OULUN SEUDUN
AMMATTIKORKEAKOULU

Janne Juntunen

ASIAKASOSAN STREAMAUKSEN TOTEUTTAMINEN QT-MEDIASOITTIMEEN

ASIAKASOSAN STREAMAUKSEN TOTEUTTAMINEN QT-MEDIASOITTIMEEN

Opinnäytetyö
Janne Juntunen
14.12.2011
Tekniikan yksikkö
Oulun seudun ammattikorkeakoulu

OULUN SEUDUN AMMATTIKORKEAKOULU TIIVISTELMÄ

Koulutusohjelma	Opinnäytetyö	Sivuja	+	Liitteitä
Tietotekniikan koulutusohjelma	Insinöörityö	38	+	

Suuntautumisvaihtoehto	Aika
Ohjelmistosuunnittelu	14.12.2011

Työn tilaaja	Työn tekijä
Nice Business Solutions Finland Oy	Janne Juntunen

Työn nimi

Asiakasosan streamauksen toteuttaminen Qt-mediasoittimeen

Avainsanat

Qt, Phonon, GStreamer, Linux, Maemo, MeeGo, streamaus, multimedia, RTP

Opinnäytetyössä tehtiin streamaus- eli virtaustoistotoiminnallisuus mediasoittimeen. Työssä jatkokehitettiin Qt:n multimediakirjasto Phononilla tehtyä mediasoitinta Maemo-mobiili-Linux-alustalle. Työn tilaajana toimi Nice Business Solutions Finland Oy, jonka tiloissa työosa suoritettiin. Opinnäytetyötä oli tekemässä kaksi opiskelijaa. Tässä raportissa keskitytään asiakasosan toteutukseen. Työn tarkoituksena oli tutustua streamaukseen. Työn aikana tavoitteisiin lisättiin tutustuminen Qt:n tukeen kolmannen osapuolen teknologioille.Työ määriteltiin sisältämään pelkän alustavan toiminnallisuuden.

Suunnitteluvaiheessa työn aiheeseen tutustuttiin ottamalla selvää saatavilla olevista mahdollisuuksista. Loppujen lopuksi kohdealustan takia käytetyksi teknologiaksi valikoitui GStreamer. Työssä päätettiin käyttää RTP (Real-time Transport Protocol) -protokollaa, ja itse datan siirtoon UDP (Universal Datagram Protocol) -protokollaa.

Toteutusvaiheessa työhön toteutettiin yksinkertainen käyttöliittymäkomponentti Qt:lla. Itse streamin vastaanottaminen ja toistaminen hoidetaan GStreamer-komponentilla, joka on sisäistetty ohjelman lähdekoodiin. Toteutusvaiheessa tehtiin myös tarvittavat muutokset ohjelmaan, jotta kolmannen osapuolen kirjastoja voidaan käyttää.

Opinnäytetyö onnistui odotusten mukaisesti. Ohjelmalla voidaan sekä lähettää että vastaanottaa RTP-streamia. Työn aikana päästiin tutustumaan sekä streamaukseen yleisesti että Qt:n tukeen kolmannen osapuolen kirjastoille. Lopputuloksena on toiminnallisesti alkeellinen, mutta kehityskelpoinen mediasoitin.

SISÄLTÖ

1 JOHDANTO

Qt-käyttöliittymäkirjasto on ollut viime vuosina paljon puhuttu ja kohuttu aihe ohjelmistosuunnittelun alalla. Norjalaisen Trolltechin 90-luvulla lanseeraama kirjasto lähti alulle vallankumouksellisesta ajatuksesta käyttöliittymäkirjastosta, jolla voisi luoda ohjelmistoja, jotka toimisivat laitteella kuin laitteella. Vuonna 2008 Nokia osti Trolltechin ja Qt:stä tuli todellinen vaihtoehto Symbian-käyttöliittymälle. Tämän myötä Qt:n suosio alalla lähti rankkaan nousuun (1.)

Opinnäytetyössä lähdimme, minä ja luokkakaverini Tuomas Kehusmaa, jatkokehittämään aiemmin koulun kolmessa harjoitteluprojektissa luomaamme mediasoitinta Maemo-käyttöjärjestelmälle. Työn tilaajana, kuten harjoitteluprojekteissakin, toimi Nice Business Solutions Oy, Fujitsun ja Nokian omistama ohjelmistoalan alihankkijayritys, jonka tiloissa Oulun Teknologiatiellä työosuus suoritettiin. Ohjelmiston alkuperäisenä tarkoituksena oli tutustua Qt:hen käyttöliittymäkirjastona sekä sen Phonon-mediakirjaston tuomiin mahdollisuuksiin videon ja audion toistossa, tavoitteet, joihin olimme päässeet harjoitteluprojektien aikana.

Itse opinnäytetöiden aiheeksi valikoitui audion streamaus verkon yli, luonnollinen kehitysaste, kun samalla laitteella olevan audion ja videon toisto oli jo ohjelman toiminnallisuuksissa. Aiheen valintaan vaikutti myös se, että tämä mahdollisti selvän kahtiajaon asiakkaaseen ja serveriin, olihan työstä tarkoitus kirjoittaa kaksi raporttia. Raportissa selvitän sekä asiakaspuolen toimintaa että koko ohjelman suunnitteluprosessia. Tästä johtuen raportissa esiintyy siellä täällä tekijänä me, jolloin kyse on ongelmasta, jota ratkoimme keskenämme.

Opinnäytetyö lähti liikkeelle puhtaalta pöydältä. Kummallakaan osapuolella ei ollut streamauksen toteuttamisesta aiempaa kokemusta. Työn alussa selvisi nopeasti, että ratkaisu täytyisi etsiä Phononin ulkopuolelta kolmannen osapuolen ratkaisuna, koska Phononissa ei ollut toiminnallisuutta tähän

suuntaan. Näin ollen työhön tuli isoksi osaksi myös sen selvittäminen, miten Qt pärjää kolmannen osapuolen ohjelmistokirjastoja käytettäessä.

2 KÄYTETYT TEKNOLOGIAT

Projektissa toteutimme Qt:llä toteutettuun mediasoittimeen streamaustoiminnallisuuden pohjapiirteissään. Kohdealustana oli Nokian N900, kosketusnäytöllinen internet-tabletti, jonka käyttöjärjestelmänä toimii Maemo 5.

Mediasoittimen perustoiminnot on tähän mennessä tehty käyttäen Qt:n omaa mediakirjastoa Phononia. Phononissa ei kuitenkaan työn tekoaikaan (versio 4.3.1) ollut streamaustoimintoa, joten päädyimme käyttämään projektissa GStreamer-multimediakirjastoa. Tiedonsiirto tapahtui käyttäen RTP-protokollaa. Tässä luvussa käsitellään edellämainittuja teknologioita ja sitä miten kohdealusta täytyy ottaa huomioon projektia toteuttaessa.

2.1 N900

N900 on viimeinen Nokian kehittämistä internet-tableteista, jotka käyttävät Maemo-käyttöjärjestelmää. Ensimmäinen Maemoa käyttänyt laite julkaistiin vuoden 2005 marraskuussa: Nokia 770 Internet Tablet, jolla pystyi selaamaan internettiä WLAN (Wireless Local Area Network) -yhteyttä käyttäen. Muita laitteita ovat N800, joka mahdollisti Skype/VoIP-puhelut WLAN:n yli, ja N810, ensimmäinen Nokian internet-tableteista, jossa oli täydellinen QWERTY-näppäimistö kosketusnäytön ohella. Helmikuussa 2010 Mobile World Congress -tapahtumassa Intel ja Nokia ilmoittivat aloittavansa yhteistyön Maemon jatkokehityksessä ja parempien päätelaitteiden tuottamisessa. Uusi MeeGo olisi suora jatko sekä Maemolle että Intelin Moblin-käyttöjärjestelmälle. (2; 3.)

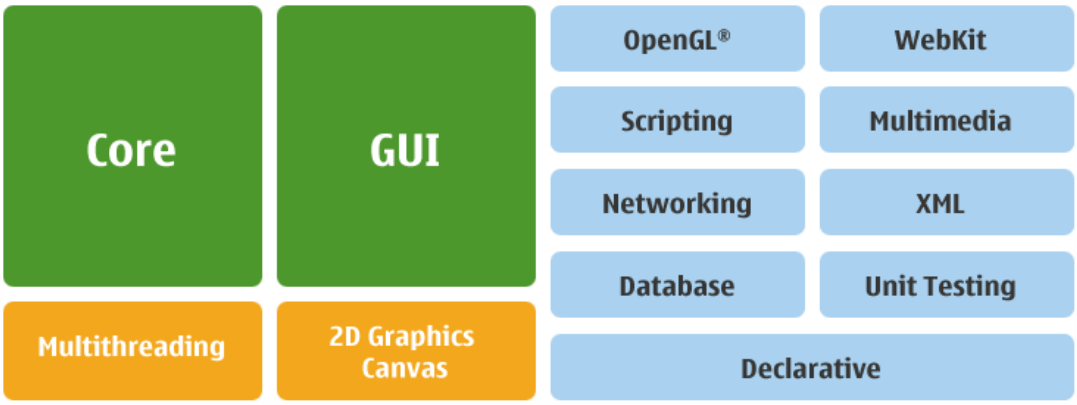
2.2 Maemo

Maemo on käyttöjärjestelmä, joka on jo kehityksen alkuvaiheessa rakennettu vapaan lähdekoodin periaatteita kunnioittaen ja koko kehitystyön ajan

muiden isojen vapaan lähdekoodin projektien (Linux kernel, Debian, GNOME) kanssa tiiviissä yhteistyössä. Käyttöjärjestelmän ydin on ARM/OMAP-pohjainen Linux-ydin, jonka päälle itse GNU (GNOME Not Unix) C -kirjastoon pohjaava käyttöjärjestelmä rakentuu. Maemon järjestelmäarkkitehtuuri ja yleinen tiedostohierarkia on kehitetty Debian-jakelun vastaavien pohjalta. Maemon Hildon-käyttöliittymäkirjasto on puolestaan rakennettu GNOME-projektin GTK+-kirjaston pohjalle. (4.)

2.3 Qt

Qt on luokkakirjasto, joka on kehitetty helpottamaan ohjelmistojen luontia usealle eri alustalle. Qt:n suunnittelu ja toteutus mahdollistavat saman lähdekoodin käytön useissa eri alustoissa. Tämä mahdollistetaan ohjelmointirajapinnalla, jossa järjestelmäkohtaiset toteutukset eivät näy ohjelmoijalle. Qt on täysin oliopohjainen. Pääasiassa ohjelmointi tapahtuu C++:lla. Tosin version 4.7.2 myötä maaliskuussa 2011 Nokia toi mukaan QML-scriptikielen. Kuvassa 1 on Qt:n modulaarinen luokkarakenne. (1; 5, s. 3.)



KUVA 1. Qt:n modulaarinen luokkarakenne (6.)

2.3.1 Projektitiedosto

Qt-ohjelmassa tärkein yksittäinen tiedosto on projektitiedosto. Se ohjaa ohjelman kääntämisen ensimmäistä vaihetta, jossa qmake, moc (metaobject

compiler) ja uic (user interface compiler) esikääntävät Qt-ohjelman tiedostoista alustakohtaiset tiedostot, jotka lopulta käännetään ja linkitetään ohjelmaksi. (5, s. 38.)

Projektitiedostossa on listattuna ohjelmaan kuuluvat lähdekoodi-, otsikko-, käyttöliittymä- ja muut resurssitiedostot. Projektitiedostossa lukevat myös muut kääntämisen kannalta olennaiset tiedot tarvittavista Qt:n kirjastoista aina kolmannen osapuolen kirjastoihin. Myös mahdolliset alustakohtaiset kääntöehdot on kirjoitettu projektitiedostoon. Seuraavassa on esimerkki. (7.)

```
# -------------------------------------------------
# Project created by QtCreator 2009-04-28T14:21:24
# -------------------------------------------------
QT += network \
    script \
    sql \
    svg \
    webkit \
    xml \
    xmlpatterns \
    phonon \
    qt3support \
    testlib \
    dbus
TARGET = media
TEMPLATE = app
LIBS += -lgstreamer-0.10
SOURCES += audio.cpp \
    video.cpp \
    mediaplayer.cpp \
    main.cpp \
    streamingServer.cpp \
    streamingClient.cpp
HEADERS += audio.h \
    video.h \
    mediaplayer.h \
    streamingServer.h \
    streamingClient.h
```

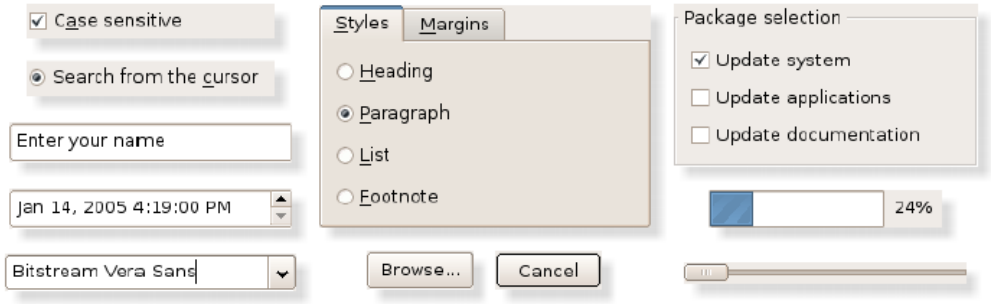
Projektin tiedostot löytyvät otsikoiden HEADERS ja SOURCES alta sekä tarvittavat Qt-kirjastot QT-nimekkeen alta. Esimerkistä puuttuu projektin käyttöliittymätiedostot, jotka voidaan luoda käyttämällä QtCreatorin mukana tulevaa Designeria. Projektitiedostossa käyttöliittymätiedostot listataan FORMS-nimekkeen alle. (7.)

2.3.2 qmake

qmake on työkalu, jota Qt käyttää helpottaamaan usealle alustalle kääntämistä. Sen pääasiallinen tehtävä on lukea projektitiedostoa ja luoda alustakohtainen Makefile, joka ohjaa ohjelman lopullista kääntämistä ja linkittämistä. (7.)

2.3.3 Widgetit

Qt:llä tehdyt käyttöliittymät pohjaavat widget-elementteihin. Jokainen graafinen käyttöliittymäelementti on oma widgetinsä, joiden ulkoasua ohjaa ikkunassa oleva muotoiluwidget. Normaaleja widgetejä ovat esimerkiksi painonapit ja liukuvalitsimet. Kuvassa 2 on esimerkkejä widgeteistä. (5, s. 5.)



KUVA 2. Esimerkkejä widget-elementeistä (5, s. 5.)

Muotoiluwidget (englanniksi layout) on elementti, jolla mahdollistetaan widgetin ulkoasun joustava hallitseminen. Koodiesimerkissä ohjelman pääikkunawidgetille annetaan ristikkomuotoiluwidget. (5, s. 6)

```
/* Take the copy of mainPage and add this pointer value to the member variable */
this->mainPage = this;
/* The top level widget for this window */
gridLayoutWidget = new QWidget(this);
this->resize(718,420);
this->setCentralWidget(gridLayoutWidget);
gridLayoutWidget->setObjectName(QString::fromUtf8("gridLayoutWidget"));
gridLayoutWidget->setGeometry(QRect(696, 396, 696, 396));
gridLayout = new QGridLayout(gridLayoutWidget);
gridLayout->setSpacing(6);
gridLayout->setMargin(11);
gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
gridLayout->setContentsMargins(0, 0, 0, 0);
```

Muotoiluwidget tuo mukanaan seuraavat automatisoidut keinot ulkoasun hallitsemiseksi:

- lapsiwidgetien sijoittaminen
- järkevä oletuskoko
- järkevä minimikoko
- kokomuutosten käsittely
- sisällön automaattinen päivitys, esimerkiksi
    - fonttikoon tai muun widgetien sisällön muuttuessa
    - lapsiwidgetejä näytettäessä ja piilottaessa
    - lapsiwidgetejä tuhottaessa.

Seuraavassa esimerkkikoodissa lisätään QGridLayout-muotoiluwidgetiin kaksi painonappia:

```
streamingLayout->addWidget(pushButton_streamingServer,1,0,1,1);
streamingLayout->addWidget(pushButton_streamingClient,2,0,1,1);
```

Ensimmäisenä addWidget-funktion kutsussa annetaan widget, joka layoutiin lisätään. Numerot funktion kutsussa puolestaan merkkaavat seuraavaa järjestyksessä ensimmäisestä viimeiseen:

1. Sijainti y-akselilla (Solu 0.0 vasemmassa yläkulmassa)
2. Sijainti x-akselilla
3. Widgetin pituus y-akselin suunnassa
4. Widgetin pituus x-akselin suunnassa. (5, s. 6; 8)

2.3.4 Signaalit ja slotit

Qt:ssä widgetit viestivät keskenään käyttäen signaaleihin ja sloteihin perustuvaa tapahtumakäsittelijää. (9.)

Qt-ohjelmassa signaalien ja slotien käytön mahdollistavat metaobjektit. Metaobjekti on metaobject compilerin eli moc:n käännön yhteydessä luoma väliaikainen C++-tiedosto, joka sisältää signaalien ja slotien välisien yhteyksien käytännön toteutuksen. Metaobjektit sisältävät myös ajonaikaista informaatiota luokasta. Jotta moc tietäisi, mistä tiedostoista metaobjektit pitää luoda, täytyy luokan määrittelyn yhteydessä otsikkotiedostossa olla Q_OBJECT-makro. Tämä makro täytyy siis löytyä jokaisen signaaleja ja sloteja käyttävän luokan määrittelystä. (9; 10.)

Jokaisella widgetillä on omat signaalinsa ja slotinsa, joiden välisiä yhteyksiä pääohjelma hallitsee. Esimerkiksi aina kun painonappia painetaan, lähetetään napin clicked()-signaali. Tämä signaali voidaan yhdistää slotteihin, joissa tehdään halutut toimenpiteet. Yhdistäminen tapahtuu yleensä kutsumalla QObject-luokalta perittyä connect-funktiota ja antamalla parametreinä seuraavan informaation:

1. Lähettävä objekti (esimerkeissä käyttöliittymästä löytyvä painonappi)

2. Lähetettävä signaali (painonappien clicked(), lähetetään nappia painettaessa)

3. Vastaanottava objekti (luokka jossa yhteys luodaan)

4. Vastaanottava slot tai signaali (haluttu toiminnallisuus).

```
QObject::connect(ui.pushButton_playlist, SIGNAL(clicked()), this, SLOT(openDialog()));
QObject::connect(ui.pushButton_Remove, SIGNAL(clicked()), this, SLOT(removeSong()));
```

Koodissa slot esiintyy ja käyttäytyy normaalin funktion tavoin. Signaalien ja slotien välisiä yhteyksiä voidaan muokata missä tahansa vaiheessa ohjelman koodia. Koska systeemi käsittelee signaalit pääohjelmassa, toisiinsa yhteydessä olevien widgettien ei tarvitse olla tietoisia toisistaan ja muistivuotojen mahdollisuus pienenee sekä monikäyttöisten komponenttien suunnitteleminen helpottuu. (9; 11.)

2.3.5 Phonon

Multimedian toistoon Qt käyttää Phonon-kirjastoa. Tämä alun perin vapaan lähdekoodin KDE (K Desktop Environment) -projektissa alkunsa saanut multimediakirjasto kehitettiin alkujaan mahdollistamaan alustasta riippumaton ohjelmointirajapinta mediatoistolle. Phonon ei itse toista mediaa vaan hyödyntää alustakohtaista mediakirjastoa ja sen ominaisuuksia. Linuxilla käytössä on pääasiallisesti GStreamer. (12.)

2.4 RTP

Real-time Transport Protocol on protokolla, joka mahdollistaa datan reaaliaikaisen siirron sovellustasolla internetin yli yhteen tai useampaan päätepisteeseen. RTP ei takaa siirron laatua eikä ole vastuussa mahdollisten resurssien varaamisesta. Datasiirtoa valvotaan RTCP (Real-time Transport Control Protocol) -protokollan avulla, joka paitsi valvoo siirtyvää dataa myös mahdollistaa päätelaitteiden välisen minimaalisen

kommunikaation. RTP ja RTCP on määritelty RFC3550-standardissa, joka on julkaistu vuoden 2003 heinäkuussa. (13.)

2.5 GStreamer

GStreamer on vapaan lähdekoodin periaatteisiin perustuva ohjelmistokirjasto, jonka avulla voidaan luoda mediaa toistavia ohjelmistoja. GStreamerin kehitys alkoi vuonna 1999 ja ensimmäinen julkinen versio oli 11.1.2001 julkaistu versio 0.1.0. Huolimatta projektia hallinneen RidgeRun-yhtiön ajautumisesta taloudellisiin ongelmiin projektin kehitys oli nopeaa ja jo vuoden 2004 maaliskuussa julkaistiin 0.8.0. Nykyinen versio on 0.10.32, joka on uusin vakaan 0.10.x-sarjan versio. (14.)

GStreamer käyttää niin sanottua pipeline-menetelmää luodakseen helposti muokattavissa olevia mediatoistoratkaisuja. Pipeline-menetelmässä jokainen pienikin osa mediatoistotapahtumaa tapahtuu oman plug-ininsa kautta, jotka ovat yhteydessä toisiinsa pipelinen sisällä. Jokaiselle mediakoodekille on omat encode- ja decode-plug-ininsa, kuin myös jokaiselle streamia muokkaavalle toiminnolle. Versiosta 0.10 lähtien plug-init on jaettu neljään pakettiin taulukosta 1 löytyvin perustein. (15; 16.)

TAULUKKO 1. GStreamerin plug-in-paketit ja niiden lyhyt kuvaus (16.)

Plug-in-paketti	Kuvaus
Base	Paketti sisältää plug-init, jotka ovat hyvin viimeisteltyjä ja ylläpidettyjä. Tämä paketti sisältää myös plug-inien suunnittelua ja toteuttamista helpottavia kirjastoja.
Good	Paketti, joka sisältää laadukkaat GNU LGPL (Limited General Public

	Licence) -lisenssin alla julkaistut plug-init.
Bad	Paketissa ovat plug-init, joiden toiminta on epävarmaa, dokumentointi puutteellista, testaus ei ole ajantasalla ja/tai ylläpito ei ole ajantasalla.
Ugly	Paketissa ovat plug-init, jotka ovat täysin toimivia, mutta joiden lisenssit voivat aiheuttaa ongelmia ohjelmaa julkaistessa.

Seuraavassa on kuvailtu lyhyesti ohjelman asiakasosassa käytetyt plug-init:

- **gstrtpbin** on plug-in, joka sisältää kaikki RTP- ja RTCP-protokollien käyttämiseen tarvittavat toiminnallisuudet yhdessä pluginissa. Plug-in on luotu helpottamaan RTP-yhteyden luomista ja ylläpitoa. Tämä plug-in löytyy Good-paketista.

- **udpsrc** on plug-in, joka ottaa vastaan dataa UDP (User Datagram Protocol) -protokollan yli. Plug-in ei muokkaa tulevaa dataa, mutta sen voi yhdistää gstrtpbinin rtpsrc- ja rtcpsrc-elementteihin. Ohjelmassa plug-inia käytetään sekä RTP-, että RTCP-datan vastaanottamiseen serveriltä. Tämä plug-in löytyy Good-paketista.

- **udpsink** on plug-in, joka lähettää dataa UDP-protokollan yli. Ohjelmassa plug-inia käytetään RTCP-datan lähettämiseen serverille. Tämä plug-in löytyy Good-paketista.

- **rtpmpadepay** on plug-in, joka ottaa vastaan RTP:n yli siirtyvän datan ja muuntaa sen MPEG-audioksi. Tätä plug-inia täytyy käyttää kun

haluaa vastaanottaa mp3-streamia. Tämä plug-in löytyy Good-paketista.

- **mad** on plug-in, jolla MPEG-audio muutetaan raw-audioksi. Tämä plug-in löytyy Ugly-paketista.

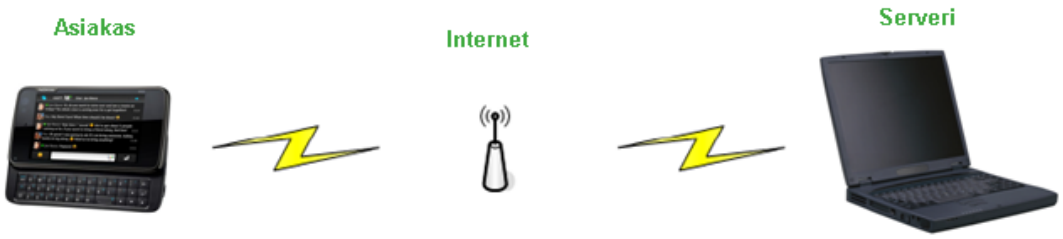
- **audioconvert** on plug-in, jolla audiota voidaan muokata eri muotoon. Tämä plug-in löytyy Base-paketista.

- **audioresample** on plug-in, jolla audion näytteenottotaajuutta voidaan muuttaa. Tämä plug-in löytyy Base-paketista.

- **autoaudiosink** on plug-in, joka etsii automaattisesti järjestelmästä sopivan elementin audion toistamiselle. Tämä plug-in löytyy Good-paketista. (17.)

3 STREAMAUSOHJELMAN MÄÄRITTELY

Opinnäytetyössä toteutetaan streamaustoiminnallisuus mediasoittimeen. Ohjelma, jonka pohjalle työ tehdään, ja työn käyttöliittymä on toteutettu käyttäen Qt-ohjelmointikirjastoa. Itse streamaus toteutetaan käyttäen GStreamer-mediakirjastoa. Työn lopputuloksena on ohjelma, jolla voi ottaa yhteyden toiseen laitteeseen ja joko lähettää tai vastaanottaa tosiaikaista mediastreamia. Kuvassa 3 on ohjelman toiminta periaatetasolla.



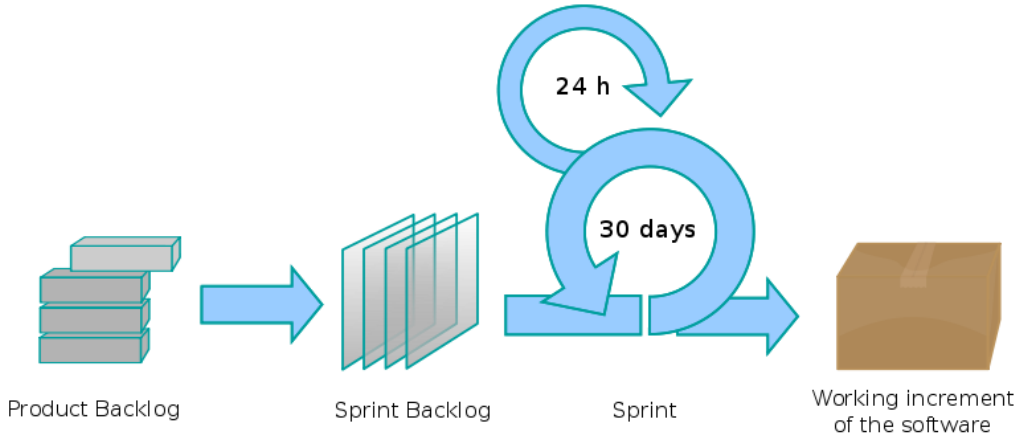
KUVA 3. Streamausohjelman toiminta periaatetasolla

Kun serveri lähettää streamia, ohjelma lukee serverin kovalevyltä mediatiedostoa ja lähettää luetun datan UDP-yhteyden yli RTP-protokollaa käyttäen. Asiakas puolestaan ottaa vastaan RTP-paketteja ja toistaa ne käyttäen vastaavia laiterajapintoja. Tämä raportti keskittyy ohjelman asiakasosaan. Työ rajattiin sisältämään vain alustavan toiminnallisuuden, josta ohjelmaa voidaan kehittää edelleen.

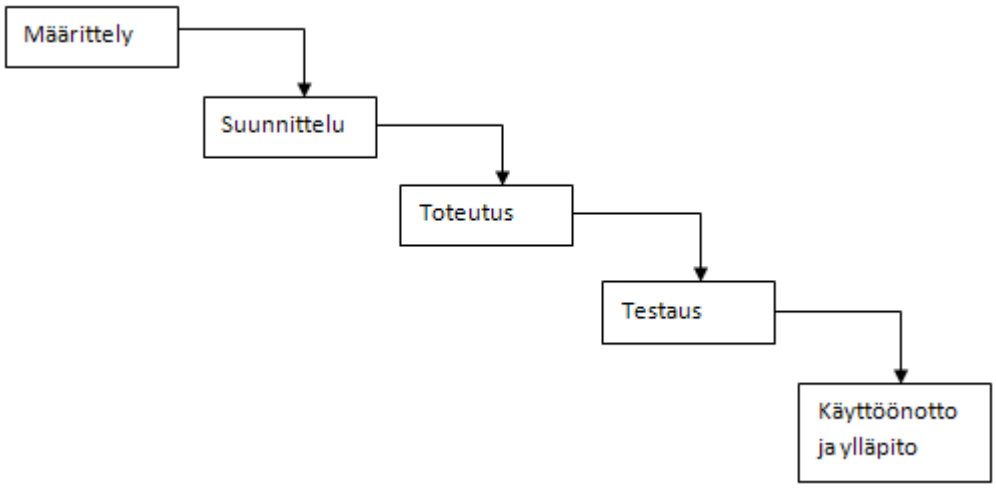
4 SUUNNITTELU JA TOTEUTUS

Ohjelma on rakennettu yksinkertaisen luokkarakenteen mukaisesti, jokainen käyttöliittymä ja engine omana kokonaisuutenaan. Tämä rakenne helpottaa uusien toiminnallisuuksien lisäämistä jälkeenpäin, koska jokainen osa on erillinen kokonaisuutensa, joka huolehtii vain ja ainoastaan itsestään. Tulevaisuudessa kuitenkin käyttötapausten ja ohjelman eri osien välisen interaktion lisääntyessä täytyy miettiä, voisiko eräänlainen MVC (Model View Controller) -malli olla käytännöllisin.

Opinnäytetyö toteutettiin yhtenä Scrum-tyylisenä Agile-sprinttinä, jonka aikana käytiin läpi yksi vesiputousmallin tapainen kehityskaari. Sprinttimme kesti 30 työpäivää. Kuvissa 4 ja 5 on kuvattu näiden mallien periaatteet.



KUVA 4. Scrum-kehitysmallin toimintaperiaate (18.)



KUVA 5. Vesiputousmallin periaate (19.)

4.1 Suunnittelu

Määrittelyn suoritimme aloituspalaverin yhteydessä. Itse projektin aloitimme selvittämällä, mitä mahdollisuuksia streamauksen toteuttamiseen on olemassa, ottaen huomioon projektin erityistarpeet. Päädyimme tarkastelemaan lähemmin kahta vaihtoehtoa.

Suunnitteluvaiheessa tutustuimme VideoLAN Organizationin avoimen lähdekoodin projekteihin, nimellisesti VLC playeriin ja libVLC-mediakirjastoon, joka on suunniteltu tuomaan VLC playerin toiminnallisuudet ohjelmistosuunnittelijoiden käytettäväksi. libVLC oli erityisen kiinnostuksen kohteena myös siitä syystä, että VLC playerin Maemo-versio oli julkistettu. Päädyimme kuitenkin jättämään libVLC:n projektimme ulkopuolelle, koska oli mielestämme sekava ratkaisu implementoida kokonainen mediakirjastokokonaisuus yhden ominaisuuden vuoksi.

Toinen lähempään tarkasteluun päätynyt ratkaisu streamauksen implementointiin oli GStreamer. Tähän päädyimme siksi, että tiesimme jo Phononin käyttävän kyseistä mediakirjastoa ja Maemon repositorystä, eli Linux-käyttöjärjestelmissä käytettävästä ohjelmistovarastosta, löytyi toimiva GStreamer-käännös. GStreameriin päädyttyämme jouduimme kuitenkin

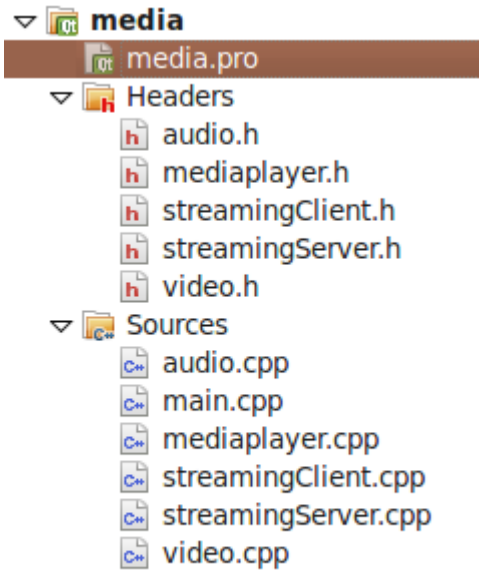
miettimään, aiheuttaisiko GStreamerin C-kielellä oleva toteutus ongelmia C++-syntaksia olevan Qt-ohjelmamme kanssa.

Suurin ongelma tuli GStreamerin tarvitsemasta Gnome main loopista, jota GStreamer-ohjelmat käyttävät ohjelman sisäiseen viestintään ja väliaikaisten resurssien varaamiseen ja vapauttamiseen. Onneksi huomasimme ohjelmaa testatessamme, että ohjelmamme main loop pystyi hoitamaan nämä tehtävät.

Yksi vaihtoehtoinen ratkaisu, jota selvitimme, oli käyttää GStreamerin C++-rajapintaa, omassa vapaan lähdekoodin projektissaan kehitettävää gstreamermm:ää. Ongelma gstreamermm:ssä kuitenkin oli sen kyseenalainen Maemo-tuki ja hiljainen päivitystahti. Lopulta päädyimmekin jättämään gstreamermm:n pois projektista.

4.2 Toteutus

Työn toteutus aloitettiin luomalla streamaukselle käyttöliittymä-, luokka- ja otsikkotiedostot. Kuvassa 6 on projektin rakenne QtCreatorissa.



KUVA 6. Projektin rakenne QtCreatorissa

Tässä raportissa käsitellään lähemmin ainoastaan tähän projektiin liittyvät tiedostot ja tiedostomuutokset.

4.2.1 media.pro

Projektin päätiedosto on media.pro, joka määrittää ohjelman kääntöprosessin: kertoo, mitä tiedostoja käännetään ja mitä kirjastoja käännöksessä huomioidaan, sekä ohjaa mm. moc:n toimintaa. Seuraavassa koodiesimerkissä ovat projektitiedostoon tehdyt muutokset:

```
TARGET = media
TEMPLATE = app
LIBS += -lgstreamer-0.10
SOURCES += audio.cpp \
    video.cpp \
    mediaplayer.cpp \
    main.cpp \
    streamingServer.cpp \
    streamingClient.cpp
HEADERS += audio.h \
    video.h \
    mediaplayer.h \
    streamingServer.h \
    streamingClient.h
CONFIG += no_keywords
INCLUDEPATH +=  /usr/include/gstreamer-0.10/ \
                /usr/include/glib-2.0/ \
                /usr/lib/glib-2.0/include/ \
                /usr/include/libxml2/ \
                /usr/local/Trolltech/Qt-4.7.0/include/
```

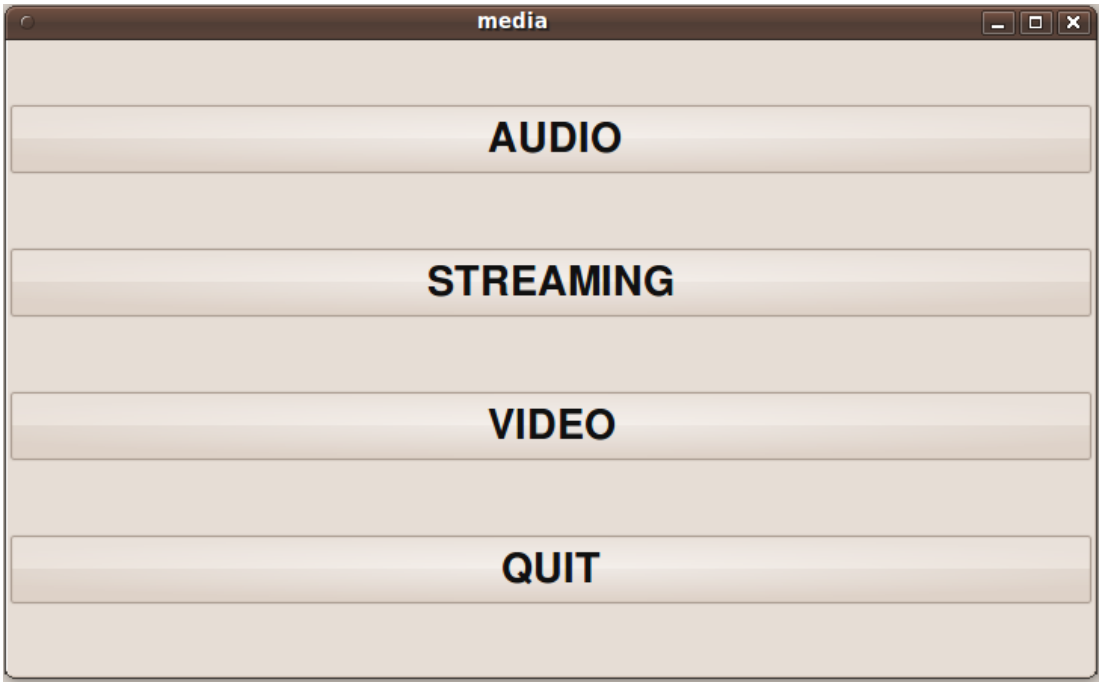
Selviä muutoksia ovat uusien tiedostojen lisääminen: SOURCES-listaan streamingClient.cpp ja HEADERS-listaan streamingClient.h. Jotta GStreameriä pystyttiin käyttämään ohjelmassamme, täytyi kirjasto lisätä LIBS-listaan sekä GStreamerin ja GLibin otsikkotiedostojen polut INCLUDEPATH-listaan. Nämä rivit tarvitaan, jotta qmake osaa tehdä tarvittavat muutokset projektia käännettäessä. Tarpeelliseksi huomattiin myös lisätä rivi CONFIG += no_keywords. Tämä rivi kieltää moc:tä määrittämästä Qt:n avainsanoja signal, slot, foreach ja emit. Rivi tarvittiin, koska GStreamer käyttää avainsanaa signal.

Normaalisti projektitiedostossa olisi myös ohjelman käyttöliittymätiedostot FORMS-listan alla, mutta aiemmissa projekteissa kohtaamiemme ongelmien

takia käyttöliittymäkirjastot jätettiin merkitsemättä. Käytännössä qmake ei huomannut käyttöjärjestelmätiedostoihin tekemiämme muutoksia. Kiersimme ongelman tekemällä muutokset suoraan otsikkotiedostoon, jonka Qt:n uic käännön yhteydessä luo käyttöliittymätiedostosta. Nämä tiedostot löytyvät projektin kääntökansiosta nimellä ui_(luokan_nimi).h ja ovat automaattisesti käyttöliittymätiedostoa luodessa linkitettynä vastaavaan otsikkotiedostoon riippumatta siitä, onko käyttöliittymätiedostoa olemassa vai ei.

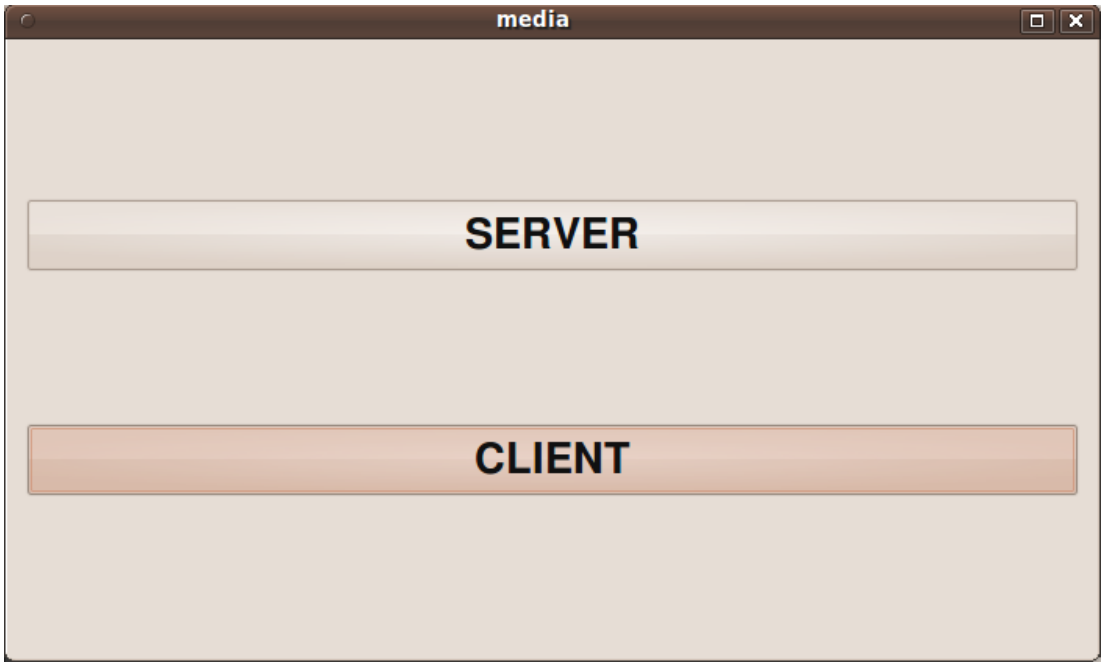
4.2.2 mediaplayer.h ja mediaplayer.cpp

Kuvassa 7 on ohjelmamme päänäkymä.



KUVA 7. Ohjelman pääikkuna

Streamaukselle oli pääikkunassa jo oma nappinsa, joka tarvitsi vain ottaa käyttöön. Suurin yksittäinen muutos oli dialogi, jossa valitaan, käynnistetäänkö serveri- vai asiakasohjelma. Kuvassa 8 on QDialog-widget, joka aukaistaan Streaming-nappia painaessa.



KUVA 8. streamingDialog

Ohjelmakoodissa dialogi ja kaksi lisättyä painonappia näkyvät pieninä muutoksina. Otsikkotiedostoon lisättiin seuraavat määrittelyt:

```
QDialog *streamingDialog;
QGridLayout *streamingLayout;
QPushButton *pushButton_streamingClient;
QPushButton *pushButton_streamingServer;
```

Jotta napit toimisivat, täytyy niille luoda otsikkotiedostossa myös slotit. Koska GStreamer käyttää avainsanaa signal, jouduttiin Qt:n omat avainsanat ottamaan pois käytöstä. Tästä johtuen Qt:n normaali avainsana slots on korvattu sanalla Q_SLOTS.

```
public Q_SLOTS:

    void setAndShowCurrentPage4();
    void setAndShowCurrentPage5();
```

Itse luokkatiedostossa dialogi luodaan muiden käyttöliittymäelementtien yhteydessä setupMainPage-funktiossa, joka on ajojärjestyksessä ensimmäisenä.

Dialogille ja sen layoutille varataan tila muistista käyttämällä avainsanaa new. Widgetiä luodessa määritellään myös, onko widget lapsiwidget vai hallitseeko se itse itseään. Jos widget on lapsiwidget, laitetaan sen vanhempi sulkujen sisään; esimerkissä this viittaa mediaplayer-luokasta tehtyyn olioon. streamingDialogin setGeometry-funktio on QWidget-luokalta peritty funktio, jolla määritetään widgetin koko. Tässä tapauksessa laitoimme kooksi nelikulmion, jonka sivujen pituudet ovat 696 ja 396 pikseliä.

```
streamingDialog = new QDialog(this);
streamingDialog->setGeometry(QRect(0, 0, 696, 396));
streamingLayout = new QGridLayout(streamingDialog);
streamingLayout->setObjectName(QString::fromUtf8("streamingLayout"));
```

Samalla periaatteella luodaan myös dialogin painonapit:

```
pushButton_streamingClient = new QPushButton(streamingDialog);
pushButton_streamingClient->setObjectName(QString::fromUtf8("pushButton_streamingClient"));
```

Kun painonapit on luotu, lisätään ne dialogin muotoiluwidgetiin:

```
streamingLayout->addWidget(pushButton_streamingServer,1,0,1,1);
streamingLayout->addWidget(pushButton_streamingClient,2,0,1,1);
```

connectSignals-funktiossa yhdistetään painonapin clicked-signaali slottiin:

```
QObject::connect(pushButton_streamingClient, SIGNAL(clicked()), this, SLOT(setAndShowCurrentPage5()));
```

Itse slotissa luodaan streamingClient-olio ja laitetaan se näkyville:

```
void mediaplayer::setAndShowCurrentPage5()
{
    streamingDialog->hide();
    if (!streamingClientPage){
          QPointer<streamingClient> streamingClientPage = new streamingClient(this,this);
          this->streamingClientPage = streamingClientPage;
       }
       this->setCentralWidget(this->streamingClientPage);
}
```

4.2.3 streamingClient.h

streamingClient.h-luokka hoitaa itse streamin vastaanoton ja soiton. Tässä osiossa käydään läpi luokan otsikkotiedosto olennaisilta osin.

Alla on otsikkotiedoston ohjelmakoodi kokonaisuudessaan:

```
#ifndef STREAMINGCLIENT_H
#define STREAMINGCLIENT_H

#include <QtGui/QWidget>
#include "ui_streamingClient.h"
#include <gst/gst.h>
#include <glib.h>



/** Import the mediaplayer class */
class mediaplayer;


class streamingClient : public QWidget
{
    /** Qt macro which needed to add singal and slot definitations to the class of audio */
    Q_OBJECT
public Q_SLOTS:
    void initClient();
    void playFunction();
public:
    /** A contructor of the class of streaming */
    streamingClient(mediaplayer *mainPage, QWidget *parent = 0);
    /** An desctructor of the class of streaming */
    ~streamingClient();



    /** All the signals which have been connected from this class */
    void connectSignalsStreamingClient(mediaplayer *mainPage);
private:

    /** An object of the class of streaming */
    Ui::streamingClientClass ui;
    /** An object of the class of main window */
    mediaplayer *mainPage;
};


#endif // STREAMINGCLIENT_H
```

Seuraavalla koodinpätkällä varmistetaan, ettei luokkaa määritellä useaan kertaan:

```
#ifndef STREAMINGCLIENT_H
#define STREAMINGCLIENT_H
```

Seuraavat luokat linkitetään käännösvaiheessa tiedostoon:

```
#include <QtGui/QWidget>
#include "ui_streamingClient.h"
#include <gst/gst.h>
#include <glib.h>
```

QWidget-luokka tarvitaan, koska streamingClient periytyy QWidget-luokasta. ui_streamingClient.h on luokan käyttöliittymätiedosto käännettynä otsikkotiedostoksi. gst.h ja glib.h tarvitaan, koska luokka käyttää GStreameriä.

Luokka periytyy QWidget-luokasta. Tämä mahdollistaa luokalle QWidgetin ominaisuudet.

```
class streamingClient : public QWidget
```

Seuraavaksi määritellään luokan tarvitsemat slotit. Q_OBJECT-makro kertoo kääntäjälle, että Qt:n signaalit ja slotit ovat käytössä luokassa.

```
    Q_OBJECT
public Q_SLOTS:
    void initClient();
    void playFunction();
```

Otsikkotiedostossa määritellään myös luokan funktiot.

```
public:
    /** A contructor of the class of streaming */
    streamingClient(mediaplayer *mainPage, QWidget *parent = 0);
    /** An desctructor of the class of streaming */
    ~streamingClient();
```

Konstruktorille annetaan viittaus ohjelman pääikkunaan. Funktion toinen parametri määrittää, onko streamingClient lapsiwidget. Jos konstruktoria kutsuessa annetaan viittaus toiseen widgetiin, tehdään widgetistä streamingClientin vanhempi.

Viimeisenä luokkaan määritellään pelkästään streamingClientille näkyvät muuttujat: muuttuja, jolla päästään käsittelemään käyttöliittymän komponentteja, sekä osoitinmuuttuja, jolla voidaan viitata ohjelman pääikkunaan.

```
private:

    /** An object of the class of streaming */
    Ui::streamingClientClass ui;
    /** An object of the class of main window */
    mediaplayer *mainPage;
```

4.2.4 streamingClient.cpp

Itse streamingClient-luokan koodi löytyy streamingClient.cpp-tiedostosta. Ensimmäisenä ajojärjestyksessä on konstruktori:

```
//Constructor, inits gstreamer and sets streamingClient as the main widget
streamingClient::streamingClient(mediaplayer *mainPage, QWidget *parent)
    : QWidget(parent)
{

    //inits gstreamer, returns to main menu if unsuccessful
    if(!gst_init_check (NULL, NULL, NULL))
    {
       mainPage->setAndShowMainPage();
       return;
    }


       this->mainPage = mainPage;
       ui.setupUi(this);
       connectSignalsStreamingClient(this->mainPage);
}
```

Ensimmäisenä konstruktorissa käynnistetään GStreamer kutsumalla gst_init_check-funktiota. Tämä funktio toimii samalla testinä, voiko GStreamer käynnistyä, ja jos käynnistyminen ei onnistu, palauttaa funktio boolean-arvon 0. Tätä hyödyntäen funktiota käytetään if-lauseen ehtona. Lauseella palataan ohjelman pääikkunaan, jos GStreamerin käynnistäminen epäonnistuu.

Konstruktorissa laitetaan näkyviin myös streamClient-luokan käyttöliittymä kutsumalla käyttöliittymän setupUi-funktiota. Lisäksi kutsutaan

connectSignalsStreamingClient-funktiota, jossa luokan tarvitsemat signaalit yhdistetään sloteihin.

```
//connectSignalsStreamingClient, connects the needed signals
void streamingClient::connectSignalsStreamingClient(mediaplayer *mainPage)
{
    QObject::connect(ui.pushButton_close, SIGNAL(clicked()), mainPage, SLOT(setAndShowMainPage()));
    QObject::connect(ui.pushButton_connect, SIGNAL(clicked()), this, SLOT(initClient()));
}
```

Ensimmäisellä connectilla yhdistetään pushButton_close-painonappi pääikkunan setAndShowMainPage-funktioon ja streamingClient suljetaan. Toinen nappi yhdistetään streamingClient-luokan initClient-funktioon, jossa itse streamin vastaanotto ja kuuntelu tapahtuvat.

initClient-funktio on streamingClient-luokan pääfunktio. Koko luokan toiminnallisuus on sijoitettu tähän funktioon. Kun initClient-funktiota kutsutaan, ensimmäiseksi määritellään muutamia ohjelman tarvitsemia vakioarvoja.

```
 #define AUDIO_CAPS "application/x-rtp,media=(string)audio,
clock-rate=(int)90000,encoding-name=(string)MPA"
```

AUDIO_CAPS on näistä määrityksistä tärkein, koska ilman niitä ohjelma ei tietäisi, mitä muotoa siirrettävä audio on, eikä siten osaisi purkaa RTP-paketteja saatikka toistaa purettua audiota. Tiedot sisältävät streamin tiedostomuodon, sen sisältämät mediat, enkoodauksen sekä kellotaajuuden.

```
#define AUDIO_DEPAY "rtpmpadepay"
#define AUDIO_DEC   "mad"
#define AUDIO_SINK  "autoaudiosink"
```

Seuraavat kolme määritystä ovat osa ohjelman nykymuotoista perustoimintaa ja liittyvät ohjelman testaukseen. Todellisuudessa nämä kolme tulisi määrittää serverin lähettämien caps-tietojen perusteella. AUDIO_DEPAY määrittää GStreamer-plug-inin, jolla RTP-paketit puretaan. Tässä tapauksessa kyseessä on mpa (eli MPEG Audio, tutummin yleinen mp3-audio). AUDIO_DEC määrittää streamin toistoon käytettävän plug-inin ja AUDIO_SINK plug-inin, jolla audio lopulta lähetetään kaiuttimiin.

Seuraavaksi luodaan GStreamerin tarvitsemat elementit:

```
GstElement *audiosrc;
 GstElement *rtpbin, *pipeline;


GstElement *rtpsrc, *rtcpsrc, *rtcpsink;
GstElement *audiodepay, *audiodec, *audiores, *audioconv, *audiosink;
```

GstElement on perusluokka, josta jokainen pipelinen elementti luodaan. Elementille luodaan toiminnallisuus kutsumalla vastaavaa GStreamerin funktiota. g_assert-funktio on GLib-kirjastoon kuuluva testifunktio, joka lopettaa ohjelman, jos parametrina annetun elementin luonti ei ole onnistunut.

Seuraavassa luodaan pipeline, joka hallitsee muita elementtejä ja huolehtii siten koko streamingClient-luokan toiminnallisuudesta.

```
pipeline = gst_pipeline_new (NULL);
g_assert (pipeline);
```

Seuraavana luodaan toiminnallisuus elementeille, jotka ovat suorassa yhteydessä serveriin käyttäen UDP-protokollaa. Tähän tehtävään käytetään udpsrc- ja udpsink-plug-ineja, udpsrc:tä vastaanottamaan ja udpsinkiä lähettämään dataa.

Jotta elementteihin saataisiin tietyn plug-inin toiminnallisuus, täytyy kutsua gst_element_factory_make-funktiota ja antaa parametreinä halutun plug-inin nimi ja halutun elementin nimi.

g_object_set-funktiolla voidaan muuttaa elementin asetuksia. Funktio ottaa parametreinä ensimmäiseksi olion, jonka asetuksia muutetaan. Seuraavana annetaan asetuksen nimi ja sen jälkeen uusi arvo, johon asetus asetetaan. Funktio osaa muuttaa useita arvoja yhdellä kutsulla, joten parametrien määrä ei ole vakio. Näin ollen funktion viimeisenä parametrinä täytyy aina antaa NULL, jotta funktio tietää, milloin asetuksien muuttaminen lopetetaan.

```
//rtpsrc handles rtp messages received over udp
rtpsrc = gst_element_factory_make ("udpsrc", "rtpsrc");
g_assert (rtpsrc);
g_object_set (rtpsrc, "port", 5002, NULL);
/* we need to set caps on the udpsrc for the RTP data*/
caps = gst_caps_from_string (AUDIO_CAPS);
g_object_set (rtpsrc, "caps", caps, NULL);
gst_caps_unref (caps);
```

rtpsrc ottaa vastaan RTP-dataa UDP:n yli. rtpsrc tarvitsee siis paitsi portin, jota kuunnellaan, myös ohjelmassa aiemmin määritellyt caps-tiedot.

Kuten jokaisessa RTP-yhteyttä käyttävässä ohjelmassa, täytyy myös tässä ohjelmassa siirtää RTCP-dataa. rtcpsrc ottaa vastaan edellä mainittua dataa serveriltä.

```
rtcpsrc = gst_element_factory_make ("udpsrc", "rtcpsrc");
g_assert (rtcpsrc);
g_object_set (rtcpsrc, "port", 5003, NULL);
```

rtcpsink puolestaan lähettää RTCP-tietoja serverille. Sen tyyppinä on udpsink.

```
rtcpsink = gst_element_factory_make ("udpsink", "rtcpsink");
g_assert (rtcpsink);
g_object_set (rtcpsink, "port", 5007, "host", DEST_HOST, NULL);
```

Seuraavalla rivillä luodut elementit lisätään pipelineen:

```
gst_bin_add_many (GST_BIN (pipeline), rtpsrc, rtcpsrc, rtcpsink, NULL);
```

Seuraavaksi vastaanotettu audio täytyy muuttaa muotoon, jossa se voidaan toistaa. audiodepay muuntaa RTP-datan normaaliksi äänidataksi:

```
audiodepay = gst_element_factory_make (AUDIO_DEPAY, "audiodepay");
g_assert (audiodepay);
```

Seuraavat elementit hoitavat audion toiston, aina puretusta datasta kaiuttimiin asti.

```
//audiodec decodes audio into raw audio
audiodec = gst_element_factory_make (AUDIO_DEC, "audiodec");
g_assert (audiodec);

//audioconv and audiores, converts and resamples received audio
//these elements enable audio playback
audioconv = gst_element_factory_make ("audioconvert", "audioconv");
g_assert (audioconv);
audiores = gst_element_factory_make ("audioresample", "audiores");
g_assert (audiores);

//audiosink plays audio
audiosink = gst_element_factory_make (AUDIO_SINK, "audiosink");
g_assert (audiosink);
```

Kun kaikki elementit on luotu, ne voidaan lisätä pipelineen ja linkittää toisiinsa:

```
gst_bin_add_many (GST_BIN (pipeline), audiodepay, audiodec, audioconv, audiores, audiosink, NULL);

res = gst_element_link_many (audiodepay, audiodec, audioconv, audiores, audiosink, NULL);
g_assert (res == TRUE);
```

rtpbin on elementti, joka hallitsee RTP- ja RTCP-yhteyksiä. Tämä elementti helpottaa huomattavasti yhteyden muodostamista ja ylläpitoa.

```
//rtpbin element automatically handles all received and sent rtp packages
rtpbin = gst_element_factory_make ("gstrtpbin", "rtpbin");
g_assert (rtpbin);

gst_bin_add (GST_BIN (pipeline), rtpbin);
```

Jotta rtpbin voisi hallita yhteyttä, täytyy rtpsrc, rtcpsrc ja rtcpsink yhdistää rtpbinin vastaaviin padeihin. rtpsrc yhdistetään binin recv_rtp_sink_0-padiin, samoin kuin rtcpsrc recv_rtcp_sink_0- ja rtcpsink send_rtcp_src_0-padeihin.

```
srcpad = gst_element_get_static_pad (rtpsrc, "src");
sinkpad = gst_element_get_request_pad (rtpbin, "recv_rtp_sink_0");
lres = gst_pad_link (srcpad, sinkpad);
g_assert (lres == GST_PAD_LINK_OK);
gst_object_unref (srcpad);
```

rtpbin käyttää dynaamisia padeja, joiden lopullinen yhteys luodaan vasta ohjelman käytön aikana, kun rtpbin löytää pipelinestä vastaanotettavaa streamia vastaavan depay-elementin. Tämän takia ohjelma tarvitsee

callback-funktion, jota kutsutaan kun ohjelma saa yhteyden. Jotta rtpbinin lähettämä GLib-signaali huomattaisiin, tarvitaan myös seuraava rivi, jolla GLib-signaali yhdistetään callback-funktioon.

```
g_signal_connect (rtpbin, "pad-added", G_CALLBACK (pad_added_cb), audiodepay);
```

Callback-funktiossa yhdistetään rtpbin pipelinen myöhempiin elementteihin.

```
static void
        pad_added_cb (GstElement * rtpbin, GstPad * new_pad, GstElement * depay)
{
  GstPad *sinkpad;
  GstPadLinkReturn lres;

  g_print ("new payload on pad: %s\n", GST_PAD_NAME (new_pad));

  sinkpad = gst_element_get_static_pad (depay, "sink");
  g_assert (sinkpad);

  lres = gst_pad_link (new_pad, sinkpad);


  g_assert (lres == GST_PAD_LINK_OK);
  gst_object_unref (sinkpad);
}
```

Kun yhteys rtpbinin ja pipelinen välillä on luotu, voidaan pipeline laittaa soittotilaan, jolloin streami toistetaan.

```
g_print ("starting receiver pipeline\n");
gst_element_set_state (pipeline, GST_STATE_PLAYING);
```

5 TULOKSET JA JOHTOPÄÄTÖKSET

Opinnäytetyön tuloksena oli määrittelyjen mukainen ohjelma, jolla voitiin luoda sekä serveri, joka lähettää audiostreamia, että asiakas, jolla streamia pystytään kuuntelemaan. Ohjelmassa käytettiin menetelmiä ja teknologioita, joiden hyödyntäminen on mahdollista siinäkin tapauksessa, kun kohdealustana on Maemo-käyttöjärjestelmä. Itse kohdelaitteella emme ohjelmaa valitettavasti päässeet testaamaan, mutta lähiverkossa kahden Linux-koneen välillä ohjelma toimi odotetusti.

Työn toteuttamiseen haastetta toivat käytännössä puhtaalta pöydältä alkanut streamaustoiminnon kehittäminen: tiesimme olemassa olevia teknologioita, mutta emme miten ne käytännössä toimivat ja miten niitä voitaisiin käyttää Qt-ohjelmassamme. Haastetta toi myös dokumentaation ja aiempien käyttäjäkokemuksien näennäinen vähäisyys: projektia suunnitellessa ja määritellessä oli hankala tehdä ratkaisuja, koska tunnuimme olevan ensimmäisiä, jotka yrittivät käyttää GStreameriä nettistreamin lähettämiseen ja vastaanottamiseen. Näin ollen aikaa suunnitteluun ja määrittelyyn kului aiottua enemmän.

Qt:n tuki kolmannen osapuolen teknologioille oli myös yksi selvitettävä asia työssä. Tähän löytyi onneksi esimerkkejä ja dokumentaatiota enemmänkin, ja GStreamer-toiminnallisuuden rakentaminen Qt-ympäristöön onnistui yllättävänkin helposti. Pieniä hankaluuksia tosin tuotti kokemuksen puute Linux-käyttöjärjestelmien kanssa, ja GStreamer- ja GLib-kirjastojen löytäminen vei vähän aikaa toteutusvaiheessa.

Loppujen lopuksi projekti kuitenkin onnistui odotettuun tapaan. Saimme kokemusta Linux-järjestelmistä ja Qt:n käyttämisestä kolmannen osapuolen teknologioiden kanssa sekä tutustuimme GStreameriin multimediakirjastona.

Työn valmistumisen ja raportin palauttamisen välisenä aikana ovat teknologiat ja käytetyt järjestelmät kerinneet muuttua useaankin otteeseen,

ovathan kyseessä paljon käytetyt ja suositut teknologiat. Näin ollen työmme ei ole enää ihan yhtä ajankohtainen.

Qt:hen on tullut Phononin lisäksi myös QtMultimedia-moduuli, joka tuo uuden mahdollisuuden kevyeen matalan tason multimediatoistoon. Phonon on vielä monipuolisempi mutta raskaampi vaihtoehto. Nokia on myös julkaissut uuden QtMobility-API:n (Application Programming Interface), joka on suunniteltu sisältämään kevyitä mutta tehokkaita, erityisesti mobiililaitteille suunniteltuja ratkaisuja esimerkiksi paikannukselle, kameralle, videokuvan ottamiselle yms. QtMobilityn sisältämässä QtMultimedia-moduulissa ei kuitenkaan ole ohjelmaamme jo ennen opinnäytetyötä sisältyneitä toiminnallisuuksia, ja Phononin puuttuessa QtMobility-API:sta ohjelman kääntäminen QtMobilitylle ei olisi aivan yksinkertaista. (20; 21.)

Myös Phononin kehitys jatkuu KDE-projektin alaisuudessa, uusin julkaisu on versio 4.5.1. Tosin dokumentaatiosta ei tällä hetkellä selvinnyt, onko projektissa implementoitu RTP-protokollaan perustuva netin yli streamaaminen. (22.)

GStreamerin kehitys on edelleen jatkuvaa ja plug-ineja päivitetään aktiivisesti; kirjoituksen aikaan uusin vakaa julkaisu on kesäkuulta 2011: GStreamer Core 0.10.35, Base 0.10.35 ja Good 0.10.30. Suuria muutoksia RTP-protokollaa käyttäviin plug-ineihin ei ole kuitenkaan tullut. (23.)

Maemo-käyttöjärjestelmä, samoin kuin N900-puhelin, ovat jääneet historiaan mutta Maemon epävirallinen jälkeläinen MeeGo on vielä hengissä: MeeGoon pohjaava Harmattan-järjestelmä on käytössä Nokian uudessa N9-puhelimessa. (24.)

Loppuyhteenvetona mielestäni ohjelmamme on hyvin kehityskelpoinen, joskin vielä alkeellinen mediasoitin, jossa varsinkin streamausosassa on paljon potentiaalia jatkokehitykselle. Esimerkiksi ohjelmamme on tällä hetkellä täysin riippuvainen koodiin määritellyistä parametreistä, jotka tulisi siirtää netin yli käyttäen esimerkiksi Qt:n erittäin käyttökelpoista QtNetwork-

moduulia. Netin yli voisi siirtää esimerkiksi myös soitettavan kappaleen tiedot, soitettavan soittolistan tiedot sekä mahdolliset toistoon liittyvät komennot kuten kelaus, kappaleen vaihtaminen tai soiton laittaminen tauolle.

LÄHTEET

1. Qt (framework). 2011. Saatavilla: http://en.wikipedia.org/wiki/Qt_%28framework%29. Hakupäivä 9.12.2011.

2. Evolution of Maemo. Saatavilla: http://maemo.org/intro/maemo_history. Hakupäivä 20.3.2011.

3. MeeGo. 2011. Saatavilla: http://en.wikipedia.org/wiki/MeeGo. Hakupäivä 9.12.2011.

4. Software Platform. Saatavilla: http://maemo.org/intro/platform. Hakupäivä 20.3.2011.

5. Qt 4.6 Whitepaper. 2009. Saatavilla: http://qt.nokia.com/files/pdf/qt-4.6-whitepaper. Hakupäivä 20.3.2011.

6. Modular Class Library. 2011. Saatavilla: http://qt.nokia.com/products/library. Hakupäivä 9.12.2011.

7. qmake Manual. 2010. Saatavilla: http://doc.qt.nokia.com/4.6/qmake-manual.html. Hakupäivä 20.3.2011.

8. Widgets and Layouts. 2010. Saatavilla: http://doc.qt.nokia.com/4.6/widgets-and-layouts.html. Hakupäivä 4.7.2011.

9. Signals and Slots. 2010. Saatavilla: http://doc.qt.nokia.com/4.6/signalsandslots.html. Hakupäivä 4.7.2011.

10. Meta-Object System. 2010. Saatavilla: http://doc.qt.nokia.com/4.6/metaobjects.html. Hakupäivä 9.12.2011.

11. QObject Class Reference. 2010. Saatavilla: http://doc.qt.nokia.com/4.6/qobject.html#connect. Hakupäivä 4.7.2011.

12. Phonon Overview. 2010. Saatavilla: http://doc.qt.nokia.com/4.6/phonon-overview.html. Hakupäivä 4.7.2011.

13. RTP: A Transport Protocol for Real-Time Applications. 2003. Saatavilla: http://www.ietf.org/rfc/rfc3550.txt. Hakupäivä 4.7.2011.

14. GStreamer. 2011. Saatavilla: http://en.wikipedia.org/wiki/GStreamer. Hakupäivä 9.12.2011.

15. Taymans, Wym - Baker, Steve - Wingo, Andy - Bultje, Ronald S. - Kost, Stefan, Bins and pipelines. Saatavilla: http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/section-intro-basics-bins.html. Hakupäivä 9.12.2011.

16. GStreamer Plug-ins splitup. Saatavilla: http://gstreamer.freedesktop.org/documentation/splitup.html. Hakupäivä 4.7.2011.

17. Overview of available plug-ins. Saatavilla: http://gstreamer.freedesktop.org/documentation/plugins.html. Hakupäivä 4.7.2011.

18. Scrum (development). 2011. Saatavilla: http://en.wikipedia.org/wiki/Scrum_%28development%29. Hakupäivä 9.12.2011.

19. Waterfall model. 2011. Saatavilla: http://en.wikipedia.org/wiki/Waterfall_model. Hakupäivä 9.12.2011.

20. Qt Mobility 1.2: Qt Mobility Project Reference Documentation. 2011. Saatavilla: http://doc.qt.nokia.com/qtmobility/index.html. Hakupäivä 9.12.2011.

21. Qt 4.7: QtMultimedia Module. 2011. Saatavilla: http://doc.qt.nokia.com/4.7-snapshot/qtmultimedia.html. Hakupäivä 9.12.2011.

22. Development/Tutorials/Phonon/Introduction. 2011. Saatavilla: http://techbase.kde.org/Development/Tutorials/Phonon/Introduction. Hakupäivä 9.12.2011.

23. GStreamer Core 0.10.35, Base Plugins 0.10.35, Good Plugins 0.10.30 stable release. 2011. Saatavilla: http://gstreamer.freedesktop.org/news. Hakupäivä 9.12.2011.

24. Nokia N9. 2011. Saatavilla: http://en.wikipedia.org/wiki/Nokia_N9. Hakupäivä 9.12.2011.