kuinka muistipuskurin koko vaihteli normalisointistrategian aktivoituessa. Datan pudotuksesta ja datan kirjoituksesta kiintolevylle tallennettiin myös metadataa samaan tiedostoon. Datan pudotusarvosta oli hyvin nähtävissä se, että oikea määrä dataa poistettiin puskurista verrattuna muistipuskurin sen hetkiseen kokoon. Myös kirjoitetun datan määrä edesauttoi kirjoituslogiikan toiminnallisuuden varmentamisessa.

6 Tulokset

6.1 Tulosten koostaminen

Tulokset ovat tutkimustyön keskeisin osa-alue, josta ilmenevät toteutetusta toiminnallisuudesta havaitut hyödyt sekä päätelmät. Tulokset niputtavat yhteen tutkimustyön tuloksena saadut ratkaisut, jotka esitetään kontekstiin sopivassa muodossa. Niistä ilmenevät tutkimustyön aikana saavutetut onnistumiset, ja niiden pohjalta voidaan arvioida myös koko työn onnistuminen. Tässä opinnäytetyössä tutkimustyön tulokset mallinnettiin kuvaajien muodossa tallennetusta metadatasta, koska niiden avulla oli kaikkein helpointa havainnollistaa muistipuskurin käyttäytyminen eri tilanteissa. Niistä voitiin havainnoida eri ajanhetkillä olevat tapahtumat ja peilata niitä tilanteiseen, joita yritettiin ratkaista. Tulosten osa-alueita opinnäytetyössä olivat normaali tilanne muistipuskurissa, lähetysviive muistipuskurissa, yhteyden katkeamiskohta muistipuskurissa, muistipuskurin kasvaminen sekä varsinaisen opinnäytetyön ongelman ratkaisu eli tilanne, jossa muistipuskuri normalisoidaan sen kasvaessa liian paljon.

Näistä tapauksista tehtiin tarvittavat kokeilut audiotallentimessa käyttäen Audio-Source-ohjelmaa apuna. Muistipuskurin käyttäytymistä testattiin ennalta määritellyissä kuvitelluissa tapauksissa, joita voisi oikeassa verkkoympäristössä tapahtua. Näiden tapausten pohjalta saadun metadatan avulla varmennettiin tulosten paikkansa pitävyys sekä luotiin tarvittavat kuvaajat havainnollistamaan asian.

6.2 Metadatan kuvaajat

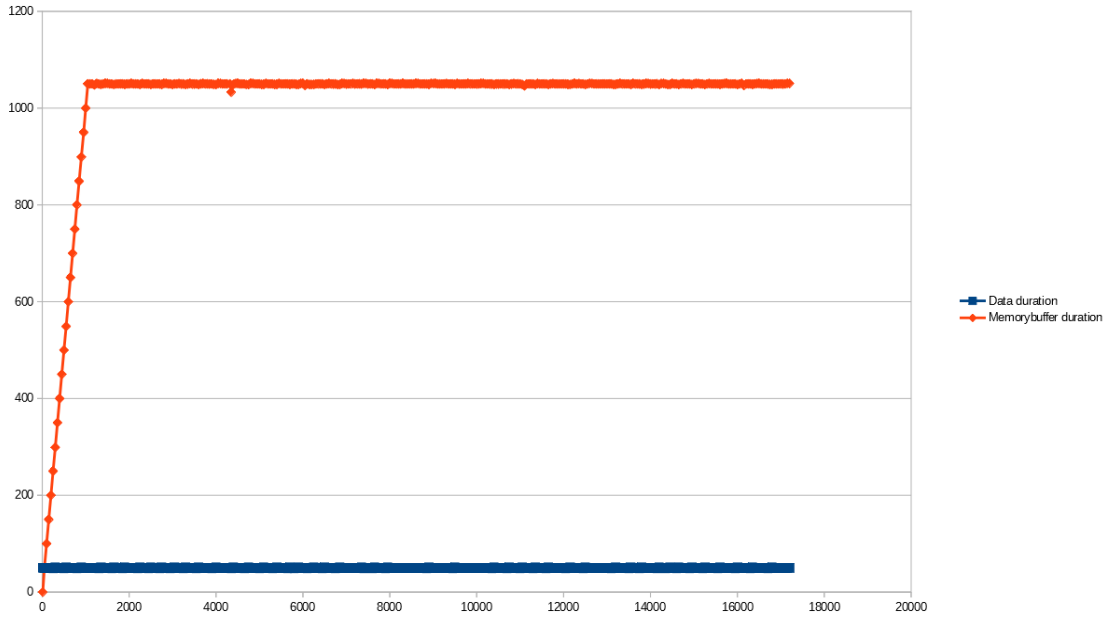
6.2.1 Yleistä

CSV-tiedostoon tallennetut arvot voitiin tuoda Calc-ohjelmaan allekkain taulukkomuotoon. Tallennetusta metadatasta voitiin taulukkolaskentaohjelmalla mallintaa tarvittavat kuvaajat, jotka kuvasivat eri tilanteita muistipuskurissa. Kuvaajissa lukuarvot olivat millisekunteina pysty- ja vaaka-akseleilla. Seuraavissa luvuissa 6.2.2-6.2.6 tullaan käymään nämä eri tilanteet yksityiskohtaisemmin läpi ja perustelemaan, miksi kuvaaja on

juuri tietynlainen. Punainen käyrä kuvaajissa kuvaa muistipuskurin kokoa ja sininen käyrä vastaanotettujen datapakettien kokoa.

6.2.2 Normaali tilanne muistipuskurissa

Normaalissa tilanteessa datan määrä muistipuskurissa nousi määrättyyn muistipuskurin arvoon, kunnes sieltä alettiin kirjoittaa dataa pois (ks. kuvio 27). Kirjoituksen aikana vastaanotettiin uutta dataa, kun sitä kirjoitettiin samalla pois. Tässä ihanteellisessa tilanteessa muistipuskurin koko pysyi lähes vakiona koko ajan, eikä se juurikaan vaihdellut kumpaankaan suuntaan.

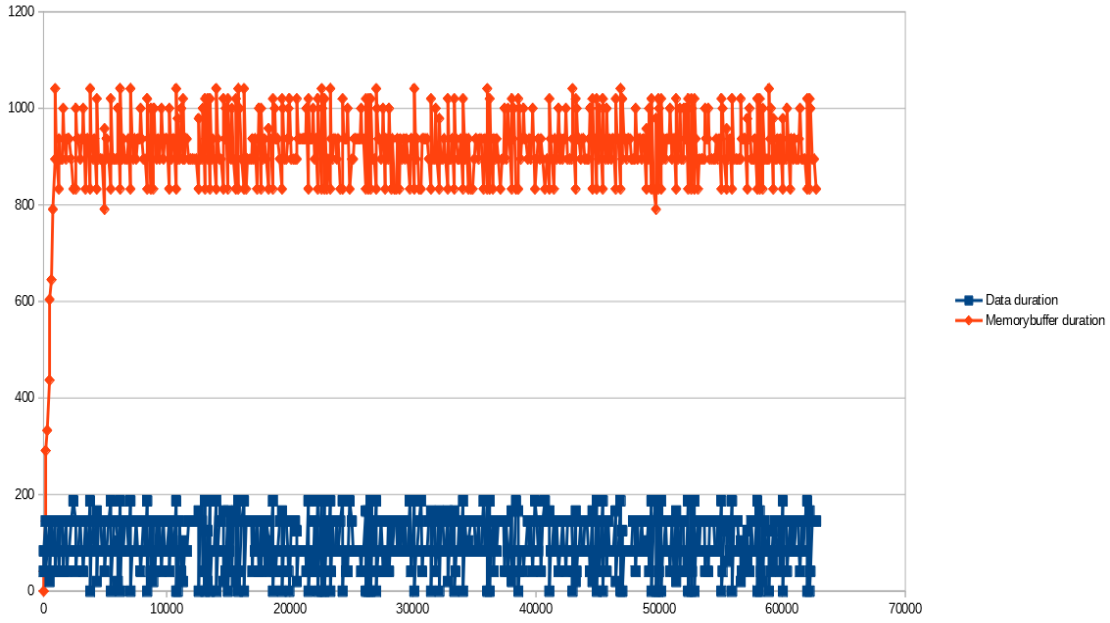


Kuvio 27. Normaali tilanne muistipuskurissa

6.2.3 Lähetysviive muistipuskurissa

AudioSource-ohjelmalla oli mahdollista lähettää datapaketteja esimerkiksi vaihteluvälillä 50 ms-200 ms. Tämä tarkoitti sitä, että dataa lähetettiin satunnaisesti 50 ms:n ja 200 ms:n väliltä. Välillä datapaketti vastaanotettiin normaalisti 50 ms:n välein ja välillä vasta esimerkiksi 180 ms:n päästä. Tämä näkyi muistipuskurin kuvaajassa siten, että dataa ehdittiin kirjoittamaan pois muistipuskurista paljon enemmän kuin normaalisti, kunnes uutta dataa vastaanotettiin. Tämä oli havainnoitavissa muistipuskurin kuvaajassa (ks. kuvio 28) vaihteluna 1000 ms:n ja 800 ms:n välillä. Samoin myös vastaanote-

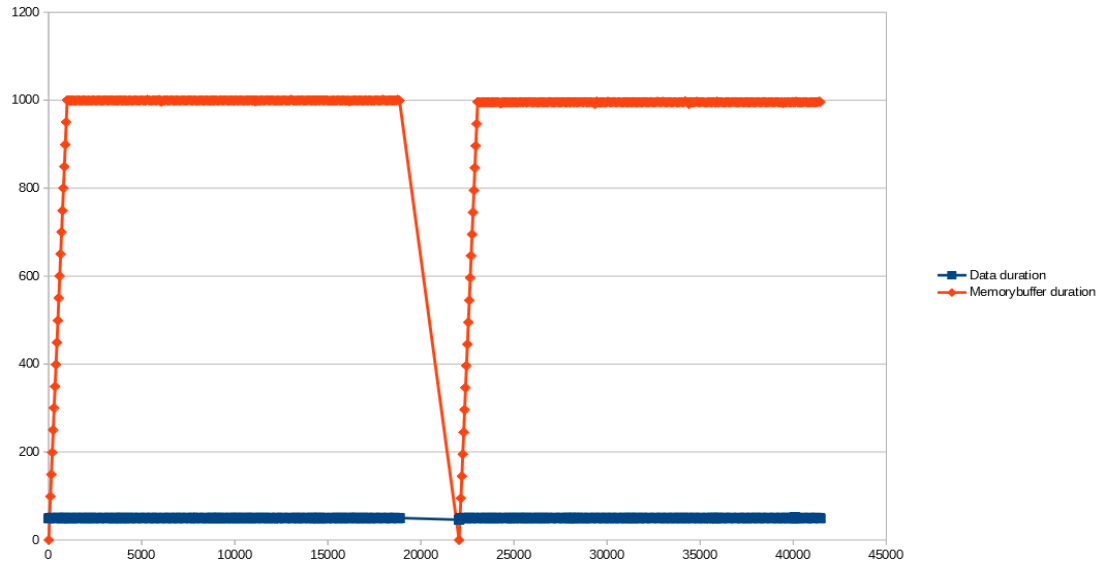
tut datapaketit olivat isompia, kun viivettä oli enemmän. Tämä havainnollistaa tilanteen, jossa verkko ei ole ihanteellinen, vaan tietoliikenteessä esiintyy jumiutumista ja hidastumista.



Kuvio 28. Lähetysviive muistipuskurissa

6.2.4 Yhteyden katkeamiskohta muistipuskurissa

Jos yhteys audiolähteeseen katkesi, oli aluksi hallittava muistipuskurissa olleen datan käsittely (ks. kuvio 29). Yhteyden katketessa olemassa oleva muistipuskuri kirjoitettiin ensin tyhjäksi, minkä jälkeen jatkettiin kirjoittamalla tyhjää dataa kiintolevyllä olleeseen tiedostoon. Yhteyden palautuessa kirjoittamista muistipuskurista ei jatkettu heti,

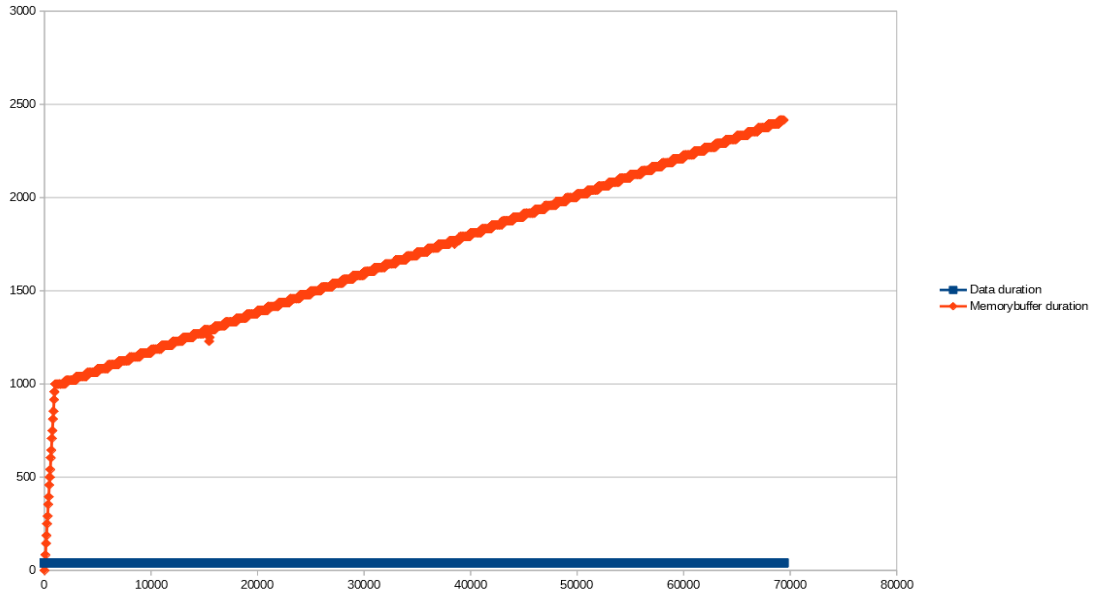


vaan ensin kerättiin dataa haluttu määrä, kunnes kiintolevylle kirjoittamista voitiin jatkaa normaalisti.

Kuvio 29. Yhteyden katkeamiskohta muistipuskurissa

6.2.5 Muistipuskurin kasvaminen

Jos normalisointilogiikkaa ei ollut käytössä, muistipuskurin koko alkoi kasvaa lineaarisesti ylöspäin (ks. kuvio 30). Tämä johtui siitä, että muistipuskurin ja audiolähteen aikakäsitys erosivat toisistaan. Tähän ongelmaan tässä opinnäytetyössä haluttiin nimenomaan toteuttaa ratkaisu. Toteutetun ratkaisun tarkoituksena oli tarkkailla puskurin koon kehitystä ja tarvittaessa reagoida siihen sekä vähentää muistipuskurissa ollut ylimääräinen määrä dataa.

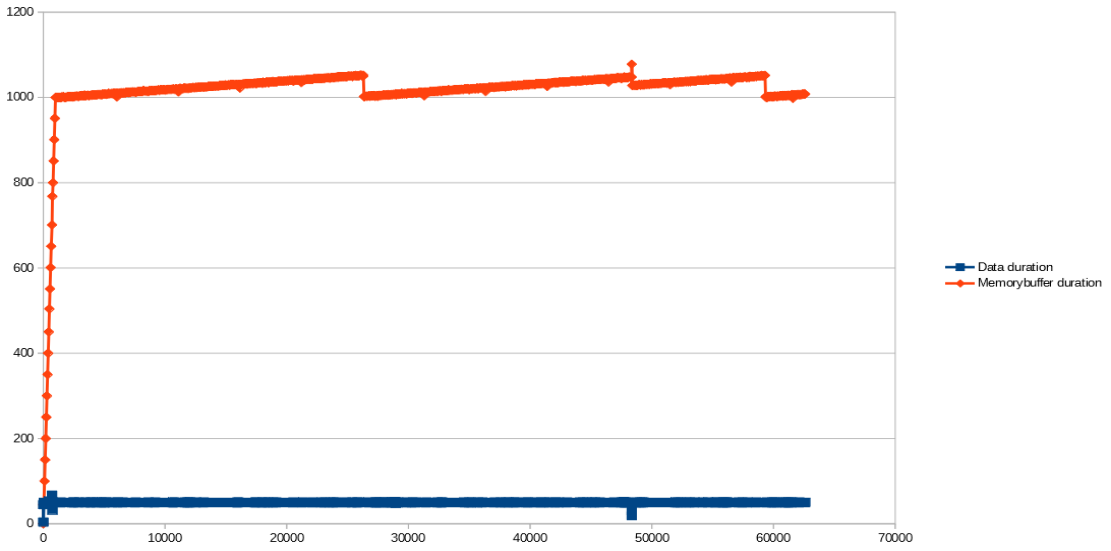


Kuvio 30. Muistipuskurin kasvaminen lineaarisesti

6.2.6 Normalisointi muistipuskurin kasvaessa

Opinnäytetyön ydinongelman ratkaisuun kehitetty normalisointilogiikan käyttäytyminen muistipuskurissa on nähtävissä metadatan kuvaajasta (ks. kuvio 31). Kuvaajasta voidaan huomata, että alussa muistipuskurin koko kasvaa haluttuun arvoon saakka, kunnes datan kirjoittaminen aloitetaan. Sen jälkeen muistipuskurin käyrä alkaa hiljal-

leen kasvaa siihen pisteeseen saakka, että normalisointilogiikka aktivoitui. Tämän jälkeen laskettiin ylimääräisen datan määrä puskurissa ja poistettiin se. Strategiana tässä tapauksessa käytettiin jatkuvaan tarkasteluun perustuvaa logiikkaa. Kuvaajasta on havaittavissa, että dataa pudotettiin välittömästi logiikan aktivoituessa tarvittava määrä, jotta muistipuskurin koko palaisi ennalta määrättyyn arvoon. Kuvaajasta voidaan myös nähdä, että tämä datan pudotus kiertokulku jatkui myöhemmin samaan tapaan niin kauan, kun AudioSource-ohjelma lähetti dataa liian paljon.



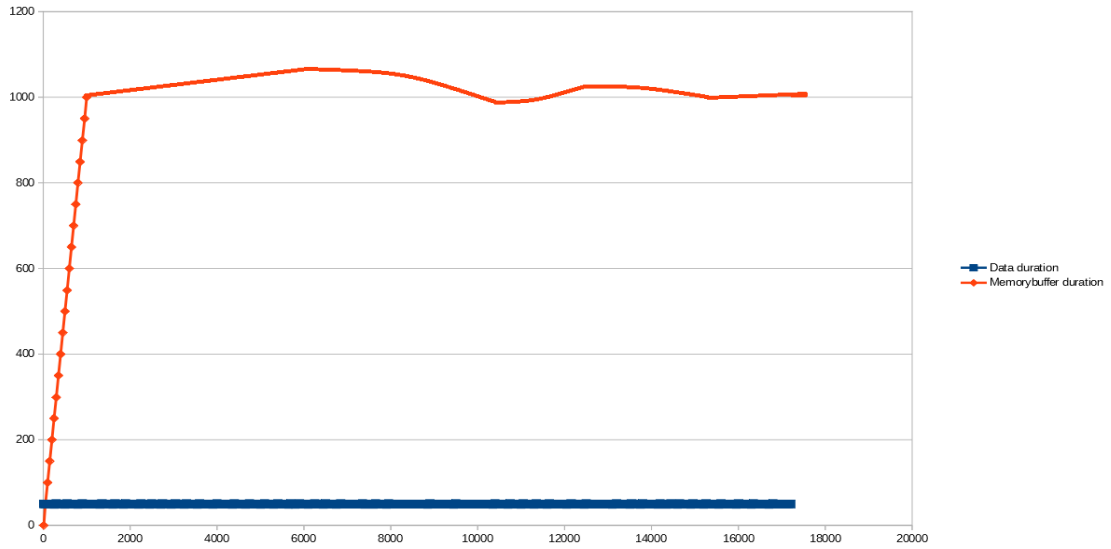
Kuvio 31. Normalisointi muistipuskurin kasvaessa

6.3 Johtopäätökset ja kehitysideat

Haluttu toiminnallisuus saatiin toteutettua, ja aiemmin kuvatuissa tapauksissa muistipuskurin koko käyttäytyi halutulla tavalla. Metadata ja niistä luodut kuvaajat olivat suurena apuna tulosten havainnollistamisessa sekä toiminnallisuuden oikeaksi varmentamisessa. Muutoin tulokset olisivat jääneet hyvin suppeiksi ja niitä olisi ollut vaikeaa havainnollistaa.

Kehityskohteena ratkaistujen ongelmien osalta olisi audion normalisoinnin parantaminen ja etenkin parempien strategioiden kehittäminen. Yhtenä vaihtoehtona voisi olla strategia, joka ei korjaisi niin radikaalisti muistipuskurin kokoa, vaan tutkisi ennemminkin pidemmällä aikavälillä ja tietyin väliajoin muistipuskurin koon kehittymistä. Tähän tietoon perustuen voitaisiin dataa poistaa maltillisesti. Tämän avulla dataa ei katoaisi niin paljon kerralla. Näin ollen vältytään tilanteelta, että dataa on poistettu liian paljon

ja audio on muuttunut liikaa. Metadatasta luodun kuvaajan käyrä (ks. kuvio 32) lähtisi alussa nousemaan lineaarisesti ylöspäin, mutta alkaisi hitaasti taittua kohti haluttu arvoa. Seuraava nousukohta ei olisi enää yhtä jyrkkä, vaan käyrä alkaisi hiljalleen normalisoitua haluttuun arvoon, kunnes se olisi liki normalisoitu tasaiseksi vaakasuoraksi viivaksi.



Kuvio 32. Kuvitteellinen tilanne muistipuskurissa

7 Pohdinta

7.1 Miten työ onnistui?

Opinnäytetyö eteni koko kehitysvaiheen ajan jouhevasti ja suunnitellusti kohti haluttua lopputulosta. Kehitystyön aikana ei ollut isoja ongelmia, jotka olisivat estäneet työn jatkamisen. Tutkimustyön aikana saatiin tietoa nimenomaan siitä, miten audion normalisointiongelma tulisi ratkaista, että se vastaisi haluttua lopputulosta mahdollisimman hyvin. Tutkimustyössä selvisi myös paljon hyödyllistä tietoa siitä, miten audio toimii sekä kuinka sitä tulisi oikeaoppisesti käsitellä ja hallita siirron aikana digitaalisessa tavumuodossa. Testauksen ja toiminnallisuuden kokeilussa saatiin aikaan tilanteita, jossa tallennettua audiota oli muokattu väärin. Tällöin audiota kuunneltaessa oli havaittavissa kohdat, jossa audionäyte oli jakautunut kahtia. Tähän tilanteeseen liittyen saatiin tutkittua sitä, miten audiota tulisi muokata, jotta mainitulta ongelmalta vältyttäisiin.

7.2 Suunnittelu vs. toteutus

Suunnitteluun peilaten kaikki halutut toiminnallisuudet tuli toteutettua, jotka vaatimusmäärittelyssä oli kuvattuna. Tämän perusteella toteutus vastasi liki täysin haluttua toteutettua lopputulosta. Kaikkein parasta mahdollista strategiaa audion normalisointia varten ei toteutettu, mutta sellainen ratkaisu saatiin toteutettua, joka vastasi pääpiirteittäin haluttua toimintakuvaa. Hienosäätö ja paremman strategian luominen olisivat jatkokehitykseen meneviä asioita, kuten jo aiemmin todettiin.

Toteutusvaiheen aikana tuli suunnitelmaan verraten muutamia lisäyksiä, mitkä olivat olennaisia koko työn kannalta ja johtivat työtä parempaan ja havainnollistavampaan suuntaan. Toteutus kulki hyvin käsi kädessä suunnitellun vaatimusmäärittelyn kanssa, eikä sivuraiteille poikettu missään vaiheessa. Toteutus pysyi näin ollen hyvin aihekeskeisenä, ja näin ollen ylimääräisiä komponentteja ei toteutettu.

7.3 Tavoitteet

Toimeksiantajan tavoitteet opinnäytetyöprojektin kannalta täyttyivät suunnitellusti. Opinnäytetyöstä saatiin toteutettua toimiva kokonaisuus, josta voi olla toimeksiantajalle hyötyä tulevaisuudessa. Opinnäytetyön ohella opinnäytetyön tekijä oli myös mukana asiakasprojektissa ja näin ollen pääsi mukaan projektityöhön jo aikaisessa vaiheessa.

Opinnäytetyön tekijälle tavoitteena oli oppia aiemmista opintojaksojen harjoitustöistä suunnittelun ja raportointityön osalta. Näiden osalta päästiin siihen, että suunnittelua oli enemmän kuin aikaisemmissa harjoitustöissä, mutta sitä olisi voinut olla vielä myös hieman lisää. Tähän osakseen vaikutti se, että aihepiiri ei ollut entuudestaan tuttu, joten suunnittelussa ei ollut tarttumapintaa aikaisempiin kokemuksiin aiheesta. Harjoitustöiden raportteihin verrattuna tästä opinnäytetyön raportista tuli alusta saakka johdonmukainen, ja tarkoituksenmukaisesti aihepiiriä pienennettiin vaiheittain kohti toteutuksen yksityiskohtaisia pieniä hienouksia. Täten tavoitteet täyttyivät opinnäytetyön tekijän osalta lähes kokonaan.

7.4 Yhteenveto

Opinnäytetyöprojektissa oli selkeä toimeksiantajan toimesta ennalta määritelty aihe, josta lähdettiin toteuttamaan haluttua ratkaisua. Koko kehitystyön ajan työ eteni eteenpäin aikataulun mukaisesti. Käytössä olleesta ketterän kehityksen menetelmästä Scrum oli suurta hyötyä työn aikana. Sprinttien avulla työ saatiin jaksotettua järkeviin osa-alueisiin, jotka toteutettiin kunkin sprintin aikana. Sprinttien päätteeksi pidettiin sprinttikatselmoinnit, jotka olivat hyödylliset sen kannalta, että projektin etenemisen kokonaiskuva hahmottui paremmin toimeksiantajan muille työntekijöille ja heiltä oli mahdollista saada palautetta sprintin aikana toteutetusta työstä.

Valitut teknologiavalinnat edesauttoivat myös työn onnistumista. GitLabin käyttö versionhallinnassa toimi moitteettomasti, ja sen avulla oli kätevää varastoida lähdekoodit talteen. Valittu LibreOfficen Calc-ohjelma oli myös helppokäyttöinen, ja se oli avainasemassa metadatasta luotujen kuvaajien osalta. Audacityn osalta roolina oli toimia varmentamisessa, kun kuunneltiin audiota tai verrattiin esimerkiksi yhteyden katkeamistilanteessa metadatasta luotua kuvaajaa Audacityn audiovirran kuvaajaan. Näissä kahdessa ohjelmassa tuli selvästi olla nähtävissä sama tapaus. Ohjelmointiympäristö sekä valittu ohjelmointikieli sopivat hyvin yhteen, koska valittu ohjelmointiympäristö Qt Creator sisälsi C++-kielellä ohjelmoituja valmiita ohjelmistoluokkia, joita hyödynnettiin audion käsittelyssä. Projektinhallintaan liittyvä Jira oli hyvä hallinnointityökalu sprinttien suunnittelussa ja niiden hallinnassa, ja siitä sai hyvän kokonaiskäsityksen projektin etenemisestä.

Toteutusvaiheessa edettiin alussa määritellyn ohjelmistorungon toteuttamisen muodossa, minkä jälkeen keskityttiin varsinaisen tutkimustyön pääpainopisteeseen ja sen ratkaisemiseen. Lopputuloksessa käytettiin aiemmin määriteltyjä teknologiavalintoja ja saatiin toteutettua ratkaisu, joka korjaa määritellyn ongelman. Ratkaisun hienosäätö on jatkokehityksen kehityskohde, mutta muuten halutut toiminnallisuudet ja määritellyt ominaisuudet saatiin toteutettua onnistuneesti. Edellä mainittujen seikkojen, tavoitteiden ja tulosten perusteella opinnäytetyön voidaan katsoa olevan onnistunut kokonaisuus.

Lähteet

About Audacity. N.d. Audacityn kotisivut. Viitattu 11.7.2017. http://www.audacityteam.org/about/

About Us. N.d. Combitech-konsernin kotisivut. Viitattu 12.6.2017. http://www.combitech.com/about-us/

C++ Programming Language. N.d. Määritelmä techopedia.com-verkkosivustolla. Viitattu 10.7.2017. https://www.techopedia.com/definition/26184/c-programming-language

Difference between Sound and Audio. N.d. Analyysi Difference Between -verkkosivustolla. Viitattu 15.6.2017. http://www.differencebetween.info/difference-between-sound-and-audio

GitLab Inc. N.d. GitLabin kotisivut. Viitattu 7.7.2017. https://about.gitlab.com/about/

GitLab vs GitHub. N.d. Blogijulkaisu GitLabin ja GitHubin eroista Usersnap-verkkosivustolla. Viitattu 7.7.2017. https://usersnap.com/blog/gitlab-github/

Importance Of Planning In Software Development. N.d. Artikkeli Method Factoryn verkkosivustolla. Viitattu 6.7.2017. http://www.methodfactory.com/about/articles/general/importance-of-planning-in-software-development

Integrated Development Environment (IDE). N.d. Määritelmä techopedia.com-verkkosivustolla. Viitattu 6.7.2017. https://www.techopedia.com/definition/26860/integrated-development-environment-ide

JIRA Software. N.d. Jira-ohjelmiston kotisivut. Viitattu 7.7.2017. https://www.atlassian.com/software/jira

Lauri, C. & Malmgren, J. 2015. Synchronization of streamed audio between multiple playback devices over an unmanaged IP network, 30-31. Pro Gradu-tutkielma. Lundin yliopisto. Viitattu 26.7.2017. http://www.eit.lth.se/sprapport.php?uid=894

Learn Version Control with Git. N.d. Opastusartikkeli Gitin käyttöön git-tower.com-verkkosivustolla. Viitattu 6.7.2017. https://www.git-tower.com/learn/git/ebook/en/command-line/remote-repositories/introduction

Luontola, E. & Paksula, M. 2009. Versionhallinta. Viitattu 6.7.2017. https://www.cs.helsinki.fi/u/paksula/versionhallinta/s10/git.pdf

Programming Language. N.d. Määritelmä Techopedian verkkosivustolla. Viitattu 6.7.2017. https://www.techopedia.com/definition/24815/programming-language

QAudioFormat Class. N.d. Detailed Description. Ohjelmistoluokan määritelmä Qt:n verkkosivustolla. Viitattu 21.7.2017. http://doc.qt.io/qt-5/qaudioformat.html#details

Rouse, M. 2014. TCP (Transmission Control Protocol). Määritelmä TechTargetin verkkosivustolla. Viitattu 21.7.2017. http://searchnetworking.techtarget.com/definition/TCP

Rouse, M. 2015. Implementation. Määritelmä TechTargetin verkkosivustolla. Viitattu 18.7.2017. http://searchcrm.techtarget.com/definition/implementation

Rouse, M. 2016a. GitHub. Määritelmä TechTargetin verkkosivustolla. Viitattu 7.7.2017. http://searchitoperations.techtarget.com/definition/GitHub

Rouse, M. 2016b. Java. Määritelmä TechTargetin verkkosivustolla. Viitattu 10.7.2017. http://searchmicroservices.techtarget.com/definition/Java

Taina, J. 2010. Ohjelmistojen vaatimusmäärittely. Viitattu 4.7.2017. https://www.cs.helsinki.fi/u/taina/ovm/k-2010/pdf/kalvot1-42_6.pdf

Team chat that's actually built for business. 2017. HipChat-sovelluksen kotisivut. Viitattu 7.7.2017. https://www.atlassian.com/software/hipchat

The IDE Qt Creator. N.d. Qt Creatorin kotisivut. Viitattu 10.7.2017. https://www.qt.io/ide/

Tietoja Combitechistä. 2014. Combitech Oy:n kotisivut. Viitattu 12.6.2017. http://www.combitech.fi/Tietoja-Combitechista-/

Valdarrama, S. 2014. Software design is only important if you want to do a good job. Blogijulkaisu Shiftedup -verkkosivustolla. Muokattu 8.10.2014. Viitattu 5.7.2017. http://www.shiftedup.com/2014/10/08/software-design-is-only-important-if-you-want-to-do-a-good-job

Visual Studio IDE. N.d. Visual Studion kotisivut. Viitattu 10.7.2017. https://www.visualstudio.com/vs/

We've been acquired by Atlassian!. 2012. Blogijulkaisu HipChatin kehittäjien verkkosivustolla. Viitattu 7.7.2017. https://blog.hipchat.com/2012/03/07/weve-been-acquired-by-atlassian/

What Is Agile Software Development?. N.d. Määritelmä Versiononen verkkosivustolla. Viitattu 18.7.2017. https://www.versionone.com/agile-101/

What is a DAC?. N.d. Artikkeli ubergizmo.com-verkkosivustolla. Viitattu 16.6.2017. http://www.ubergizmo.com/what-is/dac/

What is epoch time?. N.d. Määritelmä epochista Epochconverterin verkkosivustolla. Viitattu 21.7.2017. https://www.epochconverter.com/

What is Git. N.d. Määritelmä Atlassianin verkkosivustolla. Viitattu 6.7.2017. https://www.atlassian.com/git/tutorials/what-is-git

What is GStreamer?. N.d. Määritelmä GStreamerin verkkosivustolla. Viitattu 25.7.2017. https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html

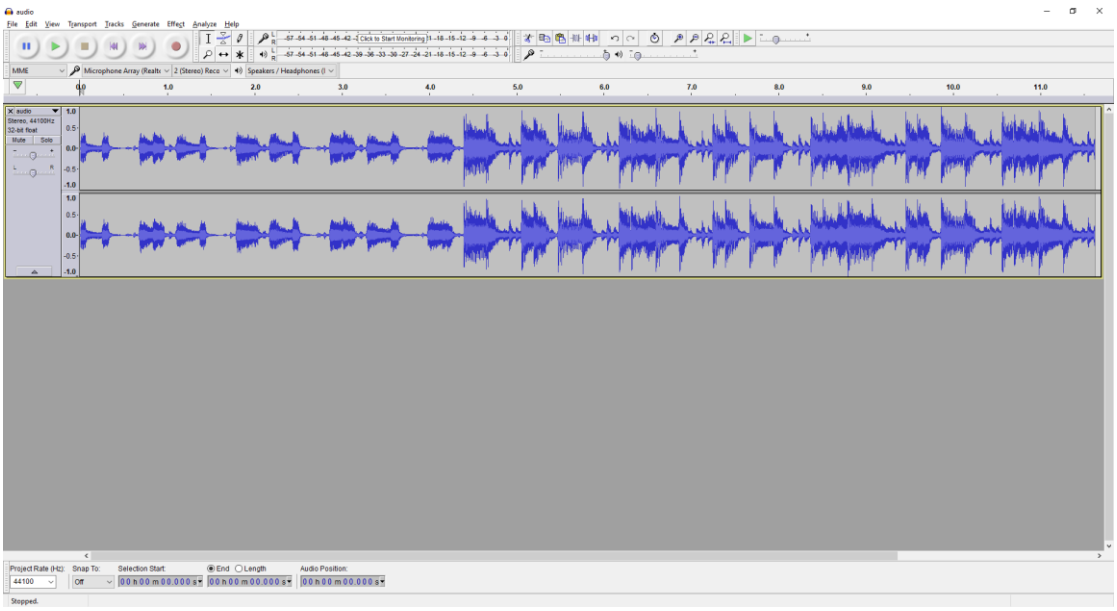
What is LibreOffice?. N.d. LibreOfficen kotisivut. Viitattu 11.7.2017. https://www.libreoffice.org/discover/libreoffice/

What is version control. N.d. Määritelmä Atlassianin verkkosivustolla. Viitattu 6.7.2017. https://www.atlassian.com/git/tutorials/what-is-version-control

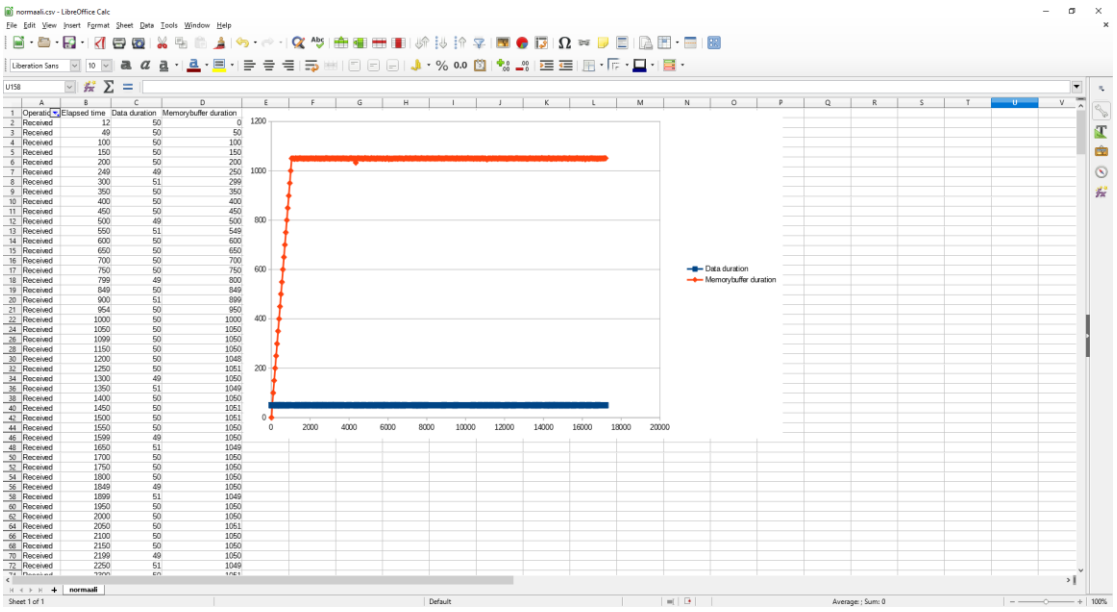
Liitteet

Liite 1. Kuvakaappauksia käytetyistä ohjelmistoista

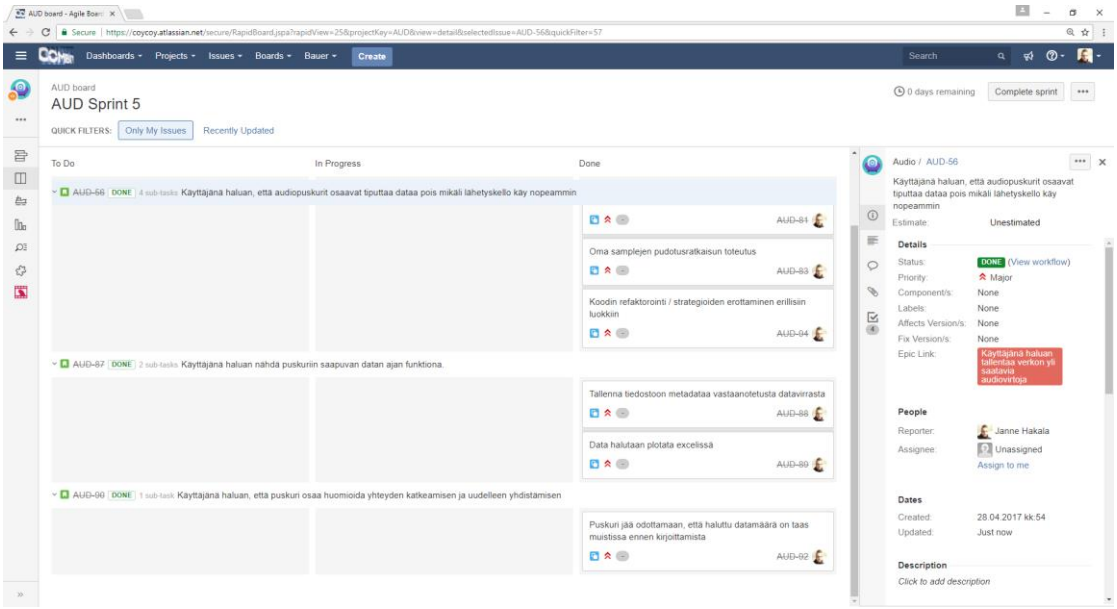
Audacity



Calc



Jira



Qt Creator