Lappeenrannan teknillinen yliopisto
Teknillistaloudellinen tiedekunta
Tietotekniikan osasto
Diplomityö

Monialustaiset avoimen lähdekoodin ikkunointikirjastot

Työn tarkastajat: Professori Kari Smolander
TkT Pauli Kuosmanen

Työn ohjaaja: TkT Pauli Kuosmanen

Kerava, 2.4.2010

Juho Valonen
Koivikontie 2 B 6
04260 Kerava
juho.valonen@lut.fi
Puhelin: 050 54 66 348

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknillistaloudellinen tiedekunta
Tietotekniikan osasto

Juho Valonen

Monialustaiset avoimen lähdekoodin ikkunointikirjastot

Diplomityö

2010

112 sivua, 2 kuvaa, 10 taulukkoa ja 8 liitettä

Tarkastajat: Professori Kari Smolander
TkT Pauli Kuosmanen

Hakusanat: Ikkunointikirjasto, Qt, GTK+, GTKmm, WxWidgets

Tämän tutkimuksen tavoitteena oli löytää vastauksia siihen, mikä on tärkeimpien avoimen lähdekoodin kirjastojen toteutuksen tämän hetkinen taso. Työssä tutkittiin WxWidgetsin, GTK+:n ja Qt:n toteutuksen tasoa käytämällä hyväksi McCaben, Henry&Kafuran ja Chidamberin & Kemererin esittelemiä staattisia menetelmiä. Lisäksi ikkunointikirjastojen lähdekoodin käännetty koko mitattiin eri käyttöjärjestelmissä. Tutkimuksessa esitellään valittujen kirjastojen arkkitehtuuri ja vertaillaan esiteltävien kirjastojen arkkitehtuurisia ratkaisuja toisiinsa. Tämän jälkeen arvioidaan staattisten menetelmien tuottamien tuloksien merkitystä kahdesta näkökulmasta: mitä tulokset kertovat kirjastoista kun niitä verrataan toisiinsa ja mitä silloin kun niitä verrataan kyseisen kirjaston ja muiden kirjastojen arkkitehtuurisiin ratkaisuihin.

Tutkimuksessa havaittiin Qt:n sisältävän kaikkein vähiten kirjaston ulkopuolisia riippuvuuksia. Tämän lisäksi sen huomattiin sisältävän muista kirjastoista puuttuvia ominaisuuksia. Osittain edellämainitusta syystä johtuen Qt:n ongelmakohdaksi havaittiin joidenkin sen osien suuri monimutkaisuus ja tästä seuraava mahdollinen vaikeasti ylläpidettävä lähdekoodi. GTK+:n lähdekoodi sisältää muita kirjastoja vähemmän sisäisiä riippuvuuksia samaan kirjastoon, on korkeammalla abstraktiotasolla ja kirjaston osat ovat siirrettävissä ja erotettavissa toisistaan. Joissakin kohdissa GTK+:n ja etenkin sen C++-rajapinnan GTKmm:n lähdekoodi on kuitenkin tarpeettoman monimutkaista. WxWidgetsin toteutuksen havaittiin Qt:n tavoin olevan hyvin itsenäinen kokonaisuus, WxWidgetsin lähdekoodin monimutkaisuus on useimmiten jotakin GTK+:n ja Qt:n väliltä. WxWidgets on Qt:a vähemmän itsenäinen mutta kuitenkin itsenäisempi kuin GTK+. Kuten muutkin kirjastot myös wxWidgetsillä on omat kohtansa, joissa sen lähdekoodi on tarpeettoman monimutkaista.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Department of Information Technology

Juho Valonen

Multiplatform open-source widget libraries

Master's thesis

2010

112 pages, 2 figures, 10 tables and 8 attachments

Supervisors: Professor Kari Smolander
Dr. Tech. Pauli Kuosmanen

keywords: Widget library, Widget, Qt, GTK+, GTKmm, WxWidgets

The target of this research was to find answers to the current quality of the implementation of the open source widget toolkits. In the research wxWidgets, GTK+ and Qt implementation quality was studied by using statical methods proposed by McCabe, Henry&Kafura and Chidamber&Kemerer. Additionally the compiled source code size was measured in the different operating systems. In the research the architecture of the widget toolkits is presented and different architectural solutions of the libraries are compared against each other. After that the meaning of the results coming from the statical methods is analyzed from the two perspectives: what the results mean when the libraries are compared against each other and what then when they're compared to their own architectural solutions and architectural solutions of the other libraries.

In the research it was noticed that Qt contains less external dependencies to the external libraries than other libraries. Additionally it was also noticed that it contains features missing from the other libraries. Partly previously mentioned reasons problematic areas in Qt were found. Some of its parts are unnecessarily complex and thus they can be hard to maintain. GTK+ contains less internal dependencies than other libraries. It is of higher abstraction and the parts of the GTK+ are easier to move and to separate. At some places the source code of the GTK+ and especially its C++-interface GTKmm is unnecessarily complex. The implementation of the WxWidgets, like Qt, was identified as being independent. Most of the time the complexity of the WxWidgets source code is something between Qt and GTK+. WxWidgets is less independent than Qt, but more independent than GTK+. Like the other libraries, source code of the WxWidgets has some locations where it is more complex than in the other libraries.

Alkusanat

Haluan kiittää Kari Smolanderia ja Pauli Kuosmasta työn ohjauksesta. Vaimoani Jenniä haluan kiittää erinomaisista neuvoista, aikataulun mahdollistamisesta ja henkisestä tuesta.

Sisällysluettelo

Lyhenteet

API	Application Programming Interface
CDE	Common Desktop Environment
GIMP	GNU Image Manipulation Program
GPL	General Public License
GNU	GNU's Not Unix
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
ISO	International Organization for Standardization
KDE	K Desktop Environment
LGPL	Lesser General Public License
MDI	Multiple Document Interface
MFC	Microsoft Foundation Classes
MOC	Meta Object Compiler
OSI	Open Source Iniative
PDF	Portable Document Format
QPL	Qt Public License
RTTI	Run-Time Type Information
STL	Standard Template Library
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
UIC	User Interface Compiler
XML	eXtensible Markup Language

1.JOHDANTO

Tämän diplomityön tarkoitus on antaa käsitys tunnetuimpien avoimen lähdekoodiin ikkunointikirjastojen toteutuksen tasosta. Motivaationa työhön on Diomidis Spinellisin julkaisema artikkeli international conference on software engineeringissä[1]. Spinellis arvio artikkelissaan avoimen lähdekoodin käyttöjärjestelmäytimien toteutuksen laatua käyttäen hyväkseen staattisia menetelmiä. Spinellis jättää tutkimuksensa ulkopuolelle kuitenkin kokonaan käyttöjärjestelmän tärkeän osan eli graafisen käyttöliittymän ja sen toteuttavat ikkunointikirjastot. Tämän diplomityön on tarkoitus on vastata tähän tarpeeseen.

Spinellis analysoi artikkelissaan C-pohjaista proseduraalista lähdekoodia. Tässä työssä analysoitavat ikkunointikirjastot perustuvat kuitenkin oliomalliseen lähdekoodiin. Oliomallisen lähdekoodin analysointiin on kehitetty useita staattisia menetelmiä. Koska Tim Littlefairin väitöskirjassaan esittämät perustelut menetelmien valintaan koettiin työn kannalta toimivimmaksi ratkaisuksi, päätettiin valita samat menetelmät kuin hän on valinnut[2].

Koska haluttiin että tehty analyysi olisi hyödynnettävissä Spinellisin tutkimuksen lisänä, useista tarjolla olevista avoimen lähdekoodin ikkunointikirjastoista valittiin kahden suositun Linux-jakelun SuSen ja Ubuntun käyttämien työpöytien KDE:n ja Gnomen ytimissä toimivat ikkunointikirjastot Qt ja Gtk+. Jotta tutkimukseen saataisiin myös täysin minkään tietyn käyttöjärjestelmän työpöydän kehityksestä riippumaton kirjasto, valittiin WxWidgets.

Itse tutkimus suoritetaan pääosin Littlefairin väitöskirjassaan esittelemillä menetelmillä[2]. Littlefairin tavoin lähdekoodin tasoa arvioidaan McCaben[3], Henryn & Kafuran[4] ja Chidamber & Kemererin[5] esittämillä menetelmiä. Tämän lisäksi tutkimuksessa käytetään hyväksi tietoa käännetyn lähdekoodin viemästä tilasta. Työn toisisijaisena tavoitteena on antaa käsitys ikkunointikirjastojen tämän hetkisestä arkkitehtuurista. Esiteltävät ominaisuudet on valittu valitsemalla yhteiset pääkohdat

käytetyn kirjastojen arkkitehtuuria esittelevän lähdekirjallisuuden sisällysluetteloista. Jokainen kirjastokohtainen toteutustapa esitellään käytännönläheisesti ja siten että lukijalle jää käsitys kyseisen kirjaston tarjoamasta toteutustavasta. Lopuksi esitellään tutkimustulokset ja pohditaan niiden merkitystä. Tutkimuksen toisena toissijaisena tavoitteena on tarjota lukijalle mahdollisuus ymmärtää kirjastojen välisiä eroja niiden heikkouksia ja vahvuuksia.

Luvussa 2 esitellään McCaben, Henryn & Kafuran[4] ja Chidamberin & Kemererin[5] menetelmät. Luvussa 3 kerrotaan tarkemmin kirjastojen ja niiden ominaisuuksien valinnasta, sekä kerrotaan kirjastojen taustoista. Luvuissa 4, 5, 6 esitellään kolmen valitun ikkunointikirjaston eli Qt:n, xWxWidgetsin ja GTK+:n arkkitehtuuri. Luvussa 7 pohditaan miten esitellyt arkkitehtuurit poikkeavat toisistaan. Luvussa 8 kerrotaan kuinka analyysi tehtiin ja pohditaan analyysitulosten merkitystä. Luvussa 9 tehdään yhteenveto työn sisällöstä.

2.TUTKIMUSMENETELMÄT

Tässä luvussa kerrotaan ensin miten tutkimusmenetelmät on valittu. Tämän jälkeen esitellään tutkimuksessa käytettävät menetelmät.

2.1 Tutkimusmenetelmien valinta

Littlefair käyttää tutkimusmenetelmien valinnassa hyväkseen tavoite/kysymys/metriikka-menetelmää. Ensin hän määrittelee itselleen tavoitteen, jonka hän haluaa saavuttaa metriikkojen avustuksella. Väitöskirjassaan hän käyttää tavoitteenaan Boehmin hierarkista määritelmää.[6] Tämän jälkeen hän luo sarjan dikotomisia eli kyllä/ei kysymyksiä jokaisesta tavoitteen määrittelemästä ominaisuudesta. Esimerkiksi, onko järjestelmä koottu ymmärrettävistä moduleista. Tämän jälkeen hän etsii metriikat, jotka kykenevät tuottamaan todistusaineistoa aiemmin luotuihin kysymyksiin vastaamiseksi. Littlefair rakentaa kysymysaineiston perusteella kolme kategoriaa, joihin hänen metriikkansa tulevat perustumaan.[2]

Proseduraaliset metriikat kategoriassa Littlefair pyrkii saavuttamaan näkemyksen ohjelmasta matalimmalla mahdollisella lähdekooditasolla. Tämän kategorian keskeisenä analyysiyksikkönä ovat yksittäiset aliohjelmat tai proseduurit. Tutkimustuloksesta voidaan arvioida lähdekoodin määrää ja sen organisointia aliohjelma-tasolla. Littlefairin valitsema McCaben metriikka on yksi tunnetuimmista proseduraalisista metriikoista[2].

Rakenteelliset metriikat tarkastelevat lähdekoodin rakennetta proseduraalisia metriikoita korkeammalta abstraktiotasolta. Niiden kantavana ideana on nähdä ohjelmisto moduulien välisten suhteiden verkkona. Näiden metriikoiden painopisteenä on siis ohjelmistomodulien väliset suhteet, eikä niiden sisäinen rakenne. Rakenteellisten metriikoiden tuottamista tutkimustuloksista voidaan arvioida enemmän ohjelmistoa tai ohjelmiston osaa kuin ohjelmiston yksittäistä modulia. Littlefairin työssä rakenteellisia metriikoita edustamaan on valittu Henryn & Kafuran menetelmä. Littlefair katsoo väitöskirjassaan Henryn ja Kafuran töiden olleen lähtökohta koko rakenteellisten

menetelmien tutkimukselle[2].

Olio-suunnitteluun keskittyvät metriikat tarkastelevat lähdekoodia keskittyen lähdekoodin modulien eli luokkien toteutukseen. Niiden tarkoituksena on varmistaa olio-suunnittelun tarkoituksenmukaisuus luokkia tutkimalla. Menetelmät ovat abstraktiotasoltaan matalammalla tasolla kuin rakenteelliset metriikat, koska keskittyvät enemmän moduleihin kuin niiden välisiin suhteisiin. Proseduraaliset menetelmät ovat kuitenkin olio-suunnitteluun keskittyviä menetelmiä matalammalla tasolla, koska luokat muodostuvat aliohjelmista. Littlefair on valinnut olio-suunnittelun analysointiin Chidamber & Kemererin esittämät menetelmät, koska kyseiset menetelmät ovat kaikista olio-suunnitteluun keskittyvistä menetelmistä eniten viitattuja[2].

2.2 McCabe

Thomas McCabe loi omaa sukunimeään kantavan metriikan vuonna 1976 ratkaisemaan ohjelmistomoduulien ylläpidettävyyteen ja testattavuuteen liittyviä ongelmia. Hänen metriikkansa avulla voidaan havaita yksittäisen moduulin monimutkaisuus ja näin ollen kohdistaa testaus- ja ohjelmistokehitys resurssit lähdekoodin oikeisiin osiin.[3]

McCaben metriikka perustuu matemaattisen graafiteorian osaan, jossa mitataan graafin lineaarisesti riippumattomien polkujen määrää eli niiden polkujen määrää joista kaikki mahdolliset moduulin suorituspolut muodostuvat. Metriikassa muodostetaan suunnattu graafi mittauksen kohteena olevasta ohjelmamoduulista siten että yksittäinen ehtolauseeton ohjelmalohko muodostaa graafin kaaren ja ehtolause muodostaa graafin solmun.[3]

McCaben cyclomaattinen kompleksisuus voidaan laskea kaavasta:

v(G) = e – n + p, missä

e on graafin kaarien määrä,

n on graafin solmujen määrä,

p on yhdistettyjen komponenttien määrä(return lausekkeiden määrä)

Kaava 1: McCaben kompleksisuuden laskeminen

Kaavan käytön helpottamiseksi Mills[6] on kuitenkin johtanut ylläolevan kaavan yksinkertaisempaan muotoon:

v(G) = D + 1, missä

D on lähdekoodissa olevien ehtolausekkeiden määrä

Kaava 2: McCaben kompleksisuuden laskeminen

Metriikka perustuu siihen olettamukseen että ohjelma, jossa on yksinkertaisia ehtolauseita on yksinkertaisempi kuin ohjelma jossa ehtolauseita on paljon.

2.3 Henry & Kafura

Henryn ja Kafuran kehittämässä menetelmässä lähestytään ohjelmiston laatua modulien välisten yhteyksien näkökulmasta. Metriikka mahdollistaa suunnittelun ongelmakohtien löytämisen jo lähdekoodia kirjoitettaessa, mutta on myös käytettävissä ylläpidettävyyden arvioimiseen[4]. Lähtökohtana mittaukselle toimii ohjelmistomodulien välisen informaatiovuon käsite. Informaativuo ja sitä kautta tämä metriikka pohjaa seuraaville määritelmille[4].

On olemassa globaali informaatiovuo modulilta A modulille B globaalin tietorakenteen D kautta, jos moduli A tallettaa informaation D:hen ja B hakee informaation D:stä.

Määritelmä 1: globaali informaatiovuo[4]

On olemassa paikallinen informaatiovuo modulilta A modulille B jos yksi tai useampi seuraavista pitää paikkansa.

1) A kutsuu B:tä,
2) B kutsuu A:ta ja A palauttaa arvon B:lle, jota B välittömästi hyödyntää tai
3) C kutsuu A:ta ja B:ta ja välittää A:n palauttaman arvon B:lle

Määritelmä 2: paikallinen informaatiovuo[4]

On olemassa suora paikallinen informaatiovuo modulilta A modulille B jos määritelmän 2 vaihtoehto 1 pitää paikkansa paikalliselle informaatiovuolle
Määritelmä 3: suora paikallinen informaatiovuo[4]

On olemassa epäsuora paikallinen informaatiovuo modulilta A modulile B jos määritelmän 2 vaihtoehdot 2 tai 3 pitävät paikkansa paikalliselle informaativuolle.
Määritelmä 4: Epäsuora paikallinen informaatiovuo[4]

Proseduurin A sisääntulovuo (eng. Fan-in) on yhtä kuin paikalliset proseduuriin kohdistuvat informaatiovuot sekä tietorakenteet joista A hakee informaatiota.
Määritelmä 5: sisääntulovuo[4]

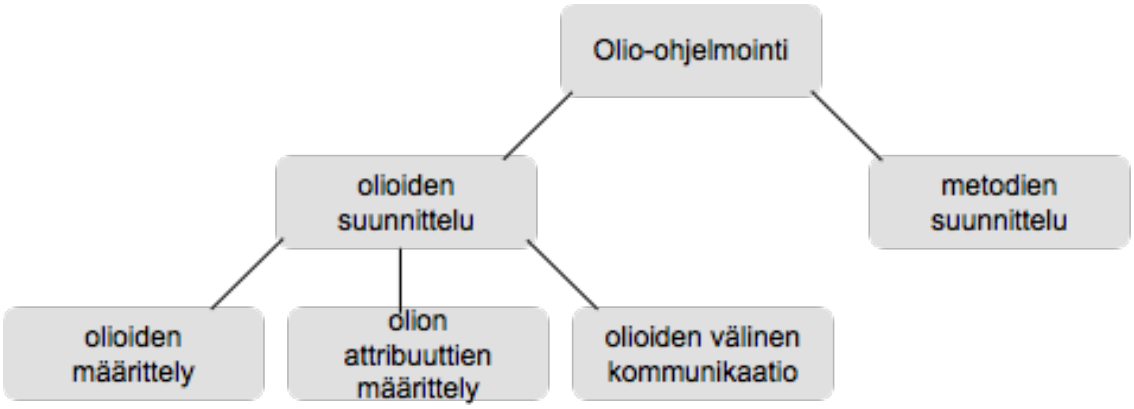
Proseduurin A ulostulovuo (eng. Fan-out) on yhtäkuin paikalliset proseduurista muualle kohdistuvat informaatiovuot sekä tietorakenteet joiden arvoa A päivittää.
Määritelmä 6: ulostulovuo[4]

Joskus kun halutaan arvioida proseduurin tai ohjelmistomodulin kompleksisuutta Henryn ja Kafuran menetelmällä ulos- ja sisääntulovuot yhdistetään johonkin koodin sisäistä kompleksisuutta kuvaavaan metriikkaan. Tällainen metriikka voi olla esimerkiksi McCaben metriikka tai yksinkertaisesti lähdekoodirivien määrä[4].

2.4 Chidamber & Kemerer

Shyam Chidamberin ja Chris Kemererin kehittämät metriikat pyrkivät mittaamaan

lähdekoodin oliototeutuksen laadukkuuden. Menetelmät nojaavat kokeneilta olio-ohjelmoijilta kerättyyn tietoon[5]. Chidamberin ja Kemererin menetelmän lähtökohtana toimii Boochin määrittelemät olio-ohjelmoinnin peruskäsitteet. Booch näkee olio-ohjelmoinnin eroavan merkittävällä tavalla perinteisestä ohjelmoinnista sen sisältämän olioiden suunnittelun osalta.



***Kuva 1: Boochin määrittely oliohjelmoinnille*[6]**

Chidamber ja Kemerer pohjaavat menetelmänsä Kuvan 1 olioiden suunnitelun kolmeen alakategoriaan. Alakategorioista he johtavat oliosuunnittelun olevan empiirinen ja suhteellinen olioista koostuva järjestelmä. Artikkelissaan[6] he muuttavat empiirisen ja suhteellisen järjestelmän, muodolliseksi suhteelliseksi järjestelmäksi, jota hyväksi käyttäen voidaan sitten johtaa seuraavat metriikat.

Painotettujen metodien määrää kuvaava metriikka[5] lasketaan summaamalla kaikkien luokan metodien staattinen kompleksisuus yhteen. Metriikka perustuu näkökulmalle siitä että suurempi metodien määrä ja kompleksisuus lisäävät luokan ylläpitoon ja kehittämiseen tarvittavaa aikaa. Myöskin luokan sovelluskohtaisuus kasvaa painotettujen metodien määrän mukana.

Luokan perimäketjun syvyyttä kuvaava metriikka[5] lasketaan, laskemalla luokan isovanhempien määrä. Metriikan ajatuksena on että kun luokan isovanhempien määrä kasvaa, kasvaa myös luokan monimutkaisuus, luokan metodien määrän kasvamisen seurauksena.

Luokan perivien lasten määrää kuvaavan metriikan[5] idea nojaa kolmeen huomioon oikeanlaisesta luokkien perimärakenteesta. Luokkarakenteen tulisi ennemmin olla syvä kuin leveä, koska syvä luokkarakenne lisää luokkien uudelleenkäytettävyyttä.[5]. Luokilla ei ole hyvä olla samaa määrää lapsi-luokkia, koska ylempänä hierarkiassa olevilla luokilla tulisi olla enemmän lapsia kuin alempana olevilla[5]. Luokan lasten määrä antaa hyvän kuvan luokan vaikutuksesta suunnitteluun, enemmän lapsia sisältävä luokka vaikuttaa suunnitteluun enemmän ja on täten monimutkaisempi[5].

Luokkien välistä kytkentää kuvaava metriikka[5] kuvaa kahden luokan välisten kytkentöjen määrää, poislukien perinnän kautta muodostuvat kytkennät. Myöskin poislukien assosiatiiviset kytkennät eli kytkennät jotka tapahtuvat välittävan luokan kautta. Metriikka pohjautuu näkökulmaan siitä miten laajamittainen, perimähierarkian ulkopuolinen, olioiden kytkentä vähentää luokkien uudelleenkäytettävyyttä[5]. Mitä itsenäisempi luokka on, sitä helpommin sitä voidaan uudelleenkäyttää[5].

Luokan vastausjoukkoa kuvaava metriikka[5] määrittää metodien määrän joita voidaan kutsua luokan ulkopuolelta, sekä niiden metodien määrän joita luokka itse voi kutsua normaalisti tai vastauksena viestiin[5]. Edellämainittujen metodikutsujen suuren määrän katsotaan kasvattavan luokan monimutkaisuutta ja vaikeuttavan sen testausta.

Luokan metodien koheesion puutetta kuvaava metriikka[5] perustuu luokan metodien jäsenmuuttujien käyttöön. Jokaiselle metodille muodostetaan sen käyttämistä jäsenmuuttujista joukko. Kaikista joukoista lasketaan ne joukot, joilla ei ole yhteisiä jäsenmuuttujia muiden joukkojen kanssa. Näiden joukkojen määrä kuvaa luokan metodien koheesion puutetta. Puutteellinen koheesio nähdään ongelmalliseksi kolmesta eri syystä[5]: se kertoo luokan kapsuloinnin puutteesta, siitä että luokka pitäisi todennäköisesti pilkkoa useampaan luokkaan ja siitä että luokka on liian monimutkainen.

3. TUTKIMUSKOHTEET

Kappaleessa kerrotaan käsiteltävien kirjastojen taustasta ja siitä miksi juuri nämä kirjastot ja ominaisuudet valittiin tutkimuskohteiksi. Tämän lisäksi kerrotaan kirjastojen lisensseistä.

3.1 GPL ja LGPL

Avoimella lähdekoodilla tarkoitetaan lähdekoodia, joka on lisensoitu eri tavalla kuin perinteisesti on ohjelmistokehityksen yhteydessä totuttu. Perinteisessä ohjelmistokehityksessä ohjelmisto usein lisensoidaan niin että ainoastaan ohjelmaa kehittävä taho pääsee näkemään sovelluksen lähdekoodin sen kirjoitetussa muodossa[7]. Mutta kuten Ashley Beard tutkimuksessaan mainitsee tämä on avoimen lähdekoodin yhteydessä toisin. Ohjelmiston käyttäjät voivat lukea, muuttaa ja välittää edelleen ohjelmistoa tai sen lähdekoodia. Mikäli käyttäjä muuttaa ohjelmiston lähdekoodia tulee hänen toimittaa muutokset ohjelmiston alkuperäisen kehittäjän saataville[7].

Erilaisia avoimen lähdekoodin lisenssejä on olemassa todella paljon, mutta niistä ehkä tunnetuin on GPL eli GNU Public License. GPL sallii samanlaisen vapauden käsitellä lähdekoodia kuin muutkin avoimen lähdekoodin lisenssit. Käyttäjän näkökulmasta sillä on kuitenkin eräs merkittävä ero. GPL rajoittaa lisenssinsä alaisen lähdekoodin käytön niin että suljettu lähdekoodi katsotaan GPL:n alaiseksi, mikäli suljetun lähdekoodin ohjelma käyttää GPL:n alaista lähdekoodia tai vain linkittyy siihen. LGPL eli Lesser GNU Public License on GPL:n versio, joka on muuten hyvin samankaltainen GPL:n kanssa, mutta sisältää yhden merkittävän eron. LGPL katsoo että jos ohjelmisto linkitetään kirjastoon, kirjasto ja ohjelma ovat toisistaan erillisiä kokonaisuuksia ja näin ollen niiden lisenssien ei tarvitse olla yhteensopiva. GPL sen sijaan katsoo ohjelmaa ja sen käyttämää kirjastoa kokonaisuutena, joka GPL:n mukaan on automaattisesti GPL:n alainen. Tästä seuraa että LGPL:n mukaista ikkunointikirjastoa voidaan käyttää

kaupalliseen ohjelmistokehitykseen ja GPL:n mukaista ei voida käyttää[7].

3.2 Tutkimuskohteiden valinta

Kuten johdannossa mainitaan, työn motivaationa olleessa Spinelliksen tutkimuksessa jätetään graafinen käyttöliittymä käsittelemättä[1]. Jotta tätä diplomityötä voitaisiin pitää Spinelliksen tutkimuksen jatkeena tulisi valittavien kirjastojen olla käytössä avoimen lähdekoodin graafisissa työpöytäympäristöissä. Tämän lisäksi ikkunointikirjastoilta edellytettiin monialustaisuutta eli niiden tulisi toimia kaikilla kolmella suositulla käyttöjärjestelmällä eli Windowsilla, Linuxilla ja Mac OS X:lla. Koska tutkimusmenetelmäksi oli valittu oliopohjaisia menetelmiä tulisi niiden tämän lisäksi omata mahdollisuus oliopohjaiseen ohjelmointiin. Koska käytettävänä analyysityökaluna olisi ainoastaan C:tä ja C++:aa ymmärtävät työkalut päätettiin kirjastoilta edellyttää C- tai C++-pohjaista rajapintaa. Koska tutkimuksen haluttiin olevan kaupallisen yrityksen käytettävissä, päätettiin lukea tutkimuksen ulkopuolelle kaikki GPL:n alainen lähdekoodi. Erilaisten lisenssien monimutkaisuuden takia nähtiin tarpeelliseksi edellyttää tutkimukseen valittavalta kirjastolta yhteensopivuutta LGPL:n kanssa.

Linux journal-lehden lokakuun 2008 artikkelissa avoimen lähdekoodin työpöydistä mainitaan kahden työpöytäjärjestelmän olevan tällä hetkellä suosituimpien jakeluiden eniten käyttämiä, nämä työpöytäjärjestelmät ovat GTK+:aan perustuva Gnome ja Qt:n nojaava KDE[8]. Koska GTK+ ja Qt mahdollistavat C:n ja C++:n käytön, toimivat kaikilla käyttöjärjestelmillä ja ovat yhteensopivia suljetun lähdekoodin lisenssien kanssa, päätettiin ne valita. Jotta tutkimusalueesta ei tulisi täysin keskittynyt Linuxin työpöytäjärjestelmiin päätettiin valita kirjasto, joka olisi syntynyt monialustaisiin projekteihin ilman liityntää tiettyyn työpöytäjärjestelmään. Esiteltävät ja vertailtavat kirjastojen ominaisuudet valittiin tutkimalla kunkin kirjaston dokumentaatiota ja valitsemalla ominaisuuksia, jotka kaikki kirjastot toteuttaisivat. Näistä ominaisuuksista sitten saatiin arkkitehtuurikappaleiden vertailtavat ominaisuudet, jotka ovat nähtävissä kappaleiden toisen tason otsikoissa.

3.3 Qt

Qt julkaistiin yleisön käytettäväksi vuonna 1995. Sen kehitystyö oli kuitenkin jo alkanut vuonna 1990, kun norjalaiset Haavard Nord ja Eirik Chambe-Eng olivat yliopistolleen tekemässään työssä havainneet tarvitsevansa oliomallista ikkunointikirjastoa. Ikkunointikirjaston toteutus aloitettiin Haavardin toimesta vuonna 1991, Eirikin auttaessa suunnittelussa. Seuraavana vuonna Eirik keksi Qt:n kannalta oleelliseen signaali-slotti menetelmän. Qt:n ensimmäinen julkaisu tapahtui, kun Qt laitettiin ladattavaksi sunsite.unc.eduun ja siitä tehtiin ilmoitus Linuxin postituslistalle. Qt:ssa on alusta alkaen ollut käytössä samanlainen kaksoislisenssi. Kaupallisille sovelluksille lisenssi on maksullinen ja avoimelle lähdekoodille ilmainen. Vuonna 2008 Nokian ostettua Trolltechin lisenssiehtoihin lisättiin vielä mahdollisuus valita lisenssointitavaksi LGPL. [9]

Qt:n kehityksen kannalta yksi ratkaisevammista askeleista tapahtui vuonna 1996, kun Matthias Ettricht päätti käyttää Qt:ta hyväksi toteuttaessaan Linuxiin uutta KDE-ikkunointiympäristöä. KDE:sta tulikin hyvin suosittu ikkunointiympäristö Linux maailmassa ja niinpä Qt saavutti käytännön standardin aseman Linux maailmassa. Matthias liittyi Qt:ta kehittävän norjalaisen Trolltechin kehittäjäkaartiin vuonna 1998. Qt:n versio 2.0 julkaistiin vuonna 99 ja siinä tapahtui erittäin suuria kehitysaskelia Qt:n arkkitehtuurin osalta. Vuonna 2000 Trolltech julkaisi Qt/Embedded kirjaston, jolla voidaan käyttää Qt:a sulautetuissa Linux järjestelmissä. Myöhemmin samana vuonna Qt julkaisi Qtopian, joka on erityisesti kädessä pitäviin laitteisiin suunniteltu Qt:n versio. Qt 3.0 julkaistiin vuonna 2001 se oli ensimmäinen Qt:n versio, jota kyettiin ajamaan Windowsilla, Linuxilla, Unixilla, Mac OS X:llä ja vielä sulautetulla Linuxilla. Blanchette *et al*. mainitseekin [9] Qt:ta kehittävän Trolltechin tuplaneen myyntinsä joka vuosi Qt:n perustamisesta lähtien. Hän myös kirjoittaa Qt:lla olevan tuhansia asiakkaita ympäri maailmaa, sekä että kymmeniä tuhansia avoimen lähdekoodin kehittäjiä osallistuu sen kehitykseen. Qt:n uusin versio eli versio 4 julkistettiin kesällä 2005. Uusin versio tukee edellä mainittujen lisäksi Solarista ja HP-UX:ää[9]. Tässä työssä

keskitytään tällä hetkellä yleisimmin käytössä oleviin Qt:n versioihin kolme ja neljä. Tämän hetken tunnetuimpia Qt:ta hyväkseen käyttäviä ohjelmistoja on esimerkiksi Skype. Blanchette *et al*. mainitsee myöskin että, tunnettuja Qt:ta hyödyntäviä yrityksiä ovat muun muassa Adobe, Boeing, IBM, Motorola ja NASA.[9]

3.4 GTK+

GTK+:n historia alkoi GIMP-kuvankäsittelyohjelmasta vuonna 1995: Peter Mattis ja Spencer Kimball tarvitsivat uutta avoimen lähdekoodin kuvankäsittelyohjelmaa kehittäessään korviketta aiemmin käytössä olleelle kaupalliselle Motif-ikkunointikirjastolle, joten he kehittivät oman avoimeen lähdekoodiin pohjaavan ilmaisen ikkunointikirjastonsa. Tässä GTK:n ensimmäisessä versiossa ei ollut mahdollista periä komponentteja ja signalointia ei vielä ollut olemassa. Kun GTK:n kehittäjät lisäsivät kirjastoon komponenttien perinnän ja nykyisen kaltaisen viestijärjestelmän, he päättivät tätä aikaansaannosta kunnioittaakseen lisätä +:n kirjaston nimen perään.[10] Samalla GTK:n kehittäjät valitsivat kirjastonsa lisenssointitavaksi LGPL:n.[10]

Marraskuussa 2002 julkaistiin GTK:n versio 2. Merkittävimmät muutokset versioon 2 olivat paranneltu tekstinpiirto käyttäen hyväksi uutta Pango-kirjastoa, täysi tuki UTF-8 merkistöstandardille, sekä uusi helppopääsyisyys-kirjasto Accessibility Toolkit. GTK:n versioon 2 siirryttäessä menetettiin kirjaston yhteensopivuus ykkösversion kanssa ja kirjastosta tuli paljon edellistä versiota raskaampi. Tästä johtuen osa sulautettujen sovellusten ohjelmoijista käyttää edelleen mieluummin ykkösversiota. Version 2 suorituskykyä kuitenkin merkittävästi parannettiin versioissa 2.2-2.8. Lisäksi näiden versioiden myötä parannettiin GTK:n tarjoamaa komponenttivalikoimaa, Tällä hetkellä uusin vakaa versio on 2.10. Tässä diplomityössä käsitellään juuri tätä GTK+:n versiota. [10]

3.5 wxWidgets

wxWidgets sai alkunsa Julian Smartin yliopistoprojektista vuonna 1992. Hän halusi että hänen tekemänsä diagrammi-työkalu toimisi sekä Sunin tietokoneissa että tavallisissa PC-tietokoneissa. Tähän tarpeeseen hän sitten kehitti oman monialustaisen ikkunointikirjaston nojaten Microsoftin MFC:n sekä Xview-kirjastoon. Xview muuttui myöhemmin Motif kirjastoksi ja MFC muutettiin puhtaaksi Win32-API:n käytöksi Borlandin tuotteiden käyttäjien pyynnöstä. wxWidgets saavutti vuosien kuluessa uskollisen käyttäjäkunnan ja siitä tehtiin useita käännöksiä eri alustoille. Markus Holzem nousi tärkeäksi kehittäjäksi kääntäessään wxWidgetsin Unix-maailmassa tärkeälle X11-palvelimen Xt-rajapinnalle. Vuonna 1997 Markus Holzem suunnitteli uuden wxWidgets 2 API:n, ja Wolfram Gloger ehdotti että tämä API tulisi kääntää tulevalle GNOME GTK+-ikkunointikirjastolle. Vuonna 1998 Robert Roebling suoritti tämän wxGTK:si nimetyn käännöksen ja siitä tuli tärkein käännös Unix/Linux alustalle. Ensimmäinen Stefan Csomorin Mac OS käännös aloitettiin. Vuonna 2002 Julian Smart ja Robert Roedling lisäsivät erittäin kevyen wxX11 käännöksen ja vuonna 2004 valmistui uusi huomattavasti paranneltu MacOSX käännös nimeltään wxMac. Tätä kirjoittaessa wxWidgetsin uusin vakaa versio on versio 2.8. WxWidgets käännökset löytyvät kaikille windowseille (wxMSW), Unixeille/Linuxeille (wxGTK, wxX11) ja Macintosheille (wxMac). [11]

WxWidgets on lisenssoitu wxWidgets lisenssillä joka on LGPL:n mukainen lisenssi. Kirjaston käyttöoikeudet ovat samat kuin edellämainitulla GTK+:lla. WxWidgets lisenssi poikkeaa kuitenkin yhdellä merkittävällä tavalla standardista LGPL-lisenssistä: wxWidgets-lisenssin mukaan kirjaston lähdekoodia muuttavat työt voidaan julkaista käyttäjän omien ehtojen mukaan niin kauan kuin ne julkaistaan ainoastaan binäärisenä. [12]

4. Qt ARKKITEHTUURI

Luvussa esitellään Qt:n ominaisuuksia sekä käsitellään tyypilliseen Qt toteutukseen liittyviä komponentteja. Kappale määrittelee tässä diplomityössä käsiteltävät Qt:n osat.

4.1 Luokka- ja modulirakenne

Taulukko 1: Qt:n kirjastot[7,8]

Kirjaston nimi	***Sisältö***
QtCore	Muiden moduulien käyttämät ei-graafiset luokat
QtGUI	Graafiset käyttöliittymäluokat
QtNetwork	Tietoverkko-ohjelmointiin tarvittavat luokat
QtSql	Tietokanta integraatioon tarvittavat luokat
QtSvg	SVG-tiedostojen näyttämiseen tarvittavat luokat
QtXml	XML:n käsittelyyn tarvittavat luokat
QtDesigner	Qt Designer sovelluksen laajentamiseen tarvittavat luokat
QtUiTools	Qt Designer sovelluksella tuotettujen tiedostojen käsittelyyn tarvittavat luokat
QtAssistant	Ohjeiden luomiseen tarvittavat luokat
Qt3Support	Qt3 yhteensopivuusluokat
QtTest	Työkaluluokat yksikkötestaukseen
QAxContainer	Kontrollit Active X kontrollien hallintaan (vain kaupallinen Windows versio)
QaxServer	Active X palvelun kirjoittamiseen tarkoitetut luokat (vain kaupallinen Windows versio)
QtDBus	Luokat prosessien väliseen kommunikointiin käyttäen hyväksi D-BUS palvelua (vain UNIX-pohjaisissa järjestelmissä)
QtMultimedia	Luokat multimedian tuottamiseen
QtOpenGL	Luokat OpenGL grafiikkakiihdytykseen
QtWebkit	Luokat webkit selainmoottoriin käyttämiseen

Taulukossa 1 on esitelty Qt:n kaikki moduulit. Jokainen moduuli vastaa yhtä Qt:n jaettua kirjastoa Qt:ta ajonaikaisesti linkitettäessä. Kukin moduuli sisältää kuvausta vastaavat luokat ja sovellusta kirjoitettaessa on mahdollista sisällyttää koko moduulin

sisältö tai yksittäinen luokka kirjoitettavaan uuteen luokkaan. Qt:n luokat jakautuvat käyttötarkoituksensa mukaan moduuleihin. Suurin osa Qt:n luokista perii yhteisen QObject – kantaluokan, joten komponentteja on helppo käsitellä kantaluokan osoittimen ja virtuaalisten funktiojäsenten avustuksella.[14]

4.2 Yhteensopivat käännösympäristöt ja kääntäminen

Qt4 toimii Microsoftin Visual Studiossa, sekä Eclipsen CDT lisäosassa, Monkey Studiossa, sekä Eric ja KDevelop kehitysympäristössä. Lisäksi kaikki avoimen lähdekoodin GCC-kääntäjään pohjautuvat käännösympäristöt ja menetelmät toimivat Qt:n kanssa. Vanhempi Qt3 toimii edellämainittujen lisäksi myös Borlandin kehitysympäristössä. Muille kuin Visual Studiolle tuki Qt:lle on ilmainen.

Qt-projektin luomiseksi tulee asentaa käytettyyn käyttöjärjestelmään sopiva versio Qt:sta. Tämän lisäksi voidaan asettaa mahdollisesti kehitysympäristökohtainen lisäke, joka lisää kehitysympäristöön Qt:n vaatimat ominaisuudet. Myös itse Qt:n sijainnin kertova QTDIR-ympäristömuuttuja tulee asettaa osoittamaan Qt:n asennushakemistoon. Mikäli kehitysympäristö ei tee tätä automaattisesti myös luotavassa Qt projektissa tarvittavat Qt:n kirjastot tulee asettaa kehitysympäristöön. Jos Qt sovellus halutaan siirtää toiseen käyttöjärjestelmään tulee projekti kääntää jokaiselle käyttöjärjestelmälle erikseen.

4.3 Ei-näkyvät komponentit

Qt sisältää STL:n kanssa yhteensopivat, mutta omaan ohjelmointikehykseen optimoidut tietosäiliönsä. Qt:n tietosäiliöt tarjoavat täsmälleen saman rajapinnan kuin STL:n vastaavat tietosäiliöt ja ne voidaan tarvittaessa muuntaa STL-säiliöiksi tai STL-säiliöstä. Niillä on kuitenkin eräitä etuja verrattuna STL:n vastaaviin säiliöihin: ne toimivat myös sellaisilla kääntäjillä, jotka eivät toimi STL:n kanssa. Niitä voidaan myös käyttää kuten Javan vastaavia säiliöitä ja ne tukevat Qt:n omia sisäisiä optimointeja. Qt:n versiot STL-

säiliöistä on helppo tunnistaa, STL-säiliön nimen eteen on lisätty Qt:n Q ja etukirjain on muutettu isoksi. Esimerkiksi STL:n vector<T> on Qt:ssa QVector<T> . [13]

Qt:ssa on optimoitu tietosäiliöitä kahdella merkittävällä tavalla. Qt:n tietosäiliöitä ei ole toteutettu mallien(eng. template) avustuksella, kuten STL:ssä. Mallien ongelmana on että ne kasvattavat suoritettavan koodin määrää merkittävästi, koska ne lisäävät jokaiselle erilaistetulle tyypille täsmälleen saman lähdekoodin. Qt:ssa tämä on sen sijaan toteutettu niin että jokainen säiliö lisää vain hyvin vähän tälle säiliölle yhteistä lähdekoodia ja loppu ohjelmakoodista on tavallisessa yksityisessä luokassa. Toinen Qt:n käyttämä optimointimenetelmä on implisiittinen jakaminen. Implisiittisessä jakamisessa tietosäiliönä toimivan luokan sisältämää tietoa ei kopioida tietokoneen muistissa ennenkuin tietoa muutetaan säiliön sisällä. Näin ollen raskasta olion kopioprosessia ei tarvitse tarpeettomasti suorittaa ja muistinkulutus pysyy kohtuullisena.[13]

Tietosäiliöluokkien lisäksi Qt:ssa on sisäänrakennettuna monialustaiset luokat tiedostojärjestelmän käsittelyyn, tietoverkon käsittelyyn, XML-tiedostojen hallintaan, säikeiden luontiin ja hallintaan, sekä useiden eri tietokantojen hallintaan. Näistä luokista löytyy enemmän tietoa esimerkiksi Qt:n whitepaperista.[13]

4.4 Käyttöliittymäkomponentit ja niiden hallinta

Käyttöliittymäkomponenteilla tarkoitetaan visuaalisia elementtejä, joita yhdistelemällä voidaan luoda käyttöliittymiä. Esimerkiksi painikkeet, menut, viestilaatikot ja sovellusikkunat. Whitepaperissa[13] on kerrottu Qt:n erityisominaisuudesta, joka on varsin mielenkiintoinen osa Qt:ta. Qt:n jokainen käyttöliittymäkomponentti kykenee nimittäin toimimaan niin kontrollina kuin kontrollivarastona. Lisäksi Qt:ssa on täysin mahdollista luoda kokonaan uusia komponentteja tyhjästä, joko perimällä kaikille Qt:n komponenteille yhteisestä QWidget-kantaluokasta tai laajentamalla jo olemassa olevista komponenteista.[9]

Qt:n kaikilla käyttöliittymäkomponenteilla on siis sama ylimmäinen kantaluokka

nimeltä QWidget. Tämä mahdollistaa paitsi kaikkien komponenttien yhteisen hallinnan yhteisillä osoittimilla, myös komponenttien tehokkaan periytymisen toisistaan. Mikä tahansa Qt:n käyttöliittymäkomponentti voi sisältää määrättömän määrän lapsikomponentteja, jotka piirretään äitikomponentin alueelle. Qt:n komponenteilla ei ole mitään rajoituksia liittyen niiden hallintasuhteeseen, mikä tahansa komponentti voi olla minkä tahansa komponentin lapsikomponentti ja komponentti, jolla ei ole äitikomponenttia on automaattisesti ikkuna. Yksittäinen komponentti seuraa aina äitikomponenttinsa tilaa: Esimerkiksi jos äitikomponentti poistetaan kaikki sen lapsikomponentit poistetaan.[9]

Piirrettävien komponenttien tulisi sijoittua järkevällä tavalla niitä piirrettäessä tai niiden piirtoalueen koon muuttuessa. Tätä varten Qt:ssa on sisäänrakennettuna niin sanottuja pohjapiirroselementtejä, joiden avulla on mahdollista säädellä komponenttien sijaintia suhteessa toisiinsa ja piirtoalustaansa. Komponenttien välille voidaan esimerkiksi määritellä tietty tyhjä tila ja sen jälkeen voidaan määritellä miten alueen koon muuttuessa tyhjä tila jaetaan eri komponenttien kesken ja miten komponenttien itsensä koko muuttuu. Tämä komponenttien koon hallinta voidaan sitten toteuttaa niin pysty- kuin vaakasuunnassakin. [14]

4.5 Asynkronisten viestien toteutus

Signaalit ja slotit ovat eräs Qt:n keskeisimmistä käsitteistä. Pitkälti niiden ansiosta Qt:a voidaan pitää oliomallia noudattavana ikkunointikirjastona. Ne mahdollistavat monia asioita, niiden ansiosta Qt:a käyttävän kehittäjän ei tarvitse monen muun ikkunointikirjaston tavoin huolehtia Callback(takaisinkutsu) funktiokutsuista. Niiden ansiosta on mahdollista sitoa kaksi toisistaan tietämätöntä oliota keskenään.

Slotit ovat ovat käytännössä täysin samanlaisia kuin normaalit C++:n metoditkin. Niillä on metodia vastaava toteutus toteutustiedostossa(.cpp). Niillä voi olla erilaisia näkyvyyksiä, ne voivat olla virtuaalisia ja ne voidaan ylikirjoittaa. Niitä voidaan myöskin kutsua kuten normaaleja C++:n metodeja. Niillä on kuitenkin eräs

mielenkiintoinen ominaisuus verrattuna tavallisiin metodeihin: niille voidaan määritellä signaali tai Qt:n termein niihin voidaan yhdistää signaali. Mikä sitten on signaali? Signaalit ovat saman näköisiä kuin normaalit C++:n metodit. Niillä ei kuitenkaan ole toteutusta toteutustiedostossa vaan ne ovat pelkkiä funktion otsikoita C++:n otsikkotiedostoissa.

```
class esimerkkiLuokka : public kantaluokka
{
        Q_OBJECT
        public:
                void vakiofunktio();
        public slots:
                void slottiEsimerkki(QString str);
        signals:
                void signaaliesimerkki(QString str);
}
```

Lähdekoodiesimerkki 1: Qt:n mukainen C++ otsikko tiedosto

Lähdekoodiesimerkissä 1 on esitetty Qt:n mukainen C++ header tiedosto. Luokan yläosan Q_OBJECT makro kertoo Qt:n esikääntäjälle, että luokka on Qt-luokka eikä tavallinen C++ otsikkotiedosto. Tämän tiedoston perusteella Qt:n MOC esikääntäjä lisää Qt:n varaamia sanoja vastaavat C++ koodipätkät oikeille paikoilleen. Esimerkiksi lähdekoodiesimerkin public slots ja signals korvataan C++ standardin mukaisella lähdekoodilla. Itseasiassa MOC lisää C++ metodien normaaliin toteutukseen signaalien toteutuksen ja vielä introspection eli metodit metaObject(), tr() ja muutamia muita. Introspektiolla tarkoitetaan että vastoin normaalia C++ objektia, Qt:n mukainen objekti on ”tietoinen” omasta tyypistään. Koska MOC:n lisäämä koodi on validia C++:aa Qt-koodista C++:aan käännetty koodi toimii minkä tahansa C++:n ISO standardia seuraavan kääntäjän kanssa.

Yleisin intospektion käyttö Qt:ssa on signaali ja slotti järjetelmää käytettäessä. Aiemmin esitellyillä signaali ja slotti tyyppisillä metodeilla on erityismerkitys.

connect(this, SIGNAL(signaaliEsimerkki(QString)), this,
SLOT(slottiEsimerkki(QString)));
Lähdekoodiesimerkki 2: Signaalin liittäminen slottiin

Signaalit voidaan liittää edellä esitetyllä tavalla slotteihin. Koodiesimerkissä on liitetty aiemmin esitetyn luokan sisällä oleva signaaliEsimerkki signaali ,joka kantaa "kuormanaan" Stringiä slottiin slottiEsimerkki, joka vastaanottaa signaalin lisäksi yhden stringin. Vaikka koodiesimerkissä on liitetty yhteen saman olion sisällä olevat signaali ja slotti, on täysin mahdollista liittää myöskin täysi toisiinsa liittymätöntä oliota kunhan oliolla joka liittää on olemassa osoitin toiseen olioon. Vaikka esimerkissä on signaalille annettu kuormana vain yksi olio, on täysin mahdollista antaa signaaleille määräämätön määrä kuormaa tai ei kuormaa ollenkaan. Tulee kuitenkin huolehtia että "kuorma" on samassa järjestyksessä signaalissa kuin slotissakin. Mikäli slotilla on vähemmän parametreja kuin signaalilla, ylimääräiset parametrit yksinkertaisesti hylätään, varoituksen kera.

emit signaaliEsimerkki(QString("hello world"));
Lähdekoodiesimerkki 3: Signaalin emittoiminen

Kun signaali sitten halutaan lähettää matkaan, se emittoidaan. Mikäli Signaali on kytketty slottiin, slotti tulee kutsutuksi ja sen koodi suoritetaan heti emittoimisen jälkeen. Mikäli signaali on kytketty useampaan slottiin kaikkia liitettyjä slotteja kutsutaan satunnaisessa järjestyksessä. Signaali voidaan liittää myöskin toiseen signaalin jolloin signaalin emittointi emittoi myös toisen signaalin. Joskin tämä ei ole kovin suotavaa, koska tästä saattaa helposti tulla väärinkäsityksiä.

disconnect(this, SIGNAL(signaaliEsimerkki(QString)), this,
SLOT(slottiEsimerkki(QString)));
Lähdekoodiesimerkki 4: Signaalin irrottaminen slotista

Kuten koodiesimerkissä 4 signaali voidaan myöskin ajonaikaisesti irrottaa slotistaan. Normaalisti tätä ei juurikaan tarvitse tehdä, koska signaali irrotetaan automaattisesti

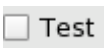
slotistaan kun jommankumman sisältämä olio tuhotaan.

Käyttäjän itsensä ja valmiiden komponenttien lisäksi tärkeä asynkronisten viestien lähde ovat niin sanotut tapahtumat(eng. events). Tapahtumat ovat käyttöjärjestelmän sovellukselle lähettämiä viestejä. Viesti saattaa esimerkiksi sisältää tiedon siitä minkä komponentin päällä hiiri sijaitsee tällä hetkellä. Qt:ssa tapahtumien käsittely on toteutettu jokaisessa Qt:n mukaisessa oliossa olevan QObject kantaluokan avulla. Jokainen QObjectin perivä luokka pitää sisällään tapahtumia käsittelevän event()-funktion. Tälle funktiolle lähetetään QEvent tyyppinen olio, mikäli viesti kyseiseen QObjectin perivään luokkaan liittyy. Ikkunointikirjaston käyttäjä voi helposti liittää toimintoja haluamansa käyttöliittymä-komponenttin tapahtumiin yksinkertaisesti ylikirjoittamalla event-funktion ja käsittelemällä sitten funktion vastaanottamaa QEvent luokkaa. Mikäli käyttäjä ei ole kiinnostunut tai ei halua huomioitavan kaikkia olioon liittyviä tapahtumia hän voi asentaa luokkaan niin sanotun tapahtumafiltterin, jossa hän sitten voi jättää muut tapahtumat kuin mistä on kiinnostunut huomioitta.

4.6. Esimerkkejä joidenkin peruskomponenttien käytöstä

Jokainen Qt:ta käyttävä tulee käytännössä käyttäneeksi Qt.:n mukana toimitettavia sisäänrakennettuja komponentteja. Ne tulevatkin hyvin nopeasti tutuksi Qt:lla ohjelmoitaessa.

Taulukko 2: valintalaatikko eri ympäristöissä

	Windows	Plastik(KDE)	Aqua(OS X)
QCheckBox (const QString & text, QWidget * parent = 0)			

Taulukossa 2 esiteltävä valintalaatikko on yksinkertainen Qt:n peruskomponentti. Sille annetaan joko pelkästään sen omistava QWidgetin perivä luokka tai sitten omistava

luokka ja valintalaatikon nimi, joka näkyy kuvassa valintalaatikon vasemmalla puolella. Laatikon tilaa voidaan ohjelmallisesti muuttaa kutsumalla sen setCheckState() funktiota. Aina kun käyttäjä valitsee valintalaatikon, valintalaatikko lähettää signaalin stateChanged.[14]

Taulukko 3: Rivin mittainen tekstilaatikko eri ympäristöissä

	Windows	Plastik(KDE)	Aqua(OS X)
QLineEdit (const QString & contents, QWidget * parent = 0)	Test	Test	Test

Taulukossa 3 esiteltävä tekstikenttä on eräs yleisimmistä komponenteista Qt-sovelluksessa. Tekstikenttä luodaan samoin kuin valintalaatikko edellä, sillä erotuksella että parametrinä annetaan tekstikentän sisältö. Kaikkia toimia joita käyttäjä voi suorittaa tekstikentälle voidaan ohjelmallisesti simuloida tekstikentän avulla. Tekstikentän tila voidaan myöskin selvittää kutsumalla tekstikentän funktioita. Jokainen tekstikentän kursorin muutos saadaan talteen cursorPositionChanged(int old, int new) signaalista. Signaalit textChanged(QString& text), textEdited(QString& text) lähettävät tiedon tekstin muuttumisesta. Lisäksi tekstin editoinnin loppumiselle, laatikon valinnalle ja returnin painallukselle on omat signaalinsa. Tekstikentän setEchoMode(EchoMode) funktio edustaa erästä Qt:lle hyvin tyypillistä toimintatapaa. EchoMode on QLineEdit luokassa oleva määritelty enum, joka sisätää laatikon eri tekstinäyttötavat määriteltyinä vakiona. Esimerkiksi QLineEdit::Password tarkoittaa salasanan syöttötavan mukaisesti kirjoitetun tekstin toistamista. QLineEdit luokka sisältää echoMode nimisen ominaisuuden, jolla voi olla eri arvoja riippuen tekstin tulostustavasta. Qt:n yleisen säännön mukaan tätä ominaisuutta varten on tehty niin sanottu setteri-funktio(tässä tapauksessa setEchoMode(EchoMode), jolla ominaisuuden arvoa voidaan muuttaa halutuksi.[14]

Qt:n pudotusvalikko luonti noudattaa Qt:n standardia kaavaa. Jokaisella komponentilla on suositeltavaa antaa vanhempi, kun luokkaa luodaan. Tämä ei kuitenkaan ole pakollista. Jos luotavalle komponentille ei anneta vanhempaa, komponentin luojan on

itse huolehdittava komponentin poistamisesta. Jos taas komponentille on määritelty vanhempi Qt osaa automaattisesti poistaa komponentin kun sovellus suljetaan tai jokin kyseisen komponentin vanhemmista poistetaan. Tämä pätee myös kaikkiin muihin Qt:n komponentteihin, ei pelkästään vain sisäänrakennettuihin komponentteihin.

Pudotusvalikon varmasti yleisin käytetty metodi on insertItem() jolle annetaan vain numero jossa item sijaitsee pudotusvalikossa ja näytettävä tekstikenttä. Pudotusvalikko on hyvä esimerkki Qt:n tarjoamasta tehokkaasta signaalien käytöstä. Liittämällä oma sovelluskoodinsa komponentin tarjoamiin activated(int) signaaleihin on esimerkiksi todella helppo huomata mitä käyttäjä on juuri tällä hetkellä valitsemassa. Vastavuoroisesti on helppo liittää jokin oman sovelluskoodinsa toiminnallisuus Qt:n clear() slottiin jolloin voidaan koko pudotusvalikko helposti ja asynkronisesti tyhjentää. Qt tarjoaa erittäin suuren määrän muitakin komponentteja, joista löytyy listaus Qt:n kotisivulta. [14]

4.7. Omat muokatut komponentit

Blanchette *et al*. mainitsee että käyttöliittymiä suunnitellessa usein tulee tilanteita, jolloin ei ole yksinkertaisesti mahdollista pelkästään muokata olemassa olevia ikkunointikirjaston tarjoamia vaihtoehtoja. Qt tarjoaa tällaisiin tilanteisiin kaksi vaihtoehtoa. Ensimmäinen vaihtoehto on nopein toteutettava eli perintä ikkunointikirjaston olemassa olevista komponenteista. Koska koko Qt on itseasiassa suuri perintäketjuista muodostunut puu, on varsin helppoa ottaa pohjaksi jokin Qt:n komponenttia ja muokata sen toimintaa haluamaansa suuntaan. Voidaan esimerkiksi ottaa edellä mainittu QLineEdit ja ylikirjoittaa vaikkapa sen cursorForward() funktiojäsen jolloin voidaan luoda tekstilaatikko, jonka kursori ei liikukaan normaalilla tavalla eteenpäin.

Joskus saattaa tulla eteen tilanteita jolloin tarvitaan komponentti, jonka kaltaista ei ole ollenkaan tarjolla ikkunointikirjaston puolesta. Tällöin Qt tarjoaa vaihtoehdon periä QWidget komponentti, josta kaikki sisäänrakennetutkin Qt:n komponentit on peritty.

Jos halutaan käyttää 3D grafiikkaa tai muuten vain tehokkaampia kaksiulotteisiakin grafiikkarutiineja on helppoa käyttää QGLWidget luokkaa normaalin QWidget komponentin sijasta. QGLWidget luokka perii QWidget luokan joten hiiren ja näppäimistön hallinta toimii vastaavalla tavalla[9]. Qt tarjoaa myöskin mahdollisuuden periä jonkin sen sisältämistä tyyleistä. Peritty tyyli voidaan sitten muuttaa halutuksi. Tämä mahdollistaa komponenttien ulkonäön määrittelemisen täysin erilaiseksi kuin mitä käyttöjärjestelmä normaalisti tarjoaisi. Tyylin ja erillisen tyylitiedoston avulla voidaan hallita kaikkien Qt komponenttien ulkonäköä ja visuaalista vastetta erilaisiin toimintoihin.[14]

4.8 Suunnittelutyökalu ja kielenkäännös: Qt Designer ja Qt Linguist

Qt Designer on Qt:n käyttöliittymien ulkoiseen suunnitteluun tarkoitettu työväline. Sillä käsitellään rakennettavan sovelluksen käyttöliittymää sellaisena kuin se sovelluksen käyttäjälle näkyy. Toisin sanoen sillä "piirretään" sovelluksen käyttöliittymä, käyttäen Qt:n sisäänrakennettuja komponentteja tai itse tehtyjä komponentteja. Sillä voidaan myöskin asettaa käyttöliittymäkomponenttien ominaisuuksia. Tyypillinen käyttötapaus voisi esimerkiksi olla kehittäjä piirtämässä jotakin suunnittelemansa sovelluksen osaa. Ensin hän sijoittaa tarvitsemansa komponentit piirtoalustalle haluamilleen paikoille ja liittää komponentit niitä mahdollisesti hallinnoiviin pohjapiirustus-luokkiin. Tämän jälkeen hän antaa kullekin komponentille nimen, jolla hän sitten kykenee käyttämään luomaansa komponenttia omassa lähdekoodissaan. Sitten hän asettaa komponentin ominaisuudet kuten komponentin näkyvyyden, toimintatavan ja alkuarvot kohdalleen.

Miten käyttöliittymätiedostot voidaan sitten liittää sovelluksen muuhun koodiin? Qt designer tuottaa .ui päätteisiä käyttöliittymätiedostoja, joissa tavallisesti on esimerkiksi yksi sovelluksen ikkuna. Riippuen siitä millaista kehitysympäristöä käytetään, kehittäjän tulee liittää .ui tiedostot projektiinsa niin että Qt:n qmake työkalu ymmärtää kääntää ne C++ kieliseksi lähdekoodiksi käyttäen Qt:n mukana tulevaa UIC kääntäjää. UIC-kääntäjä sitten tuottaa projektin käännöksen yhteydessä uuden luokan, joka voidaan liittää kehittäjän projektiin objektina. Tämän jälkeen kehittäjä kykenee

käyttämään hyväkseen luomaansa objektia liittämällä sen muihin suunnittelemansa käyttöliittymän komponentteihin ja näin ollen näyttämään sen luomansa sovelluksen ikkunassa tai ikkunoissa.

Qt käyttää sisäisesti, käyttöliittymässä ja kaikissa merkkijonomuuttujissaan Unicodea. Käytännössä tämä tarkoittaa sitä että Qt:lla suunniteltu sovellus toimii kaikilla maailman kielillä ja merkistöillä. Koska varsin usein sama sovellus käännetään usealle kielelle Qt tarjoaa tähän tarkoitukseen Qt Linguist käännösympäristön. Ohjelmoijan kannalta olennaisin asia käännöksen kannalta on käyttää jokaisessa Qt-objektissa mukana tulevaa tr() funktiota. Esimerkiksi kuten koodiesimerkissä 5.

button->setText(tr("paino Napin teksti","kääntäjälle näkyvä kommentti");
Lähdekoodiesimerkki 10: tr() funktion käyttöesimerkki

Kun ohjelmoija merkitsee käyttöliittymälle näkyvät merkkijonot näin, merkkijonoista saadaan QTranslator objekteja, jotka nojaavat levylle talletettuihin kielikohtaisiin .qm tiedostoihin. Kielenkääntäjät käyttävät sitten Qt Linguist sovellusta hyväkseen luodakseen .ts tiedostoja, jotka pitävät sisällään kunkin merkkijonon käännöksen useilla eri kielillä. Lisäksi kääntäjä voi Qt Linguistin avulla valita myös näppäimistöoikotiet kutakin kieltä vastaavaksi.[13]

4.9 Piirtorutiinien toteutus eri alustoilla

Qt:n arkkitehtuurin perustana on Qt:n mahdollisimman hyvä soveltuvuus käytettäväksi eri käyttöjärjestelmissä. Qt pyrkii samaan aikaan olemaan mahdollisimman tehokas ja mahdollisimman monialustainen. Toisin kuin useat muut monialustaiset ikkunointikirjastot Qt:a ei ole rakennettu minkään valmiin olemassa olevan ikkunointikirjaston "päälle". Qt käyttää hyväkseen mahdollisimman matalan tason piirtorajapintoja kustakin käyttöjärjestelmästä. Voidaankin sanoa Qt:n piirtävän, jokaisen piirtämänsä komponentin "viiva viivalta" kullakin alustalla. Vaikka Qt siis piirtää "itse" jokaisen käyttämänsä käyttöliittymä-komponentin, se ei kuitenkaan muuta

sillä ohjelmoitavan sovelluksen ulkonäköä omanlaisekseen, vaan emuloi alla toimivan käyttöjärjestelmän toiminnallisuutta ja ulkonäköä. Qt:lla toteutettua sovellusta onkin mahdoton erottaa käyttöjärjestelmän vakiokomponenteilla toteutetusta sovelluksesta. Qt:n komponenttien ulkonäkö ei ole kuitenkaan mitenkään sidottu käyttöjärjestelmän ulkonäköön, jossa Qt toimii. Jokaisella Qt:n komponentilla on virtuaalifunktiot, joita laajentamalla ohjelmoija voi muuttaa komponentin ulkoasun ja tarvittaessa jopa toiminnallisuuden haluamakseen. Lisäksi Qt tarjoaa QStyle luokan, joka mahdollistaa kaikkien sovelluksen komponenttien ulkonäön täysin tälle sovellukselle ominaiseksi.

Qt:n Linux- ja Unix-toteutus pohjautuu Xlib-kirjastolle, jota Qt käyttää suoraan hyväkseen kommunikoidakseen X-palvelimen kanssa suoraan. Koska X-windows pohjaisilla järjestelmillä saattaa olla käytössään useita ulkonäkö ja käyttäytymistapa malleja Qt sopeutuu näistä yleisimpiin eli, kuten Qt:n whitepaper kirjoittaa[13], sillä on käytössä paikallinen käyttötuntuma Motif-, CDE-, GNOME- ja KDE-ympäristöissä. Qt:n käyttää Windows piirrossa suoraan hyväkseen Windowsin API:a sekä GDI:tä tapahtumiin ja piirtoprimitiiveihin. Lisäksi Qt:n mainitaan, luonnollisestikin, sopeutuvan kunkin Windows-version versiokohtaiseen ulkonäköön ja toiminnallisuuteen. Qt:n Mac OS X-tuki on toteutettu käyttäen hyväkseen sopivaa yhdistelmää OS X:n Cocoa API:a ja Carbon API:a. Qt-sovellus toimii yhtä lailla intel- kuin powerPC Macintosh-tietokoneessa.[13]

5. GTK+ ARKKITEHTUURI

Kappaleessa esitellään GTK+:n ominaisuuksia sekä käsitellään tyypilliseen GTK+ toteutukseen liittyviä komponentteja. Kappaleessa käsitellään GTK+:n käyttöä niin sen oman C-pohjaisen rajapinnan, kuin C++-pohjaisen Gtkmm-rajapinnan kautta. Kappale määrittelee käsiteltävät GTK+:n osat. Käsiteltävät osat ovat GTK+:n vastineita edellä esitellyille Qt-ikkunointikirjaston osille.

5.1 Luokka- ja modulirakenne

Taulukko 4: GTK+ 2:n osat[15]

Kirjaston nimi	***Sisältö***
LibGtk	GTK+ komponenttien toteutukset ja rajapinnat
GObject	GTK+:n käyttämä oliojärjestelmä
GLib	Tarjoaa pääsyn tapahtumasilmukkaan, säikeisiin, ajonaikaisiin tapahtumiin ja GTK+:n oliojärjestelmään(GObject).
Pango	Mahdollistaa tekstin piirron ja fonttien käsittelyn
ATK	Mahdollistaa tavanomaisesta poikkeavien syöttölaitteiden käytön GTK+ 2:n kanssa
GDK	Tarjoaa käyttöjärjestelmästä riippumattoman matalan tason rajapinnan grafiikan piirtämiseen
GDK-pixbuf	Mahdollistaa kuvien käsittelyn

GTK+ jakautuu osiin taulukossa 4 esitellyllä tavalla. Käyttääkseen GTK+:a ohjelmoijan tarvitsee käyttää GLib kirjastoa ja liittää gtk:n otsikkotiedosto sovellukseensa. Näin ollen ohjelmoijan ei tarvitse sisällyttää kuin kaksi otsikkotiedostoa päästäkseen käsiksi koko ikkunointikirjaston toiminnallisuuteen. Mikäli käyttäjä haluaa kuitenkin laajentaa pelkän GTK+:n toiminnallisuutta hänen täytyy laajentaa tai käyttää, jotakin tai joitakin

GTK:n muista osista.[15]

Taulukko 5: Tyypillisen GTK+:n sovelluksen käyttämiä ulkoisia ja monialustaisia kirjastoja[17]

Kirjaston nimi	***Sisältö***
Cairo	Käyttöjärjestelmä- ja laitteistoriippumaton vektorigrafiikka-kirjasto
pkg-config	Työkalu, jonka avulla saadaan käännettävälle GTK+ sovellukselle sopivat käännös- ja linkitysliput
GNU libiconv	kansainvälisten merkistöjen tuki
GNU gettext	kansainvälisten merkistöjen käsittely
libpng	png-kuvien käsittely
libjpeg	jpg-kuvien käsittely
zlib	tiedonpakkaus-kirjasto (libpng ja libjpeg vaativat tämän)
libexpat	XML-tiedostojen käsittely
gtkmm	GTK+:n C++ - rajapinnat
libsigc++	Tarjoaa C++:n mukaisen signaalien käsittelyn GTK++:n
STL(Standard Template Library)	Tarjoaa yleisesti käytettyjä säiliöitä ja algoritmeja
libxml++	C++ kirjasto XML:n käsittelyyn
Libglibmm	C++ rajapinnat glib:n käyttöön
Libgdkmm	C++ rajapinnat gdk:n käyttöön
Libcairomm	C++ rajapinnat cairon käyttöön
Libatkmm	C++ rajapinnat ATK käyttöön
Libpangomm	C++ rajapinnat pangon käyttöön

Kuten Taulukosta 5 voidaan nähdä, tyypillinen GTK+ sovellus tarvitsee useita ulkoisia kirjastoja toimiakseen. Esimerkiksi tämän hetken uusin GTK+:n versio 2.12 ei toimi ollenkaan ilman Cairoa. Koska pelkän GTK+:n tarjoama rajapinta on C-pohjainen,

tarvitaan myöskin tuki C++:n ominaisuuksille kuten polymorfismille, perinnälle ja signaalien liittämiselle olioiden jäsenfunktioihin. Gtkmm ja libsigc++ - projekteissa on toteutettu kaikki tarvittava GTK+:n käyttämiseksi C++:ta. Kummatkin kirjastot ovat lisensoitu LGPL:llä GTK+:n tapaan ja ovat myöskin alustariippumattomia.[15]

Linux / Unix ympäristöön kehitettäessä GTK+ sovelluksen tarvitsemat kirjastot löytyvät usein jo valmiiksi asennettuna järjestelmästä. Tai niihin on helppo asettaa riippuvuus, jolloin käyttöjärjestelmä huolehtii kirjastojen lisäämisestä/kääntämisestä automaattisesti. Kun GTK+ sovellus toimitetaan Windows- tai Mac OSX-alustalle, nämä kirjastot tulee toimittaa sovelluksen mukana. [16]

5.2 Yhteensopivat käännösympäristöt ja kääntäminen

Kaikki avoimen lähdekoodin GCC-kääntäjään pohjautuvat käännösympäristöt toimivat GTK+:n kanssa. GTK+ sovellukset on mahdollista kääntää käyttäen Microsoftin Visual Studiota. GTK+:n osalta käännös onnistuu parhaiten käyttäen Visual Studion versiota 6. [16] Sen sijaan gtkmm:m ja libsigc++:n Windows-käännösohjeessa suositellaan käyttämään Visual Studion versiota 7.1 tai uudempaa[18]. GTK+:lle ei ole tarjolla valmiita lisäkkeitä millekään IDE:lle, mutta GTK+:lle kirjoitetun koodin kirjoitus ja kääntäminen onnistuu kätevästi mikäli kehitysympäristö vain tukee Automakella tai makella luotuja käännöstiedostoja, sekä kehitysympäristön kääntäjä tukee GTK+:a. Lisäksi esimerkiksi Anjuta ja Kdevelop tarjoavat mahdollisuuden luoda automaattisesti koko GTK+ projekti, käyttäjän tarvitsematta juurikaan muuttaa projektiasetuksia käsin.

GTK+ 2-projektin luonti tapahtuu samoin kuin normaalin make- tai automake-projektin luonti. Windowsin tapauksessa voidaan käyttää CygWiniä tai MinGw:tä jotka tarjoavat kyseiset työkalut windowsille tai sitten vain kehitysympäristön omaa projektitiedostoa. Tyypillisessä GTK+-projektissa on ohjelmoijan työn helpottamiseksi käännös- ja linkkausliput luotu automaattisesti käyttäen pkg-config työkalua. Myös Visual Studiolla Windowsissa käännettäessä kannattanee käyttää pkg-config-työkalua, tosin Windowsin

cmd.exen puutteiden takia työkalua ei voi suoraan käyttää nmaken make-tiedostosta, vaan työkalun tuloste pitää kopioida kyseiseen make-tiedostoon. [16]

5.3 Ei-näkyvät komponentit

Taulukko 6: Eräitä GLib:n valmiiksi tarjoamia tietorakenteita

Tietorakenteen tietotyyppi	*Tietorakenteen selitys*
GList	Kahteen suuntaan linkitetty lista
GString	Ajonaikaisesti suureneva ja pienenevä merkkijono
GQark	Assosiaatio merkkijonon ja vakionumeron välillä
GData	Assosiatiivinen taulukko
GNode	N-ulotteinen puu
GRelation	Tietokannan taulua vastaava tietorakenne

GTK+:n itsensä tarjoamat ei näkyvät komponentit on sijoitettu GLib-kirjastoon. Kirjastosta löytyvät kaikki tarpeelliset perustietotyypit alustariippumattomina versiona. Perustietotyypit on toteutettu yksinkertaisimmalla mahdollisimmalla tavalla: ne ovat käytännössä ainoastaan uudelleennimettyjä saman alustan normaaleja perustietotyyppejä. GLib sisältää valmiina tyypillisesti käytettyjä tietorakenteita, yleisimmin käytettyjä tietorakenteita on lueteltu taulukossa 6. GLib sisältää tietotyyppien muunnokset ja muistinhallintaan liittyvät funktiot. Myös tiedostojärjestelmän, säikeiden ja prosessien hallinta on osa GLib:ä. Enemmän tietoa GLib:n tarjoamista funktioista löytyy GLib:n referenssi dokumentaatiosta. [19]

Koska GTK+ kuten GLib ovat C-pohjaisia kirjastoja on C++:lla ja Gtkmm:llä ohjelmoitaessa mahdollista käyttää myös GLib:n tarjoamia tietorakenteita ja algoritmeja hyväksi. Tämä ei kuitenkaan ole järkevää kaikissa tapauksissa, esimerkiksi tietorakenteiden osalta. Jokainen C++ ohjelmoija tietää että osoittimet voivat aiheuttaa hankalia ongelmatilanteita ja niiden käyttöä tulee välttää aina kun mahdollista. Koska

STL-tarjoaa C++:n kanssa yhteensopivat ja jäsenmuuttujiksi liitettävät tietorakenteet on huomattavasti järkevämpää käyttää niitä kuin GLib:n tarjoamia kilpailevia vaihtoehtoja. C++:lla ohjelmoitaessa lieneekin järkevämpää välttää suoraa GLib:n käyttöä, ellei se ole pakollista toteutuksen tai siirrettävyyden takia. Merkkijonoja voidaan kuitenkin pitää poikkeuksena tästä säännöstä. Vaikka STL:n std::string merkkijonot käyttävät UTF-8 merkistökoodausta, ne eivät tue kaikkien kielien UTF-8 merkistökoodausta. Ongelmia on esimerkiksi Japanin ja Kiinan merkistökoodauksen kanssa. GLib:n Glib::ustring merkkijonot kuitenkin seuraavat Unicode Consortiumin määritystä, jonka ansiosta myös STL:n toteutuksesta puuttuvat ja jopa kaatumisia aiheuttavat kielet ovat tuettuina.

5.4 Käyttöliittymäkomponentit ja niiden hallinta

GTK+:ssa kaikki oliot perivät GObjectin ja kaikki käyttöliittymäkomponentit perivät GtkObjectin. Näkyvät käyttöliittymäkomponentit perivät edellämainittujen lisäksi vielä GtkWidget luokan. Käyttöliittymäkomponentteja voidaan hallita kantaluokkiensa avustuksella. Komponenttien toimintoihin pääsee käsiksi käytettävästä rajapinnasta riippuen kahdella eri tavalla. Mikäli niitä käytetään oliomallisesti gtkmm:n kautta, niiden funktiojäseniä kutsutaan suoraan. Mikäli niitä käytetään suoraan GTK+:n rajapintojen kautta kutsutaan suoraan kyseisen toiminnon toteuttavaa proseduraalista funktiota ja annetaan mahdollisesti osoitin käsiteltävään komponenttiin sen oikeanlaiseksi osoittimeksi tarkistavan makron kautta. [20], [21]

GTK+:n komponentit jakautuvat selkeästi kahteen leiriin. Jotkut komponentit pitävät sisällään toisia komponentteja ja toimivat näin säiliöinä. Toiset taas toteuttavat jonkin yksinkertaisen toiminnallisuuden, lähettävät signaaleja ja ovat säiliökomponenttien osana. Tyypillisiä säiliökomponentteja ovat esimerkiksi ikkunat, pohjapiirustukset ja erilaiset paneelit. Säiliökomponentit pitävät tavallisesti sisällään toisia säiliökomponentteja ja säiliökomponentilla on hallintasuhde näyttämiinsä komponentteihin tai lapsiinsa. Osa säiliökomponenteista on näkyviä kuten ikkunat, osa taas ainoastaan hallinnoi muitten komponenttien sijaintia, kuten esimerkiksi

pohjapiirustus-luokat.[21], [20]

Pohjapiirustus luokat voivat olla vaaka- tai pystysuuntaisia ”jonoja” tai mahdollisesti taulukoita. Pohjapiirustus luokkiin voidaan ”pakata” komponentteja, jolloin ne asettuvat niille merkityille paikoille. Kunkin näkyvän komponentin sijaintia pohjapiirustuksessa voidaan säädellä antamalla komponentin sijainti pohjapiirustuksessa, sekä lisäksi annettavalla vakiomuotoisella luetellulla tyypillä, joka kertoo miten komponentti käyttäytyy suhteessa muihin pakattaviin komponentteihin.[21] GTK+ edellyttää jokaiselta näkyvältä komponentilta, pääikkunaa lukuunottamatta, kuulumista pohjapiirustukseen. Tämä mahdollistaa GTKmm-lisäkirjaston toteuttaman automaattisen muistinhallinnan, jossa pohjapiirustukseen komponenttia lisättäessä se voidaan merkitä hallituksi(eng. managed). Hallitun komponentin elinkaari määräytyy sitten pohjapiirustuksen mukaan: mikäli pohjapiirustus poistetaan, poistetaan myös kaikki sen sisältämät komponentit. Koska kaikkien näkyvien komponenttien tulee kuulua pohjapiirustukseen kaikki poistettavan pohjapiirustuksen sisältämät pohjapiirustukset ja niiden sisältämät komponentit jne. poistetaan. [20]

5.5 Asynkronisten viestien toteutus

GTK+ signaalijärjestelmä perustuu GObject-luokkien tarjamille signaaleille ja niiden liittämiselle kirjaston käyttäjän tarjoamiin staattisiin takaisinkutsu-funktioihin (eng.callback). GObject–kirjastossa sijaitsevat oliot pitävät sisällään toimintaansa kuvaavien signaalien lisäksi myös käyttöjärjestelmän objektiin kohdistavat tapahtumat.

```
static void painalluksenKasittely(GTKWidget *widget, gpointer data)
{
        g_print(”OMG signaali saapui \n”);
}

static void ikkunanSulkemisenKasittely(GTKWidget *widget, GdkEvent *tapahtuma
,gpointer data)
```

```
{
        g_print(”OMG tapahtuma saapui \n”);
        return FALSE;
}
..
g_signal_connect (G_OBJECT (window), "delete_event",
        G_CALLBACK(ikkunanSulkemisenKasittely),  NULL);
..

g_signal_connect (G_OBJECT (button), "clicked",
        G_CALLBACK (painalluksenKasittely), NULL);
..
```

Lähdekoodiesimerkki 5: Signaalin ja tapahtuman liittäminen takaisinkutsu-funktioihin

Signaalit ja tapahtumat liitetään tapahtumia käsitteleviin funktioihin koodiesimerkin 5 mukaisesti, esimerkissä olevat makrot G_OBJECT ja G_CALLBACK huolehtivat että liitettävä osoitin on oikeasti olemassa, toteuttaa oikeanlaisen signaalin/funktion ja että se on muunnettavissa kantaluokan osoittimeksi. Merkkijonoparametrina on yhdistettävän signaalin/tapahtuman nimi. Painalluksen signaalia käsittelevässä funktiossa on ensimmäisenä parametrina osoitin komponenttiin, joka on lähettänyt signaalin. Toisena parametrina on osoitin mahdolliseen ”hyötykuormaan”, esimerkissä hyötykuormana on tyhjä NULL arvo. Tyypillisessä GTK+ sovelluksessa hyötykuormaa käytetään melko paljon, koska sen avulla voidaan helposti kuljettaa tietoa signaalinkäsittelijöiden ja muiden ohjelman osien välillä. Hyötykuorma mahdollistaa myös C++ koodin liittämisen staattiseen C-pohjaiseen GTK+ koodiin. Takaisinkutsu-funktio voi nimittäin olla myös C++ olion staattinen funktiojäsen ja hyötykuormana voidaan toimittaa this-osoitin, jolloin staattinen funktiojäsen pystyy kutsumaan olionsa tavallisia funktiojäseniä. Tiettyyn objektiin liittyvät tapahtumat voidaan liittää tapahtumankäsittelijöihin hyvin pitkälti signaaleja vastaavalla tavalla. Tapahtuman käsittelevässä funktiossa on kuitenkin pari merkittävää eroa. Ensinnäkin se saa parametrina osoittimen GDKEvent-tyyppiseen olioon(tietueeseen), josta on saatavilla lisätietoa tapahtumasta. Toiseksi tapahtumien käsittelijät palauttavat boolean arvon,

jonka true arvolla GTK+ ei käsittele tapahtumaa ja false arvolla tapahtuma käsitellään. [21]

Gtkmm ja libsigc++ tarjoavat GTK+:n itsensä tarjoaman C-pohjaisen tapahtumien ja signaalien käsittelyn sijaan aidosti oliomallisen ja ajonaikaisen signaalien käsittelyn. Niiden ansiosta kirjaston käyttäjän ei tarvitse huolehtia GTK+:n vaatimien staattisten takaisinkutsu-funktioiden oliomalliin liittämisen tuomista ongelmista, erityisesti ne poistavat tarpeen takaisinkutsu-funktioiden helposti sivutusvirheisiin johtavalta hyötykuorman ylikuormittamiselta. Muutenkin riskialttiiden osoittimien käyttöä voidaan merkittävästi vähentää. Lähdekoodiesimerkistä 6 voidaan nähdä miten GTK+:n staattisen luonteen sijaan tapahtumat ja signaalit voidaan suoraan liittää luokkaan ja voidaan myöskin käyttää hyväksi oliomallin mukaista pisteoperaattoria liittämisessä. Esimerkissä on liitetty nappia painettaessa lähetettävä clicked-signaali suoraan tapahtumankäsittelijään ja vastaavasti käyttöjärjestelmän lähettämä pressed-tapahtuma on liitetty omaan käsittelijään. Esimerkin tapauksessa saattaa olla hankala nähdä eroa signaalin ja tapahtuman eroa, niillä on kuitenkin eräs merkittävä ero. Signaali on aina GTK+:n lähettämä ja signaalia lähetettäessä tapahtuma on jo hyväksytty käyttöjärjestelmälle, tapahtumankäsittelijän tapauksessa tapahtumaa ei sen sijaan ole vielä välttämättä hyväksytty ja kirjaston käyttäjän on mahdollista olla reagoimatta tapahtumaan millään tavalla tai reagoida siihen täysin poikkeavalla tavalla. [20]

```
class signaaliLuokka
{
        void nappiaPainettiin(); //signaalinkäsittelijä
        void tapahtumaSaapui(GdkEventButton *tapahtuma);//tapahtuman käsittelijä
};
main()
{
        Gtk::Button nappi;
        SignaaliLuokka vastaanottavaOlio;
        nappi.signal_clicked().connect( sigc::mem_fun(vastaanottavaOlio,
        &signaaliLuokka::nappiaPainettiin )  );
        nappi.signal_button_press_event().connect( sigc::mem_fun(vastaanottavaOlio,
```

```
        &signaaliLuokka::tapahtumaSaapui )  );

}
```

Lähdekoodiesimerkki 6: Signaalin ja tapahtuman liittäminen C++-luokan käsittelijöihin

5.6. Esimerkkejä joidenkin peruskomponenttien käytöstä

```
int main(int argc, char *argv[])
{
        GtkWidget *window = NULL, *checkbox;
        gtk_init(&argc, &argv);
        window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_container_set_border_width (GTK_CONTAINER (window), 20);
        checkbox = gtk_check_button_new_with_label("tässä on valintalaatikko");
        gtk_widget_show(checkbox);//asetetaan valintalaatikko näkyväksi
        gtk_container_add (GTK_CONTAINER (window), checkbox);
        gtk_widget_show(window); //näytetään ikkuna
        gtk_main(); //käynnistetään sovelluksen tapahtumasilmukka
        return 0;
}
```

Lähdekoodiesimerkki 7: GTK+ ikkuna, jossa yksi valintalaatikko

Lähdekoodiesimerkissä on nähtävissä että GTK+:n käyttö, ikkunointikirjaston GObject-osan oliomallisuudesta huolimatta, on hyvin proseduraalista. Näytettävät komponentit luodaan samaa kaavaa käyttävien funktioiden avulla. Esimerkiksi uusi komponentti luodaan aina funktiolla gtk_*_new, missä ”*” on komponentin nimi. Ikkunointikirjastoon viittaavat funktiokutsut alkavat aina gtk tunnisteella, jonka jälkeen funktion nimessä tyypillisesti tulee komponentin nimi johon toiminta kohdistetaan ja lopuksi vielä toiminta jota tehdään. Tyypillisesti funktiolle annetaan käsiteltävään

luokkaan osoitin, jonka kelvollisuus on ensin makron avulla tarkistettu muistivuotojen varalta ja sitten muunnettu sopivaan kantaluokan tietotyyppiin.Vastaavalla tavalla komponentit lisätään pohjapiirustus-luokkaansa gtk_container_add() funktiolla ja merkitään näytettäväksi kutsumalla gtk_widget_show() funktiota.[21]

```
int main(int argc, char *argv[])
{
  Gtk::Main kit(argc, argv);
  ExampleWindow window; //luodaan ikkunan sisältämä olio
  Gtk::Main::run(window); //käynnistetään tapahtumasilmukka
..

ExampleWindow::ExampleWindow()
: m_button("something")
{
  set_title("checkbutton example"); //asetetaan ikkunalle otsikko
  set_border_width(10); //tehdään ikkunasta 10 pikseliä leveä
  add(m_button); //lisätään valintalaatikko ikkunaan
  show_all_children(); //merkitään kaikki ikkunan sisältämät komponentit näkyviksi
}
..
class ExampleWindow : public Gtk::Window // luokka on peritty Gtkmm:n
                                         // ikkunaluokasta
..
GTK::CheckButton m_button; // valintalaatikko luokan otsikkotiedostossa
..
```

Lähdekoodiesimerkki 8: Gtkmm ikkuna, jossa valintalaatikko

Lähdekoodiesimerkissä 8 on esitelty lähdekoodiesimerkkiä 7 vastaava toiminnallisuus toteutettuna C++:lla ja GTK+:n staattisen C API:n kätkevällä Gtkmm oliokirjastolla. Yhtäläisyydet tämän ja aiemman esimerkkin välillä on helppo nähdä: komponentti lisätään samaan tapaan pohjapiirustus-luokkaan, komponentti merkitään erikseen näytettäväksi ja tapahtumasilmukka käynnistetään pääohjelman lopussa tosin Gtkmm

versiossa tapahtumasilmukan kohteeksi annetaan ikkunaa kuvaava olio. Erot ovat kuitenkin yhtäläisyyksiä suurempia: Oliomallisuuden ansiosta ikkuna- komponentti peritään yhteen luokkaan, joka sitten kapseloi sisälleen kyseisen ikkunan sisällään pitämät komponentit. Periessään oliokirjaston ikkunakomponentin luokka saa käyttöönsä kaikki ikkunan ja sen sisältämien komponenttien hallintaan tarvittavat metodit. Oliomallisuuden ansiosta vältytään globaalien funktioiden laajalta käytöltä ja vaaralliselta osoittimien käytöltä. Komponentteja voidaa myöskin käsitellä luonnollisina kokonaisuuksina niin että esimerkiksi sovelluksen eri näkymät sijaitsevat omissa muista erillisissä olioissaan.[20]

5.7. Omat muokatut komponentit

Gtkmm tarjoaa muokattujen komponenttien luomisen kolmella erilaisella tavalla. Olemassa olevia komponentteja on mahdollista periä ja siten hyödyntäen C++:n polymorfismia ylikirjoittaa niiden toteuttamien virtuaalifunktioiden toimintaa. Ylikirjoitus toimii kaikkien Gtkmm:n komponenttien osalta, mutta tyypillisimmin tällä tavoin käsitellään pohjapiirustus-luokkia. pohjapiirustus luokkien normaalit ominaisuudet eivät välttämättä riitä kattamaan kaikkia nykyaikaisten rikkaiden käyttöliittymien tarvitsemia ominaisuuksia.

Olemassa olevien komponenttien ylikirjoittamisen lisäksi Gtkmm tarjoaa luokat Gtk::Container ja Gtk::Widget. Nämä komponentit tarjoavat olemassa olevista toteutuksista riippumattoman valmiin pohjan, jonka toteuttamalla voidaan luoda uusia komponentti- ja pohjapiirustus-luokkia. Gtk::Widget tarjoaa mahdollisuuden suunnitella kokonaan uuden näköisiä komponentteja ikkunointikirjastoon käyttäen hyväksi aiemmin mainittua Cairo-grafiikkakirjastoa. Luonnollisesti myöskin olemassa olevien komponenttien vapaa yhdisteleminen on täysin mahdollista. [20]

Suoraan GTK+:aa käytettäessä omien komponenttien luominen on myöskin mahdollista. C-pohjaisuudestaan huolimatta GTK+ sisältää laajennettavan oliorakenteen. GTK+ määrittelee oliot kahden keskeisen tietueen avulla: Luokka-

rakenteen sisältävä tietue määrittelee luokan kantaluokan sekä luokan sisältämien signaalien funktio-osoittimet, olio-rakenteen sisältävä tietue pitää sisällään kaikki luokan suorituksenaikaisesti määriteltävät ominaisuudet. Näiden tietorakenteiden lisäksi jokaisen uuden GTK+-olion tulee määritellä ainakin get_type()-funktio, class_init()-funktio, init()-funktio, sekä new()-funktio ja luonnollisesti signaalien käsittelijät. Luokan get_type()-funktio kertoo uudesta luokasta GLib kirjastolle palauttamalla luokan identifioivan ID-numeron. class_init()-funktio alustaa luokkarakenteen sekä luo ja rekisteröi luokan signaalit GTK+:n kanssa. init()-funktio alustaa luokan oliorakenteen ja olion sisältämät mahdolliset muut GTK+ komponentit. new()-funktio luo lopullisen instanssin oliosta. Uudessa itseluodussa GTK+ komponentissa voidaan käyttää Cairo-grafiikka kirjastoa ja sitä voidaan käyttää hyväksi myöskin GTKmm-rajapinnan tai C++:n kautta.[21]

5.8 GTK+ suunnittelutyökalu: Glade

Glade on apuväline käyttöliittymien ulkoiseen suunnitteluun. Tyypillinen Gladen käyttötapaus on täysin vastaava Qt Designerin kanssa. Käyttöliittymää voidaan muokata sellaisena kuin se näkyy ja komponentit voidaan nimetä, jotta niihin päästään käsiksi suoraan sovelluksen lähdekoodista. Glade tuottaa suunnitellusta käyttöliittymästä XML-muotoisen kuvauksen, jota ei enää uusimmassa Gladen versiossa 3 käännetä C tai C++ muotoiseksi lähdekoodiksi. Versio 3 sisältää myöskin tuen introspektiolle, joten Gladea käyttäen on mahdollista luoda käyttöliittymäoliolle ajonaikaisia ominaisuuksia ja signaaaleja.[22]

Gtkmm rajapintaa käytettäessä käytetään hyväksi libglademm nimistä kirjastoa. Gtkmm:n mukana toimitettava kirjasto täytyy linkittää mukaan sovelluksen lähdekoodiin ja sen avulla voidaan suoraan Gtkmm sovelluksen C++ lähdekoodista hallita XML-tiedoston avulla määriteltyjä komponentteja. [20]

```
Gtk::Dialog* pDialogi = 0;
Glib::RefPtr<Gnome::Glade::Xml> refXml;
```

refXml = Gnome::Glade::Xml::create(”basic.glade”);
refXml->getWidget(”Dialogi”, pDialogi);
Lähdekoodiesimerkki 11: Glade dialogin liittäminen C++ lähdekoodiin

Lähdekoodiesimerkistä on nähtävissä miten Gladen tuottamasta tiedostosta voidaan luoda dialogi sovellukseen. Koska Glade tuottaa GTK+ yhteensopivan XML-tiedoston, joudutaan Gladen tuottamia tiedostoja Gtkmm:stä käytettäessä käsittelemään Glib yhteensopivilla osoittimilla[20]. Gladen käyttö C-pohjaisesta GTK+ kirjastosta on mahdollista käyttämällä Gladen versiota 2, joka tuottaa käyttöliittymän suoraan C lähdekoodina. Tai käyttämällä Libgladea Gtkmm:ää vastaavalla tavalla: Luodaan Glib osoitin glade-tiedostoon ja luodaan osoittimen avulla näytettäväksi halutusta komponentista ilmentymä.

5.9 Piirtorutiinien toteutus eri alustoilla: Cairo

Ennen versiota 2.8 GTK+ käytti piirtämisessä hyväkseen omaa GDK kirjastoaan joka oli sitten toteutettuna erilaisille käyttöjärjestelmille ja alustoille. Versiosta 2.8 lähtien GTK+ on kuitenkin käyttänyt hyväkseen Cairoa piirtäessään komponenttejaan. Koska Cairo on monialustainen kaksiulotteiseen piirtoon erikoistunut grafiikkakirjasto niin GTK+:n avulla voidaan piirtää grafiikkaa käyttäen samaa rajapintaa eri alustoilla. Tämän hetkisen Cairon version 1.4 avulla voidaan piirtää Unixien ja Linuxien käyttämälle X11-palvelimelle, Windowsin Win32 rajapinnalle, Postscript tiedostoihin, PDF-tiedostoihin sekä SVG-tiedostoihin. Tuki Mac OS X:n käyttämälle Quartz rajapinnalle oli tätä kirjoitettaessa kokeellinen. Cairo kuten GTK+ on kirjoitettu käyttäen hyväksi C-ohjelmointikieltiä, mutta Cairon käyttöön löytyy rajapinta C++:lle ja muille yleisimmille ohjelmointikielille.Cairossa kuten Qt:ssa on pyritty käyttämään kullakin alustalla mahdollisimman matalan tason rajapintaa piirtorutiinien toteutukseen. [23]

GTK+ tarjoaa mahdollisuuden kirjoittaa uusia GObjecteja käyttäen hyväksi Cairoa.

GTK+:n sisältä voidaan suoraan määritellä Cairon piirtopintoja ja tehdä funktiokutsuja Cairon rajapintoihin. Cairon ansiosta GTK+:lla voidaan haluttaessa toteuttaa käyttöjärjestelmän ulkonäöstä täysin riippumattoman näköisiä sovelluksia. Periaatteellisena filosofiana GTK+:ssa on kuitenkin että standardeja GTK+:n komponentteja käyttävä sovellus näyttää ja toimii samalla tavalla käyttöjärjestelmästä riippumatta.[23]

6. WXWIDGETS ARKKITEHTUURI

Kappaleessa esitellään wxWidgetsin ominaisuuksia sekä käsitellään tyypilliseen wxWidgets toteutukseen liittyviä komponentteja. Kappale määrittelee käsiteltävät wxWidgetsin osat. Käsiteltävät osat ovat wxWidgetsin vastineita edellä esitellyille Qt ja GTK+ ikkunointikirjastoille.

6.1 Luokka- ja modulirakenne

Taulukko 7: wxWidgetsin kirjastot[24]

Kirjaston nimi	***Sisältö***
wxAui	Tuki kelluville ikkunoille ja MDI sovelluksille
wxBase	wxWidgetsin pakolliset komponentit.
wxNet	Tietoverkko-ohjelmointiin tarvittavat luokat
wxRichText	Muotoilut sisältävän tekstin käsittely(eng. rich text).
wxXML	XML-tiedostojen käsittely
wxCore	wxWidgetsin sisältämät graafiset komponentit.
wxAdvanced	Erikoisemmat graafiset ja ei-graafiset luokat.
wxMedia	Luokat multimedian käsittelyyn.
wxGL	OpenGL kirjaston käyttämiseen tarvittavat luokat.
wxHTML	HTML tiedostojen piirtämiseen tarvittavat luokat
wxODBC	Tietokantaohjelmointiin tarvittavat luokat.
wxQA	Laadunvarmistukseen tarvittavat luokat

Taulukossa 7 on esitelty wxWidgetsin tarjoamat kirjastot. Tyypillisessä wxWidgetsiä

käyttävässä toteutuksessa tarvitaan aina wxBase ja wxCore moduuleita. Jokaisen wxWidgets-ohjelman tulee vähintäänkin linkittyä wxBase moduuliin, joka pitää sisäällään kaikki wxwidgetsin keskeiset luokat, esimerkiksi merkkijonot. Pelkästään wxBase-moduulia käyttämällä voidaan luoda alustariippumattomia komentorivisovelluksia. wxCore pitää puolestaan sisällään kaikki graafiseen käyttöliittymään tarvittavat yleisimmät peruskomponentit. Jokaisen graafisen käyttöliittymän sisältävän wxWidgets sovelluksen tulee linkittyä vähintään wxBase ja wxCore kirjastoihin. Useimmin sovelluksessa tarvittavat ominaisuudet ovat saatavissa kirjaston muista moduuleista ja luonnollisesti kirjastoa on aina mahdollista laajentaa omiin tarpeisiin sopivaksi. Tyypillisestä wxWidgets-sovelluksesta löytyy taulukossa olevien wxWidgets kirjastojen lisäksi myös kuvien käsittelyyn tarkoitetut libjpg tai libpng.

6.2 Yhteensopivat käännösympäristöt ja kääntäminen

WxWidgetsin tämän hetkinen versio 2.8 tukee wxWidgetsin wikin mukaan kaikkia yleisimpiä kääntäjiä mukaanlukien GCC:n, Borland C++:n ja Microsoft Visual Studion kaikki versiot. WxWidgets ei sisällä itsessään valmiita lisäkkeitä millekään IDE:lle. Kolmannen osapuolen tuottamia kaupallisia ja avoimen lähdekoodin laajennoksia löytyy valtava määrä. Myöskin wxWidgetsiä käyttäviä monialustaisia ja avoimia kehitysympäristöjä on kymmenittäin, tyypillisesti nämä IDE:t perustuvat GCC työkalusettiin.[25]

WxWidgetsin perustuvan projektin luonti alkaa etsimällä käytetylle käyttöjärjestelmälle oikeanlainen versio. Tämän lisäksi tulee valita käytetylle käännösympäristölle sopiva versio haetun wxWidgets kirjaston sisältä. Riippuen haetusta versiosta, käännetään kirjasto halutulle kääntäjälle tai suoritetaan asennusohjelma, joka asentaa tarvittavat tiedostot käyttöjärjestelmään. Käännöksen tai asennuksen jälkeen asetetaan sovelluksessa tarvittavat kirjastot käännösympäristöön, sekä lisätään käyttöjärjestelmään WXWIN-ympäristömuuttuja osoittamaan kirjaston sijaintiin. Sopivat käännösliput asennetulle wxWidgetsille saadaan kirjaston mukana toimitettavan

wx-config sovelluksen avulla. [25]

6.3 Ei-näkyvät komponentit

WxWidgets sisältää omat tietosäiliöluokkansa. Nämä luokat vastaavat toiminnallisuudeltaan 90% STL:än vastaavia tietorakenteita. Kirjastoon on myöskin sisäänrakennettu tuki STL-kirjaston sisältämille tietorakenteille ja tuki voidaan kääntää päälle kirjaston käännösasetuksista. Tämä kuitenkin pidentää kirjaston käännösaikaa ja kirjaston ajonaikaista kokoa. Lisäksi STL:n mukaiset tietosäiliöt täytyy muuttaa takaisin kirjaston tietosäiliöluokkiin mikäli niiden sisältöä aiotaan näyttää käyttäjälle. Tästä johtuen Julian Smart ja Kevin Hock suosittelevatkin kirjassaan käyttämään ennemmin wxWidgetsin omia sisäisiä tietosäiliöluokkia.[11]

wxWidgetsin wxString pitää sisällään merkkijono-operaatiot. Mikäli UTF-8 on käännettynä päälle kirjaston setup.h-tiedostosta, merkkijono luokka tukee automaattisesti kansainvälistä tekstiä. WxWidgets tarjoaa makrot myöskin merkkijonovakioiden automaattiseen määrittämisen kansainväliseksi tekstiksi. Mikäli esimerkiksi wxString merkkijonoon halutaan sijoittaa tavallisen ANSI merkistön mukaisen char*-merkkijonon sijaan Unicode-merkkijono, tulee merkkijono syöttää wxWidgetsin tarjoamaan wxT-makroon.[11]

Taulukko 8: wxWidgetsin yleisiä säiliöluokkia

Tietorakenteen tietotyyppi	***Tietorakenteen selitys***
wxString	Suorituksenaikaisesti suurenettava ja pienenettävä merkkijono
wxArray, wxArrayString, wxObjArray	Tietoalkioiden peräkkäishakuun käytettävät, ajonaikaisesti muistia käsittelevät taulukot.
wxListBase	wxNode alkioita sisältävä kahteen suuntaan linkitetty lista.
wxHashMap	Assosiatiivinen taulukko satunnaishakuun

Taulukossa 8 olevat wxWidgetsin tietoalkoita sisällään pitävät tietorakenteet ovat, wxStringiä lukuunottamatta, eroavia standardin C++ määritelmän mukaisesta tietotyyppien määrittelystä. Tietotyyppien käyttö edellyttää tietotyypin määrittelyä vastaavien makrojen lisäämistä sekä tietotyyppiä käyttävän olion otsikko- että toteutustiedostoihin. Kullakin wxWidgetsin säiliöluokalla on omanlaisensa tapa määritellä tietotyyppinsä. Tyypillisesti kuitenkin voidaan sanoa että otsikkotiedostoon lisätään WX_DECLARE-alkuinen makro ja toteutus tiedostoon, normaalin säiliöluokan otsikkotiedoston jälkeen, WX_DEFINE-alkuinen makro. wxWidgets sisältää myöskin tarvittavat komponentit säännöllisten lausekkeiden, päivämäärien ja erilaisten käyttöliittymän mittojen ilmaisuun. Myös minkä tahansa perustietotyypin sisältävä wxVariant luokka on keskeinen osa wxWidgetsiä käyttäjälle näkymättömiä komponentteja.[11]

Useat wxWidgets luokat käyttävät hyväkseen kopiointi kirjoitettaessa -menelmää(eng. copy on write). Käytännössä tämä tarkoittaa että, kun olio siirretään toiselle oliolle kopiointia ei itseasiassa tapahdu ollenkaan, vaan kirjasto kasvattaa ainoastaan laskuria siitä montako viittausta samaan olioon on tehty. Oliosta ei tehdä tarpeetonta kopiota ennenkuin ohjelmassa halutaan muuttaa olion sisältöä. Myöskään oliota ei poisteta ennenkuin laskuri sitä käyttävistä muista olioista menee nollaan. WxWidgets mahdollistaa myöskin käyttäjän omien kopionti kirjoitettaessa menetelmää tukevien olioiden kirjoittamisen. [24]

6.4 Käyttöliittymäkomponentit ja niiden hallinta

WxWidgetsin kaikki luokat perivät wxObject kantaluokan. Luokan tarjoaman metainformaation avulla voidaan saada tietoa kantaluokan osoittamasta luokasta, sekä tarvittaessa hallita kaikkia luokkia yhteisillä osoittimella. Kaikki käyttöliittymässä näkyvillä olevat wxWidgetsin luokat perivät wxWindow-luokan. Tämä perimäketju mahdollistaa näkyvien komponenttien automaattisen poiston. Kun esimerkiksi sovelluksen ikkuna poistetaan, poistetaan myös kaikki ikkunan sisältämät wxWindow-luokan perivät komponentit. Yhteinen wxWindow kantaluokka tarjoaa jokaiselle

aliluokalle yhteiset käsittelymetodit sekä muuttujat. Yhteisissä käsittelymetodeissa on eräs erityisesti huomattava asia. Koska wxWidgets ohjaa useimmat Get-alkuiset metodit omalle sisäiselle doGet-alkuiselle funktiolleen niin, vastoin C++:n standardia toimintatapaa, kirjaston sisäiset metodit tulee ylikirjoittaa ilman funktioiden määrittelemistä virtuaaliseksi. [26]

WxWidgets tarjoaa perustoiminnaltaan hyvin pitkälti muiden yleisten kirjastojen kaltaiset pohjapiirustusluokat. Kaikki wxWidgetsin pohjapiirustusluokat omistavat joukun wxWindow-luokan periviä näkyviä komponentteja, sekä tietenkin myös tyhjää tilaa. Sovelluksen ikkunaa luotaessa, ikkuna kysyy omistamaltaan pohjapiirustusluokalta sen tarvitseman tilan ja määrää oman kokonsa tämän ohjeen mukaan. pohjapiirustusluokka puolestaan laskee kertomansa koon sisältämiensä näkyvien luokkien ja tyhjän tilan perusteella. Tyhjän tilan ja komponenttien koon pohjapiirustusluokka määrittelee itseensä asetettujen ja sisältämiinsä komponentteihin asetettujen arvojen perusteella. Ikkunan kokoa muutettaessa pohjepiirrustusluokka säätää hallitsemiensa komponenttien sijainnin ja koon omien, sekä komponenttien itsensä pohjapiirustuluokalle kertoman informaation perusteella. wxWidgets sisältää peruskomponenttien lisäksi paljon valmiita yleisesti käytettyjä ikkunoita. Valmiit komponentit eivät rajoitu vain viestilaatikoihin, lisäksi löytyy esimerkiksi valmiit luokat yleisimpiin sovelluksen osiin kuten esimerkiksi haku- tai tiedostonvalinta-dialogeihin. [26]

6.5 Asynkronisten viestien toteutus

WxWidgets nojaa ikkunointikirjaston viestien ja tapahtumien liittämisessä kahteen rinnakkaiseen menetelmään. Ajonaikaiseen sekä käännöksenaikaiseen. Käännöksenaikainen menetelmä nojaa muutamaan pääkohtaan. Tapahtumia käsittelevän luokan tulee periä wxWidgetsin wxEvtHandler luokka. Luokan otsikkotiedostoon tulee lisätä wxWidgetsin DECLARE_EVENT_TABLE()-makro, sekä käytettävät tapahtumankäsittelijäfunktiot. Tapahtumakäsittelijäfunktioiden tulee vastaanottaa käsitelemäänsä tapahtumaa vastaava event muuttuja. Oikea event-muutujan tyyppi on

löydettävissä wxWidgetsin dokumentaatiosta. Luokan toteutustiedostoon tulee lisätä wxwidgetsin BEGIN_EVENT_TABLE()-makro, jolle annetaan parametriksi sekä tapahtumat lähettävä wxWidgets luokka että kyseisen luokan perivä oma toteutettava luokka.[11]

```
BEGIN_EVENT_TABLE(OmaLuokka, wxFrame)
        EVT_MENU(wxID_EXIT,     OmaLuokka::onQuit)
END_EVENT_TABLE()
```

Lähdekoodiesimerkki 7: wxWidgets tapahtumataulukko

Tämän jälkeen luetellaan käsiteltävät tapahtumat. Mikäli liitettävällä tapahtumalla voi olla erilaisia tapahtuman lähteen yksilöiviä vakioita niin ne annetaan tapahtumatyyppi-makron ensimmäisenä parametrina. Toisena parametrina kerrotaan oman luokan funktio, johon tapahtuma liitetään. Lähdekoodiesimerkissä 7 on annettu esimerkki tapahtumataulukosta, joka liittää omaLuokka-ikkunan sulkemisen saman luokan onQuit funktioon. WxWidgetsin ikkuna määrittelee ikkunan sulkemisen menu-tapahtumaksi, joka erikseen määritellään sulkemiseksi wxID_EXIT-vakion avulla, joten EVT_MENUn ensimmäisenä parametrina on annettu wxID_EXIT vakio. Lähdekoodiesimerkin kaltainen vakion määrittely ei ole pakollinen. Tapahtuma voidaan liitää myös suoraan sitä käsittelevään funktioon, antaa tapahtumalle tapahtumatyypiksi wxID_ANY ja sitten funktion sisällä tutkia, saatavasta tapahtumaoliosta, minkä tapahtuman lähteen identifioivan vakion se sisältää. Lähdekoodiesimerkin kaltainen tapahtumataulukon määrittely makrojen avulla ei kuitenkaan ole pakollista. Tapahtumia voidaan hallita myöskin ajonaikaisesti wxEvtHandler::Connect ja wxEvtHandler::Disconnect-funktioiden avulla.[11]

```
omaIkkuna->connect(wxID_EXIT,
                   wxEVT_COMMAND_MENU_SELECTED,
                   wxCommandEventHandler(OmaLuokka::onQuit) );
```

Lähdekoodiesimerkki8: wxWidgets tapahtuman ajonaikainen liittäminen

Lähdekoodiesimerkissä 8 on liitetty sama tapahtuma kuin lähdekoodiesimerkissä 7,

mutta tällä kertaa tapahtuma on liitetty ajonaikaisesti. Ensimmäisenä parametrina funktiolle annetaan tapahtumatyypin yksilöivä vakio tai kahtena parametrina vakio väli, tämän jälkeen annetaan tapahtumaluokan tyyppi ja lopuksi annetaan tapahtumaluokan tapahtumia käsittelevän funktion palauttama arvo. WxWidgets mahdollistaa myöskin omien tapahtumien luomisen. Käyttäjä voi määritellä oman tapahtumansa kahdella tavalla. Joko käyttäen hyväksi olemassa olevia tapahtumaluokkia ja määritellä näille sopivat uudet tapahtumatyypit. Tai sitten käyttäjä voi luoda kokonaan uuden tapahtuman perimällä wxEvent kantaluokan ja määrittelemällä luokalle tarvittavat makrot. Olemassa olevia tapahtumaluokkia käyttäen tapahtuman määrittely tapahtuu lisäämällä DECLARE_EVENT_TYPE()-makro luokan tapahtumataulukkoon ja antamalla makrolle tapahtuman nimi parametrina. Toteutustiedostoon tulee tämän lisäksi lisätä DEFINE_EVENT_TYPE()-makro. [27]

```
wxCommandEvent tapahtuma(wxEVT_OMA_TAPAHTUMA, GetId());
tapahtuma.setEventObject(this);
GetEventHandler()->ProcessEvent(tapahtuma);
```

Lähdekoodiesimerkki 9: Oman tapahtuman lähettäminen wxWidgetsissä[27]

Lähdekoodiesimerkissä 9 on esimerkki commandEvent typpisen oman tapahtuman lähettämisestä. Ensin luodaan tapahtuma olio, jolle annetaan parametriksi tapahtumatyyppi, sekä tapahtuman lähettävän luokan tunnusnumero. Esimerkin tunnusnumero saadaan suoraan kirjaston itsensä tähän olioon automaattisesti määrittelemältä GetId()-funktiolta. Tapahtuman käsittelevän olion asettamisen jälkeen tapahtuma annetaan tapahtumankäsittelijän hoidettavaksi.[27]

6.6. Esimerkkejä joidenkin peruskomponenttien käytöstä

wxWidgets sovellus rakentuu pääikkunan ympärille, joka perii tyypin wxFrame tai wxDialog. Pääikkuna luodaan wxWidgets ohjelman käynnistävän, wxApp olion perivän, pääolion OnInit()-funktiossa. Sekä wxFrame että wxDialog ottavat vastaan samanlaiset parametrit. Kolme ensimmäistä parametria ovat pakollisia kaikille päätason

ikkunoille. Kaksi ensimmäistä parametria ovat pakollisia kaikille wxWidgets komponenteille. Ensimmäinen parametri on osoitin ikkunan tai komponentin vanhempaan. Tämä on tarpeellista, koska wxWidgets poistaa luotavan lapsikomponentin automaattisesti lapsikomponentin antaman osoittimen yhteydessä. Pääikkunan tapauksessa tämä parametri on tavallisesti NULL, koska päätason ikkuna on oliohierarkian ylimmäinen olio ja se poistetaan automaattisesti kun ikkuna suljetaan. Toisena kaikille pakollisena parametrina annetaan ikkunan tai komponentin identifioiva vakio, jota käytetään muun muassa viestien käsittelyn yhteydessä. Päätason ikkunoille annetaan tämän lisäksi ikkunan otsikon sisältävä parametri. Tämän jälkeen tulevilla vapaaehtoisilla parametreilla on mahdollista määritellä ikkunan sijainti, koko, tyyli ja ikkunan nimi. WxWidgets sisältää suuren määrän valmiita ikkunoita erilaisiin tarkoituksiin, lisäksi on useita erilaisia valmiita ikkunan osia. Lähdekoodiesimerkissä 9 on esitettynä tyhjän ikkunan luonti. [24]

*new wxFrame((wxFrame *) NULL, OMA_VAKIO_ID, ”ikkunan otsikko”);*
Lähdekoodiesimerkki 9: uuden päätason ikkunan luonti

Tyypillisessä wxWidgets sovelluksessa pääikkuna on laajennettu joko wxDialogista tai wxFramesta. Laajennetun luokan rakentaja-funktiossa sitten luodaan tarvittavat komponentit. Tyypillisesti komponentti merkitään pääikkunan lapseksi. Rakentajassa myöskin asetetaan kaikki ikkunan erikoispiirteet, kuten työkalupalkit ja menut, joiden toteutus usein sijaitsee muissa luokissa. Lopuksi kun ikkunan kaikki komponentit on luotu ikkuna asetetaan näkyväksi. wxWidgetsin sisältämät kontrollit eivät eroa ikkunoista ja muista komponenteista periaatteessa mitenkään. Merkittävänä erona voidaan oikeastaan ainoastaan pitää että kontrollit tavallisesti sijoitetaan näkymättömän pohjapiirustusluokan hallittavaksi, jotta niiden sijainti pysyisi järkevänä ikkunan koon muuttuessa,[24]

6.7. Omat muokatut komponentit

Ikkunoiden osalta wxWidgets nojaa varsin vahvasti kirjaston tarjoamien valmiiden

ikkunoiden perimiseen ja niiden laajentamiseen. Valmiita ikkunoita löytyy yleisimpiin tarkoituksiin, kuten tiedoston avaamiseen wizardeihin ja niin edelleen. Omien muokattujen komponenttien luomiseksi täytyy laajentaa olemassaolevaa kirjastoa. Laajentamisen kannalta merkittävää on ensinnäkin funktioiden ylikirjoittaminen ja toisaalta polymorfismin mahdollistava ajon aikainen tyypin määritys, joka paremmin tunnetaan englanninkielisellä nimellä run-time type identification(eng.) eli RTTI. wxWidgets pitää sisällään oman makroihin nojaavan RTTI toteutuksen, eikä täten käytä hyväkseen C++:n sisäisesti tarjoamaa toteutusta. wxWidgetsin komponentteja laajennettaessa ei siis käytetä ollenkaan virtuaalisia funktioita. Polymorfismi toteutetaan wxWidgetsissä antamalla otsikkotiedostossa luokan nimi DECLARE_DYNAMIC_CLASS()-makrolle ja antamalla kantaluokan ja luokan nimi IMPLEMENT_DYNAMIC_CLASS()-makrolle joka sijaitsee toteustustiedostossa.[24]

Omaa kontollia luotaessa wxWidgets mahdollistaa vapaan perintäketjun, luotava komponentti voi olla peritty yhtä hyvin wxWidgetsin geneerisestä wxControl luokasta kuin wxWidgetsin muistakin luokista. WxWidgetsin sisältämät ikkunoiden osat eivät peri samaa kantaluokkaa kuten useimmissa muissa kirjastoissa. Uuden wxWidgets kontrollin tulee toteuttaa edellisessä kappaleessa mainitut RTTI-makrot, jotta se voisi laajentaa kirjaston edellyttämät pakolliset metodit eli esimerkiksi Create()-funktion ikkunan luomiseen ja DoGetBestSize()-funktion ikkunan minimikoon määrittämiseen. Koska itsemääritellyille komponenteille eivät usein riitä wxWidgetsin mukava tulevat tapahtumaluokat, niin uudelle kontrollille määritellään usein myös oma tapahtumaluokka. Uudelle kontrollille ikkunointikirjastolta saapuvat komennot tulee myöskin liittää niitä käsitteleviin metodeihin. Erityisen tärkeitä ovat käyttöliittymän päivittymiseen liittyvät komennot. Useimmiten myöskin määritellään metodit matalan tason hiiri ja näppäimistökomentojen käsittelemiseen, sekä tietenkin itse komponentin sisältämän näytettävän tiedon näyttävä metodi.[11]

6.8 wxWidgets suunnittelutyökalut

wxWidgetsille on tarjolla useita erilaisia työkaluja käyttöliittymien luomiseen, niillä

kaikilla on kuitenkin eräs yhteinen piirre: ne luovat XML-pohjaisia XRC-tiedostoja. XRC-tiedostot pitävät sisällään hyvin määriteltyä XML:ää, joka on tarkasti wxWidgetsin määrittelemä. Tästä syystä niitä on helppo editoida myös tavallisella tekstieditorilla tarpeen niin vaatiessa. wxWidgetsin manuaali mainitsee XRC-tiedostojen eduksi ettei sovellusta tarvitse joka kerta kääntää ja linkittää uudelleen jos resurssit muuttuvat. XRC-tiedoston käyttötapaus alkaa käyttöliittymän määrittelemisellä. Tuotettu XML-tiedosto voidaan tämän jälkeen liittää suoraan sovellukseen. Sovellus tulee liittää wxWidgetsin tarjoamaan xmlres.h otsikkotiedostoon, jonka tarjoaman wxXmlResource-rajapinnan avulla voidaan sitten ladata xrc-tiedosto ja ottaa sen tarjoamat resurssit käyttöön. Myös XRC-tiedostossa esitellyt tapahtumat voi ottaa suoraan käyttöön kunhan ne suodatetaan XRCID-makron lävitse. Makron tarkoituksena on muuttaa XML-tiedoston tapahtumat määrittelevät numerovakiot toisistaan riippumattomiksi. WxWidgetsin mukana toimitetaan wxrc sovellus, jonka avulla on mahdollista muuntaa xrc-tiedostoja, binäärimuotoisiksi xrs-tiedostoiksi tai suoraan C++ lähdekoodiksi. [24]

6.9 wxWidgets ja GTK+ kielenkäännös : GNU gettext

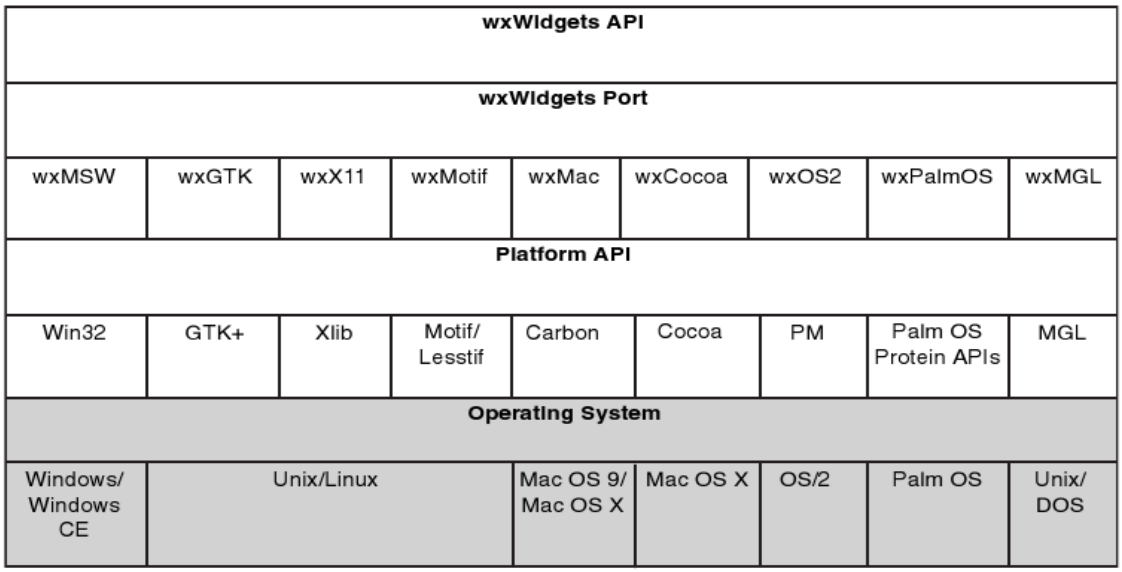
wxWidgets ja GTK+ nojaavat kielenkäännöksen osalta gettext pakettiin, jonka libintl(i18n)- ja libasprintf-kirjastot on lisensoitu LGPL lisenssin mukaisesti. Tosin wxWidgets ei käytä gettext pakettia suoraan, kuten GTK+, vaan sen sisältämä käännösjärjestelmä tuottaa gettextin kanssa binääriyhteensopivia käännöstiedostoja. Tiedostoja voidaan sitten kääntää gettextin sisältämäillä työkaluilla. Gettext saadaan käyttöön Windows käyttöjärjestelmässä edellä mainittuihin kirjastoihin linkittymällä tai GNU järjestelmissä automaattisesti. Gettextin peruskäyttö ohjelmoijan näkökulmasta on suhteellisen yksinkertaista: kirjaston otsikkotiedoston liittämisen lisäksi tarvitsee määritellä kirjaston käyttämät _() ja gettext() funktiot esikääntäjälle. Määritellä käännöstiedostojen sijainnit ja valitun kielen sisältävät vakiot joko sovelluksen Make-tiedostoon tai suoraan käännettävään ohjelmaan. WxWidgetsin tapauksessa käännöksen määrittely on vielä yksinkertaisempaa: Ohjelmoijan tulee asettaa wxLocale luokan avulla ohjelmaansa kohdejärjestelmän kieli ja lisäksi tarvitsee määritellä sovellukseensa

käännettäviksi haluttavat merkkijonot wxGetTranslation luokan tai _() makron avulla. Mikäli sovelluksen kielitiedostot eivät sijaitse vakiosijainnissa tulee vielä asettaa AddCatalogLookupPathPrefix() funktion avulla niiden sijainti.[25][26]

Ohjelmakoodin ulkopuolella käännöstä valmistellessa toimitaan tyypillisesti seuraavalla tavalla. Kun suunnittelija saa sovelluksen riittävään pisteeseen hän muuttaa kaikki edellä mainituilla makroilla tai funktiokutsuilla merkityt merkkijonot yhdeksi .po päätteiseksi käännöstiedostoksi. Tämän käännöstiedoston hän sitten toimittaa kääntäjälle. Kääntäjä voi käyttää haluamaansa avoimen tai suljetun lähdekoodin sovellusta .po-tiedoston kääntämiseen kohdekielelle. Tunnettuja sovelluksia ovat esimerkiksi Gtranslator, KBabel tai poEdit. Kun kääntäjä on saanut käännöstyönsä päätökseen, hän toimittaa ohjelmistokehittäjälle tuottamansa kohdekielisen PO-tiedoston. Ohjelmistokehittäjä kääntää .po tiedoston binääriseksi .mo tiedostoksi käyttäen gettext-paketin sisältämää msgfmt-sovellusta. Tämän jälkeen kehittäjän tarvitsee enää kopioida .mo tiedostot sovelluksen käännöstiedosto-kansioon, jonka jälkeen hän voi muuttaa ohjelman lähdekoodista käsin tai käännöksen yhteydessä sovelluksen merkkijonot haluamaansa kieleen.[28]

6.10 Piirtorutiinien toteutus eri alustoilla

Kuten kuvassa 2 on nähtävissä wxWidgets toteuttaa piirtorutiininsa sekä muiden käyttöliittymäkirjastojen avulla, että matalan tason piirtorajapintoja käyttämällä. Periaatteessa wxWidgets määrittelee ainoastaan toteutettavan rajapinnan. Tämä rajapinta voidaan sitten toteuttaa vapaasti valittavalla ikkunointikirjastolla tai piirtorajapinnalla. Näin muodostuu wxWidgets käännös, jonka tiedostoihin linkittämällä voidaan luoda kyseisellä rajapinnalla tai ikkunointikirjastolla toteutettu sovellus.



Kuva 2: wxWidgets arkkitehtuuri[11]

Kirjastoa käyttävän ohjelmoijan tulee siis kääntää sama sovellus uudelleen kutakin alustaa varten. wxMSW on kaikkia 64- ja 32-bittisiä windowseja tukeva käännös, lisäksi käännös voidaan kohdistaa myös Windows CE laitteisiin. Sovellusta käännettäessä on mahdollista valita käyttääkö sovellus windowsin omia standardeja grafiikoita vai wxWidgetsin universaaleja grafiikoita. wxGTK, wxX11 ja wxMotif käännökset tarjoavat toiminnallisuuden Unix ja Linux alustoille. wxGTK on suositeltavin käännös, koska sitä on kehitetty kaikkein kauimmin ja se sisältää kaikki wxWidgetsin komponentit wxWidgetsin versiossa 2.6. wxX11 käyttää hyväkseen hyvin matalan tason Xlib kirjastoa ja soveltuu lähinnä sulautetuille laitteille. Koska Sun Microsystems on siirtymässä käyttämään Solaris käyttöjärjestelmässään GTK:ta, wxMotif on poistumassa oleva käännös. WxWidgetsistä löytyy käännös kummallekin Mac Os:n rajapinnalle niin vanhemmalle Carbonille kuin uudemmalle Cocoalle. Tätä diplomityötä kirjoitettaessa Cocoan toteutus oli vielä merkittävästi puutteellinen. WxMGL käännös tarjoaa mahdollisuuden toteuttaa sovellus käyttäen ainakin windowseissa ja Linuxeissa toimivaa Scitechin sulautettuihin järjestelmiin suunniteltua MGL kirjastoa.[11]

7. ARKKITEHTUURIEN EROJEN POHDINTA

Kappaleessa pohditaan aiemmin esiteltyjen kolmen ikkunointikirjaston. Qt:n, wxWidgetsin ja Gtk+:n arkkitehtuureja. Kappaleessa tuodaan kootusti esille arkkitehtuurien erot, jotta niitä olisi helpompi käyttää hyödyksi analyysituloksien pohdinnassa.

7.1. Luokka- ja modulirakenteet

Tässä työssä esiteltävät ikkunointikirjastot jakautuvat kahteen osaan: kirjasto on joko koko toiminnallisuuden sisältävä monoliittinen ratkaisu tai sitten se koostuu useista erillisistä, jopa eri toimittajilta saatavista erillisistä osista. Koska Qt käyttää suoraan matalinta käyttöjärjestelmän tarjoamaa rajapintaa piirtämiseen ja muihin tarjoamiinsa toimintoihin niin Qt on tarjotuista ratkaisuista selkeimmin monoliitti ja täten siis eniten muista riippumaton itsellinen komponentti. WxWidgets on hieman vähemmän itsellinen kuin Qt, sillä se saattaa olla, riippuen käytetystä käännöksestä, riippuvainen toisesta ikkunointikirjastosta. Lisäksi wxWidgets käyttää esimerkiksi kuvien käsittelyssä hyväkseen täysin alustakohtaisia kirjastoja, esimerkiksi libjpg Linuxissa. GTK+ on vähiten itsellinen. Pelkästään piirron toteuttamiseen tarvitaan Cairon apua tai C++:n käyttöön Gtkmm kirjastoa.

Ikkunointikirjaston rajapinnan tai toiminnallisuuden muutoksessa Qt:ta käyttävä kehittäjä arvioi muutoksen vaikutukset omaan lähdekoodiinsa. WxWidgetsin kehittäjä arvioi vaikuttaako muutos muihin käyttämiinsä kirjastoihin, sekä lisäksi hän voi joutua arvioimaan wxWidgets käännöksen käyttämän kirjaston mahdollisia muutoksia. GTK+ käyttävä kehittäjä joutuu arvioimaan muutoksen vaikutuksen C++-rajapinnassa, Cairossa ja mahdollisesti itse C-pohjaisessa rajapinnassa. Lisäksi hän saattaa joutua arvioimaan ristiin vaikutusta kaikkiin muihin käyttämiinsä komponentteihin, mikäli ne ovat vaikutuksen alaisia. Lisäksi tulee kuitenkin huomata että C:tä käyttävä GTK+-kehittäjä välttyy kuitenkin rajapintamuutosten tapauksessa

versioyhteensopivuusongelmilta sillä, johtuen kielien sisäisestä toteutuksesta, C-rajapinta ei vaadi yhtä tarkkaa binääriyhteensopivuutta kuin C++ rajapinta.

Huolimatta edellä mainituista mahdollisista ylläpito-ongelmista, voimakkaammin hajautetun rakenteen käyttäminen ei aina ole pelkästään haitallista. Se tarjoaa myös mahdollisuuden helposti korvata sovelluksen käyttämiä osia toisilla tai kirjoittaa niitä itse kokonaan uudelleen. Hajautetumman kirjaston eri osissa on myös vähemmän lähdekoodia ja tästä johtuen on todennäköisempää että käytetyn kirjaston lähdekoodi sisältää vähemmän vain ikkunointikirjaston käyttöön soveltuvaa lähdekoodia. Näin ollen lähdekoodin siirrettävyys erilaisiin ympäristöihin paranee. Monoliittisia kirjastoja huomattavasti pienemmällä vaivalla voidaan siirtää kokonaisia sovelluksia uusille alustoille. Hajautetun kirjaston osia on helpompi käyttää hyväksi erilaisissa käyttötarkoituksissa, koska kirjaston osat ovat edellä mainituista syistä yleiskäyttöisempiä. Tästä johtuen niillä on usein enemmän käyttäjiä ja paine lähdekoodin parantamiseen muodostuu voimakkaammaksi.

Kirjastojen tarjoamat luokkarakenteet muistuttavat toisiaan hyvin suurelta osin. Kaikki kirjastot pohjautuvat, pienin merkityksettömin eroin, yleisesti hyväksyttyyn oliomallin mukaiseen hierarkia-rakenteeseen. Kaikki kirjastot noudattavat samaa ylhäältäpäin erilaistavaa näkemystä hierarkiaan. Komponenttien sijoittelussa hierarkiaan on pieniä eroja. Erot johtuvat erilaisesta näkemyksestä luokkien erikoistumiseen ja erilaisista tarpeista kirjastojen sisäisten toimintojen toteuttamisen osalta. Koska näkemys oikeanlaisesta hierarkiasta on hyvin voimakkaasti käyttäjäkohtainen, nämä pienet erot voidaan nähdä kirjaston käytettävyyden kannalta merkityksettöminä.

7.2 Yhteensopivat käännösympäristöt ja kääntäminen

Esitellyille avoimen lähdekoodin ikkunointikirjastoille kaikkein yhteensopivin kääntäjä on avoimen lähdekoodin GCC-kääntäjä. GCC:tä voidaankin pitää pääasiallisena kääntäjänä työssä esitellyille kirjastoille. GCC kääntäjän tukeminen mahdollistaa erittäin laajan määrän tuettuja käännösympäristöjä, erityisesti avoimessa Linux

käyttöjärjestelmässä. Windowskehitys on myös mahdollista käyttäen GCC-kääntäjää. Tällöin tulee kuitenkin käyttää hyväksi MinGW tai Cygwin kirjastoja, jotka emuloivat Linuxin järjestelmärajapintoja Windowsilla. Edellä mainittua emulointia käytettäessä sovellus tulee kääntää kirjastoja vasten ja emulointirajapinnan toteuttavat kirjastojen tiedostot tulee luonnollisesti toimittaa sovelluksen mukana. Emuloinnilla saattaa olla pieniä merkityksettömiä vaikutuksia sovelluksen suorituskykyyn. GCC:tä on myös mahdollista käyttää hyväksi Mac OS X ympäristössä, jossa se on tarjolla kaikille Mac OS X:n omistajille Xcode tools paketin mukana. Ikkunointikirjastoilla tuotettuja sovelluksia voidaan tämän jälkeen ajaa OS X:ssä käyttäen hyväksi OS X:n X11 emulointia. Myös sovelluksen ajaminen natiivisti, ilman emulointia, on mahdollista käyttäen hyväksi kirjastojen Mac OS X käännöksiä. GTK+:n käännös pohjautuu kokeelliseen Cairoon, joten se ei ole vielä tätä kirjottaessa kokonaan valmis. Sen sijaan Qt ja WxWidgets tarjoavat täysin vakaat käännökset Mac OS X:lle. Qt:a ja wxWidgetsiä voidaan siis käyttää ilman erikoisempia ongelmia kaikilla kolmella alustalla GCC:ä kääntäjänä käytettäessä. Virallinen GTK+ on sen sijaan toistaiseksi rajoittunut Mac OS X:ssä ja sitä joudutaan käytännössä ajamaan erittäin raskaan X11 emuloinnin kautta. Suosittua Eclipse IDE:ä käytettäessä, Qt tarjoaa valmiin lisäkkeen ja integroinnin suoraan Eclipsen käyttöliittymään. Eclipseen voidaan lisätä tuki GCC kääntäjälle, jolloin WxWidgetsin ja GTK+ lisääminen Eclipseen on myös hyvin yksinkertaista. WxWidgetsille voidaan käyttää valmista projektipohjaa, joka toimitetaan kirjaston mukana. GTK+:lle ei toimitettu valmista projektipohjaa, mutta sen sijaan ohjeita projektin asetusten tekemiseen on runsaasti tarjolla kolmannen osapuolen lähteistä. Valmista integraatiota ei kuitenkaan kummallekaan ikkunointikirjastolle ole olemassa Qt:n tapaan.

Windows-alustalla erittäin suositun Microsoftin Visual Studion käyttäminen on myöskin mahdollista kaikkien ikkunointikirjastojen kanssa. GTK+-sovelluksen kehittäminen edellyttää eniten käsityötä. Sovellus tulee luoda konsolisovelluksena ja Gtkmm:n mukana toimitettavat erityisasetuksia sisältävät tiedostot tulee liittää Visual Studioon. Tämän lisäksi tulee tehdä joitakin muutoksia linkkerin ja kääntäjän asetuksiin, sillä valmiina tulevat asetukset eivät GTK+:n kanssa suoraan toimi. Qt tarjoaa valmiiksi asennettavan lisäkkeen visual studioon, joten sen käyttäminen on vaihtoehdoista

selkeästi yksinkertaisinta. Valmis lisäke on kuitenkin tarjolla ainoastaan kaupalliseen Qt:n versioon. Avoimen lähdekoodin versio on tehty helpoksi kääntää avoimen lähdekoodin kääntäjillä, sen sijaan Visual Studion käyttäminen avoimen lähdekoodin Qt version kanssa vaatii erittäin useiden asetusten muuttamista. Avoimen lähdekoodin kehittäminen Qt:n avulla onkin yksinkertaisinta Linux ympäristössä. WxWidgetsin käyttäminen Visual Studiosta käsin on aivan yhtä yksinkertaista kuin Qt:kin kanssa, mikäli käytetään kolmannen osapuolen toimittamia liitännäisiä. WxWidgetsin vakiokokoonpanoa käytettäessä projektiasetuksia joudutaan useasta kohtaa muuttamaan käsin ja tämän lisäksi joudutaan muokkaaman tekstieditorilla omaan projektiin soveltuva projektitiedosto.

Toisin kuin kaksi muuta vaihtoehtoa, Qt käyttää kääntäjän mukana tulevan esikääntäjän lisäksi omaa esikääntääjäänsä. Kaikki Qt makroja sisältävät luokat käsitellään Qt:n oman MOC–esikääntäjän toimesta. Tämä antaa toisaalta vapautta Qt-lähdekoodin syntaksille, mutta toisaalta se hidastaa Qt-sovelluksen kääntämistä jonkin verran. Johtuen esikääntäjän luomista ylimääräisistä tiedostoista ja lähdekoodista samalla kääntäjällä käännettäessä tyypillisen Qt–sovelluksen kääntäminen kestääkin jonkin verran kauemmin kuin vastaavan sovelluksen kääntäminen wxWidgetsillä tai GTK:lla.

7.3 Ei näkyvät komponentit

Kirjastojen sisältämien tietosäiliöiden toteutukset ovat hyvin lähellä toisiaan, kaikissa on nähtävissä sukulaisuus STL:n vastaavaan toteutukseen. Vaikka Qt:n tietosäiliöt muistuttavatkin STL:n vastaavia Qt toteuttaa ne itse. Säiliöiden tarvitema lähdekoodi sisällytetään käyttäjän sovellukseen Qt:n MOC-kääntäjän käsitellessä sovelluksen luokkia.WxWidgets erottuu eniten muusta joukosta. WxWidgetsiä käytettäessä suunnittelija määrittelee itse määriteltyjä olioita sisältävien tietosäiliöiden tietotyypit käännöksenaikaisesti makrojen avustuksella. Menetelmä lisää ohjelmoijan työtä, sillä määrittely on selkeästi uuden tietotyypin lisäämistä pidempi toimenpide. Makrojen käyttö saattaa joidenkin kääntäjien osalta nopeuttaa säiliöiden lähdekoodiin kääntämiseen kuluvaa aikaa. Ja koska makrot ovat varsin kiinteä ja yksiselitteinen osa

C-standardia kirjaston kääntäjätuki on tietosäiliöiden osalta erinomaisen hyvä. Toisaalta makrojen lisäämä lähdekoodi saattaa olla ongelmallista. Jos kehittäjä esimerkiksi lisää makron väärään tiedostoon saattaa seurauksena olla hankalasti selvitettäviä linkkerivirheitä. Mikäli käytettävä kääntäjä ei tue MOC:n lisäämää lähdekoodia, saattaa seurauksena olla erittäin hankalasti selvitettäviä virheitä.

Johtuen UTF-8 enkoodauksen hallitsevasta asemasta avoimen lähdekoodin maailmassa kansainvälinen merkistö on kaikissa kirjastoissa tuettuna. Merkistön käytettävyydessä on kuitenkin eroja, Qt ja GTK kytkevät UTF-8 enkoodauksen automaattisesti päälle, kun sen sijaan wxWidgets vaatii erillisen UTF-8 käännöksen kirjastosta ja erillisten tekstinkäsittely-makrojen käytön monikielistä sovellusta kirjoitettaessa.

Qt on omaksunut kirjastoista voimakkaimmin sovelluskehyksen roolin, se sisältää kirjastoista eniten muuhun kuin pelkkiin piirtorutiineihin keskittyvää toiminnallisuutta. Se sisältää esimerkiksi hyvin laajat rajapinnat verkon tai tietokantojen käyttämiseen. WxWidgets on lähtenyt samaan suuntaan, mutta on ainakin tätä työtä kirjoitettaessa hieman Qt:a jäljessä. GTK+ on valinnut muista poikkeavan linjan ja katsoo tehtäväkseen olla pääasiassa ikkunointikomponentteja sisällään pitävä kirjasto. Kirjaston laaja rooli sovelluskehyksenä parantaa kirjastoa käyttävän sovelluksen ylläpidettävyyttä ja siirrettävyyttä, mitä vähemmän sovelluksella on riippuvuuksia ulkoisiin lähteisiin, sitä vähemmän sovellusta ylläpidettäessä tarvitsee huolehtia niiden välisistä epäyhteensopivuuksista ja toimimattomuuksista eri alustoilla. Toisaalta laajan määrän toiminnallisuuksia sisältävä kirjasto saattaa nostaa linkittyvän sovelluksen kokoa tarpeettomasti, ainakin jos ikkunointikirjasto ei ole tarpeeksi pienissä ja eroteltavissa osissa. Mikäli sovelluksen pieniä osasia on tarkoitus siirtää toisiin sovelluksiin tai sovellus on erittäin rajattu käyttämiensä kirjastokutsujen osalta, riippuvuus yhdestä laajasta sovelluskehyksestä voi olla ongelmaksi. Ikkunointikirjastoa tai omaa lähdekoodia ei voida tällöin pilkkoa riittävän pieniin osiin ja sovelluksen mukana joudutaan kuljettamaan tarpeetonta lähdekoodia.

7.4 Käyttöliittymäkomponentit ja niiden hallinta

Käyttöliittymäkomponenttien toteutus nojaa kaikissa kirjastoissa yhteen perusluokkaan, josta kaikki käyttöliittymäkomponentit on peritty. Tällainen laajentaminen on oleellista, jotta kirjaston sisällä voitaisiin hallita komponentteja yhteisillä osoittimilla. Yhteisen kantaluokan ansiosta mahdollistuu myös komponttien automaattinen poisto, esimerkiksi pohjapiirustusluokan poiston yhteydessä voidaan poistaa kaikki sen sisällään pitämät komponentit. Yhteisen kantaluokan käytössä on eroa kirjastojen välillä. Qt ja GTK+ ovat valinneet standardin C++:n RTTI toteutuksen ja suosittelevat käyttämään C++:n vakiotyyppisiä virtuaalisia funktioita. Sen sijaan wxWidgets on toteuttanut RTTI:n kokonaan uudestaan oman tyyppisenä toteutuksenaan. WxWidgetsin toteuttajat ovat katsoneet C++:n tavan kätkeä perityssä luokissa ylikirjoitetut julkiset virtuaaliset aliohjelmajäsenet ongelmalliseksi. He ovat siirtäneet mahdolliset ylikirjoitettavat virtuaaliset funktiot suojattuun(eng. protected) näkyvyyteen ja ohjeistavat kirjaston käyttäjää ylikirjoittamaan perityissä luokissa näitä Do-alkuisia aliohjelmajäseniä. Kokemattoman tai C++:n tottumattoman suunnittelijan kannalta wxWidgetsin tässä suhteessa erilainen toimintapa on erittäin turvallinen. Suunnittelija välttää varsin yleisen ja hankalasti selvitettävän sudenkuopan. Oikean metodin sitominen ajonaikaisesti toteutukseen saattaa esimerkiksi johtaa tilanteeseen jossa kutsutaan päällepäin oikein toimivaa toteutusta, mutta joka erikoistilanteissa toimii täysin virheellisellä tavalla. Kokeneen C++ suunnittelijan kannalta toteutustapa saattaa olla omituinen, sillä se vaatii kokonaan uuden tekniikan opettelun polymorfismia käytettäessä. Tästä saattaa myös, kääntäjästä riippuen, olla haittaa sovelluksen suorituskyvylle ja muistinkulutukselle.

Näkökulma komponenttien hallintaan on hyvin samanlainen eri kirjastojen välillä. GTK+ ja wxWidgets sisältävät täsmälleen samanlaisen rakenteen komponenttien hallintaan. Kummatkin pitävät sisällään erilliset luokat komponenttien hallintaan, jotka sitten omistavat hallitsemansa luokat. Qt on toteutukseltaan kohtuullisen lähellä kahta edellä mainittua, mutta vie hierarkian käytön vielä edellämainittuja kirjastoja pidemmälle. Sen sisältämä komponenttien hallintasuhde on vapaampi, mutta toisaalta myös muita laajempi. Qt:n kaikista sovelluksen komponenteista rakentama hierarkia

tarjoaa mahdollisuuksia erittäin tehokkaisiin rakenteisiin, tästä esimerkkinä tyylitiedostojen käyttö Qt:ssa. Koska kaikki sovelluksen komponentit ovat liitoksissa hierarkian kautta keskenään, voidaan koko sovelluksen ulkoasu muuttaa vain asettamalla sovellukselle erilainen tyylitiedosto. Hierarkian laajamittainen käyttö esimerkiksi yhteiseen lapsien hallintaan tai viestimiseen muiden soveluskomponenttien kanssa on varsin yleistä Qt-sovellusten lähdekoodia lukiessa.

Kirjastojen sisältämä näkemys ikkunointikirjaston toteuttamasta toiminnallisuudesta on melko erilainen. Ikkunoinnista ja graafisesta ulkoasusta puhuttaessa WxWidgets sisältää kirjastoista eniten valmiita kokonaisia toiminnallisuuksia. Toisin sanoen wxWidgets toteuttaa erikoistuneempia asioita valmiiksi kirjaston toimesta, kun taas Gtk+ ja Qt toteuttavat vähemmän tiettyyn tarkoitukseen erikoistuneita komponentteja. Kummassakin mallissa on puolensa. Isomman määrän valmiita korkean tason komponentteja sisältävä kirjasto nostaa ohjelmoijan tehokkuutta, kun useimmat asiat on toteutettu valmiiksi. Iso määrä korkean tason komponentteja, kuitenkin nostaa kirjaston kokoa siihen linkittäytyessä ja näin ollen hidastaa sovelluksen suorituskykyä ja toimitettavan sovelluspaketin kokoa. Internetin aikakaudella ison sovelluspaketin toimitus saattaa muodostua ongelmalliseksi, erityisesti jos sovellus on suosittu ja palvelinkapasiteetti rajoitettua. Mikäli tilaa on säästetty matalamman tason komponenttien määrän kustannuksella, saattaa tulla ongelmia voimakkaasti personoitua sovellusta kehitettäessä. Kirjasto ikään kuin sanelee sovelluksen mahdollisen ulkonäön ja pakottaa sovelluskehittäjän käyttämään runsaasti aikaa piirtorutiinien ja sovelluksen ulkomuodon personoinnin kanssa.

7.5 Asynkronisten viestien toteutus

Syntaksin kyky välttää virheitä ennalta on eräs keskeinen tekijä asynkronisten viestien käytettävyyttä mietittäessä. Asynkroniset tapahtumat ovat luonteeltaan ajonaikaisia jo pelkästään tapahtumien satunnaisen laukaisemisen takia. C ja C++ tarkistavat sovelluksen oikeellisuuden ainoastaan käännöksenaikaisesti, tästä johtuen jokainen C-ohjelmoija varmasti tietää kuinka vaikeaa on ajonaikaisten tapahtumien ja niistä

seuraavien virheiden korjaaminen on. Mikäli ikkunointikirjasto mahdollistaa virhealttiiden tapahtumien käytön saattaa seurauksena olla hankalasti käsiteltäviä ongelmia. Vaikka kyseessä ovat ikkunointikirjastot, pitää ottaa huomioon että tapahtumat moderneissa sovelluksissa eivät läheskään aina ole seurausta käyttäjän toimista. Esimerkiksi erilaiset laitetason rajapinnat saattavat ilmoittaa toiminnastaan asynkronisin viestein. Mikäli näiden rajapintojen käsittelyyn jää ongelmia, saattaa niistä seurata erittäin hankalasti ratkaistavia ongelmia.

Qt käsittelee signaalit käyttäen omaa C++ standardista poikkeavaa syntaksiaan. Kielen laajentaminen Qt:n tavoin saattaa vaikeuttaa ohjelmakoodin lukemista, erityisesti niiden osalta jotka eivät Qt:a tunne. Qt:n oma MOC-kääntäjä ilmoittaa käännöksenaikaiset virheet etukäteen. Signaalien käyttö on Qt:ssa viety selkeästi muita kirjastoja pidemmälle ja uusien signaalien määrittely on todella yksinkertaista ja myös turvallista. Signaalit käyttävät hyväkseen Qt:n sisäistä introspektiota ja näin ollen signaalien hyötykuorman täytyy välittää, joko yksinkertaisia tietotyyppejä tai Q_OBJECT makron sisältäviä Qt:n omia objekteja. Tästä seuraa selkeä heikkous signaalien käsittelyssä. Muut kuin Qt-luokat täytyy erikseen rekisteröidä sovelluksen käyttöön ja näin ollen muut kuin tietotyypin rekisteröivä säie eivät kykene tietotyyppiä vastaanottamaan. Tästä seuraa yksi yleinen Qt-sovelluksen kaatumiseen johtava syy.

GTK+ on voimakkaasti C-pohjainen kirjasto, sen asynkronisten viestien käsittelyyn sopiva syntaksi nojaa laajoilta osin osoittimien ja takaisinkutsu-funktioiden käyttöön. Perinteisen hyväksi havaitun menetelmän käyttäminen tarjoaa C-ohjelmoijalle tutun ympäristön ohjelmointiin. Kirjasto on kuitenkin kaikkein virhealttein esitellyistä kirjastoista. Kirjastoa käytettäessä virheet jotka Qt:n MOC-kääntäjä tai wxWidgetsin tapauksessa esikääntäjä havaitsee käännösvaiheessa vasta kun, jokin kirjaston funktioista palauttaa virhekoodin. Kirjasto antaa osoittimen tapahtuman lähettäneeseen komponenttiin suoraan käyttäjän tapahtumankäsittelyfunktioon ja jättää näin ollen kirjaston käyttäjän vastuulle tarjoamiensa osoittimien oikeellisuuden tarkistamisen. Myös useampien säikeiden tapauksessa vastuu tiedon jakamisesta jätetään kirjaston käyttäjän vastuulle. Käyttäjän tulee huolehtia etteivät säikeet pääse keskenään aiheuttamaan ongelmatilanteita, poissulkevia tilanteita ja niin edelleen. Gtkmm

laajentaa suuresti GTK+:n toimintaa ja tarjoaa huomattavasti enemmän C++ mallin mukaisen näkökulman GTK+ ikkunointiin. Se tarjoaa turvallisemman ratkaisun erityisesti GTK+:n sisäisten komponenttien tuottamien asynkronisten tapahtumien käsittelyyn. Sen syntaksi on kuitenkin huomattavasti Qt:ta vaikeaselkoisempaa ja virhealttiimpaa. C++:n funktio-osoittimia runsaasti sisältävän syntaksin lukeminen on vaikeaselkoista, lisäksi kirjaston käyttäjän tulee seurata jonkinlaista nimeämiskäytäntöä, jotta signaalinkäsittelijät eroaisivat muista funktioista. Koska GTK+ ja GTKmm käyttävät standardeja C-osoittimia ei niillä ole ongelmia komponenttien rekisteröinnin kanssa. Kaikkia mahdollisia tietorakenteita voidaan välittää parametrina ja kuljettaa säikeiden välillä.

WxWidgetsin tapa käsitellä tapahtumia on esitellyistä kirjastoista kaikkein eniten C++-standardin mukainen. Kirjasto ei määrittele omia varattuja sanoja, vaan hallitsee kaiken makroilla ja tavanomaisilla funktiokutsuilla. Käännöksenaikaisesti määritellyt tapahtumat ovat turvallisia käännöksenaikaisen määrittelynsä takia. Huonona puolena on että WxWidgetsin makrot antavat joillakin kääntäjillä hankalasti selvitettäviä virheilmoituksia kirjaston sisältä. Tässä tilanteessa varsinkin Qt:a paremmin dokumentoitu ja kommentoitu kirjaston sisäinen lähdekoodi auttaa. Ajonaikaisten tapahtumien syntaksi on helppolukuinen, erityisesti verrattuna Gtkmm:n vastaavaan. WxWidgetsin tapa käyttää numerovakioita viestien identifiointiin on varmasti tehokas numerojen nopean käsittelyn takia, mutta saattaa sovelluksen virheitä etsiessä muodostua ongelmalliseksi. Pelkkä numero saattaa olla ongelmallinen, koska se ei välttämättä identifioi viestiä yksiselitteisesti. Eri viestien nimiin liittyvä numero saattaa nimittäin olla sama, jos viestiä vain käytetään eri kontekstissa. Tämä on ongelma muihin kirjastoihin verrattuna sillä Qt:n viesti identifioidaan viestin nimen perusteella ja GTK+:n viesti identifioidaan viestin lähettävän komponentin muistiosoitteen avulla. Muistiosoite ja nimi on helppo liittää virheitä korjatessa sovelluksen tiettyyn lähdekoodin osaan, mutta pelkkä numero ei itsessään kerro kehittäjälle yhtään mitään. Toisin kuin Qt ja GTK+ wxWidgets ei erillisiä säikeitä käytettäessä aiheuta ongelmia. Kirjasto nojaa muita enemmän kopiointiin ja säie-turvallisiin tapahtumajonoihin, näin ollen jaetun datan ongelma pääsee muodostumaan useimmiten vain wxWidgetsistä riippumattomissa tapauksissa.

7.6 Omat muokatut komponentit

Qt ja GTKmm käyttävät hyväkseen C++ standardin mukaista komponenttien laajentamista. Standardin mukainen laajentaminen tekee kirjaston käytön oppimisesta nopeaa, helpottaa lähdekoodin lukemista ja vähentää kehittäjälle pakollista kirjastokohtaisten toimintatapojen opiskelua. Standardin mukaisessa mallissa on kuitenkin omat ongelmansa: Tyypillisessä esimerkissä sovelluskehittäjä kutsuu erehdyksessä väärää tietotyyppiä väärässä kontekstissa ja tällä toimellaan aiheuttaa virheellisen muistiviittauksen ja kaataa koko sovelluksen. WxWidgets pyrkii toteutuksellaan suojelemaan kehittäjää edellisen kaltaiselta virheeltä. Ideaalitapauksessa ajonaikainen virhe muuttuu käännöksenaikaiseksi ja tulee hoidettua ilman mahdollista lopputuotteeseen jäävää virhettä. GTK+:n tapa laajentaa olemassaolevia komponentteja ei ole suoraan vertailukelpoinen muiden kirjastojen kanssa, koska käyttää hyväkseen C-kielellä toteutettua oliomallia. GTK+:n käyttämästä mallista voidaan kuitenkin sanoa että se on sisältämänsä runsaan funktio-osoitinmäärän takia huomattavasti muita kirjastoja turvattomampi käyttää.

Kaikki kirjastot tarjoavat toisiaan muistuttavat tyhjät kontrollipohjat, joiden päälle voidaan rakentaa omia kontrolleja. Gtk+ ja Gtkmm:n mukana toimitettava Cairo on erittäin suosittu matalan tason piirtokirjasto. Sen tunnettuus ja suosio auttaakin suuresti muokattuja komponentteja kehitettäessä. Esimerkkejä piirto-operaatioiden käytöstä on tarjolla runsaasti Internetissä. Sen sijaan wxWidgetsin ja Qt:n komponentteja piirtämiseen käytettäessä joudutaan nojaamaan täysin kirjastojen omaan dokumentaatioon.

7.7 Kielenkäännös ja suunnittelutyökalut

wxWidgetsin ja GTK+:n kielenkäännösten toteutuksessa ei ole merkittävää eroa. WxWidgetsin toteuttama wxLocale rajapinta tekee kuitenkin selkeän eron GTK+:n

hyvin proseduraaliseen toteutustapaan. WxWidgetsin wxLocalen ansiosta sovelluksen käännöstä on mahdollista hallita oliomallisesti ja käännösolion toimintaa voidaan laajentaa perinnän avulla. Näin käännöksen toiminnallisuutta voidaan hallita ja muuttaa huomattavasti yksinkertaisemmin kuin GTK+:ssa, jossa käännös on sidottu makroihin sisältyvien tekstipätkien korvaamiseen toisilla.

Qt:n näkökulma käännösrajapintaan on vielä enemmän oliopohjainen kuin wxWidgets vastaava. Yhden käännökseen keskittyneen olion sijasta jokainen Qt:n olio pitää sisällään instrospektion, johon myös käännösfunktiot sisältyvät. Toisin kuin muut ikkunointikirjastot, Qt tuottaa automaattisesti tiedon käännökseen käännettävän tekstin sijainnista oliohierarkiassa. Näin käännöstä sovellukseen siirtävän kehittäjän on helpompi löytää käännettävä teksti sovelluskoodista, hänen ei välttämättä tarvitse lisätä tarkkoja kommentteja sovelluksensa jokaiseen tekstipätkään ja hän voi tarvittaessa helpommin keskustella tekstipätkän kontekstista kääntäjän kanssa. Koska sovelluksen itsensä oliomalli pitää sisällään käännöksen, on käännös vielä enemmän riippumatonta ulkoisista lähteistä kuin GTK+:n ja wxWidgetsin vastaavissa ratkaisuissa. Qt:lla sovellusta kirjoittavan kehittäjän ei tarvitse erikseen miettiä käännöksen suorittavan kirjaston olemassaoloa tai sen versiotietoja sovellusta kehittäessään. Sovelluksesta tulee näin ollen riippumattomampi kohdejärjestelmästä ja sen asetuksista.

GTK+:n ja wxWidgetsin käyttämä gettext ja libIntl on Unix pohjaisissa järjestelmissä käytännön standardi. Näin ollen sovelluksen mukana ei tarvitse toimittaa ollenkaan käännöksen toteuttamiseen liittyvää toiminnallisuutta ja sovellus kuluttaa vähemmän resursseja kun käännöstoiminnallisuus ei ole osa sovellusta. Mikäli sovelluksen käännöstä suoritetaan avoimen lähdekoodin periaatteiden mukaisesti osittain tai kokonaan sovelluksen käyttäjien toimesta, gettext on ylivoimainen ratkaisu. Gettext käännöstä tuottavia työkaluja on saatavilla useita ja ne ,kuten gettext XML-formaattikin, ovat hyvin todennäköisesti tuttuja sovellusten käännöksiä kirjoittaville harrastajille.

7.8 Piirtorutiinit

Kirjastojen lähtökohdat piirtorutiineiden toteutukseen ovat hyvin erilaiset. Qt:n käyttämä hyvin matalan tason rajapintojen käyttäminen samanlaiseen alustan mukaiseen piirtämiseen on kätetyistä piirtotavoista selkeästi työläin toteutettava. Se kuitenkin tarjoaa suurimman mahdollisen mukautettavuuden. Kirjasto kykenee tarjoamaan hyvin mukautettavan käyttöliittymän samanaikaisesti käyttöjärjestelmäkohtaisen yhteenkuuluvuuden kanssa. Koska Qt:n toimintatapa uudelleenkirjoittaa paljon käyttöjärjestelmän omaa toiminnallisuutta Qt:n siirtäminen uusille alustoille lienee suhteellisen hidasta. GTK+:n ja wxWidgetsin verrattuna Qt:n ongelmana saattaa olla laajan kokonaisuuden liittäminen yhteen kirjastoon. Koska Qt ei ulospäin selkeästi erota piirtorajapintaa komponenttien toteutuksesta, ongelmana saattaa olla muuttuvat piirtorajapinnat ja tästä johtuen kirjaston oma ylläpidettävyys.

GTK+:n käyttämä Cairo kirjasto on erittäin tunnettu avoimen lähdekoodin yhteisössä. Esimerkiksi laajalti tunnettu Mozilla säätiö käyttää sitä hyväkseen. Yleisesti tunnettu piirtorajapinta on suuri etu GTK+:lle. Se tarjoaa samanaikaisesti piirtorajapinnan laadukkuuden ja uusien kehittäjien saatavuuden GTK+ kirjaston kehittäjäjoukkoon. Myös siirrettävyys paranee. Aina kun Cairo käännetään uudelle alustalle, GTK+:n kääntäminen samalle alustalle tulee erittäin helpoksi. Cairon käytön ongelmana Qt:n verrattuna on yleiskäyttöisyyden asettamat rajoitteet. Kun alla oleva rajapinta pyrkii olemaan samalla käytettävä kaikkeen muuhunkin, kuin vain ikkunakomponenttien piirtämiseen, tulee helposti esille ristiriitaisia vaatimuksia. Myös uusien ominaisuuksien lisääminen vaatii aina paljon kommunikointia osapuolten välillä, kun osapuolet eivät ikään kuin ole ”saman katon alla”.

WxWidgets on itseasiassa vain määritelmä ikkunointikirjastolle, kukin käännös tarjoaa alustakohtaisen toteutuksen tälle määritelmälle. WxWidgetsin vahvuus on valinnan vapaudessa. Mikäli käytössä oleva käännös ei miellytä, sovelluksen voi aina kääntää uudelleen eri käännöksen avulla. Muista kirjastoista poiketen WxWidgetsin toteutus ei riipu lainkaan tietystä piirtorajapinnasta. Tämä on suuri etu sovellusta erilaisiin

ympäristöihin siirrettäessä. Varsin hankalaa matalan tason toteutusta ei ole pakko toteuttaa lainkaan uudelle alustalle. Voidaan käyttää kyseisen alustan parhaiten wxWidgetsille soveltuvaa rajapintaa tai vaikkapa toista monialustaista kirjastoa. Erityisesti korkean tason käännöksistä voi koitua myös ongelmia kehittäjälle. Vaikka eri käännösten toteuttama rajapinta on sama, ei voida millään taata täydellistä toiminnallista vastaavuutta käännösten välille. Kahden eri käännöksen hyväkseen käyttämien kirjastojen näkemys sovelluksen ulkonäöstä ja vuorovaikutteisuudesta saattavat olla täysin toisistaan poikkeavia. Tästä johtuen sama sovellus saattaa toimia täysin eri tavoin erilaisilla alustoilla.

8. ANALYYSITULOKSET JA POHDINTA

Tässä kappaleessa esitellään tutkimuksen tuloksia, kerrotaan miten tutkimus tehtiin ja käsitellään analyysituloksien merkitystä ja niiden tulkintaa.

8.1 Käännetyn lähdekoodin viemä tila moduleittain eri käyttöjärjestelmissä

Liitteessä 1 on listattu kaikkien kolmen kirjaston käännettyjen lähdekoodien viemä tila Windowsissa, liitteessä 2 Linuxissa ja liitteessä 3 Mac OS X:ssa. Windowsissa käännös tehtiin Visual Studio 2008:lla, Linuxissa ja Mac OS X:ssa GCC 4.4:lla. Mac OS X jouduttiin ottamaan huomioon käyttöjärjestelmäkohtainen rajoitus. Mac OS X laskee virheellisesti yhden kilotavun olevan 1000 tavua, tästä johtuen käännöksen tulos mitattiin käyttöjärjestelmän ilmoittamasta arvosta tavuina ja jaettiin se 1024:lla koon saamiseksi kilotavuina.

Liitteiden 1, 2 ja 3 tuloksista saadaan yhteen laskemalla kaikkien kirjaston osien viemä tila eri käyttöjärjestelmissä. Tästä voidaan päätellä että käyttöjärjestelmästä riippumatta käännetyn kokonaislähdekoodin viemä tila on jakautunut niin että wxWidgets kuluttaa tilaa vähiten ja tämän jälkeen järjestyksessä tulevat GTK+/Gtkmm ja Qt. Yhtenä syynä tähän on arkkitehtuurien erilaisuus. Qt:ssa ajatuksena on toteuttaa käyttöjärjestelmäkohtainen toiminnallisuus käyttäen kullakin alustalla mahdollisimman matalan abstraktiotason rajapintoja. Tästä seuraa sekä hyötyjä että haittoja. Matala abstraktiotaso tarkoittaa suurta joustavuutta toteutuksessa, se mahdollistaa samankaltaisen toiminnallisuuden toteuttamisen eri alustoilla. Matala abstraktiotaso tarkoittaa myös suurempaa lähdekoodin määrää, koska samalla koodirivillä voidaan tehdä vähemmän. Suuremmasta lähdekoodin määrästä seuraa luonnollisesti myös suurempi käännetyn lähdekoodin viemä tila.

Liitteistä 1-3 on myöskin nähtävissä että GTK+ tarvitsee toteutukseensa paljon ulkoisia kirjastoja. GTK+:n suuresta riippuvuuksien määrästä voidaan päätellä että ajatuksena on rakentaa toiminnallisuutta olemassa olevien komponenttien päälle. Tämä mahdollistaa abstraktiotason nostamisen eri kirjaston osissa. Osa komponenteista, esimerkiksi Cairo, rakentuu hyvinkin matalan abstraktiotason lähdekoodin ympärille. Käännetyn lähdekoodin määrä ei niissä kuitenkaan muodostu Qt:a vastaavaksi, pienempi käyttötarkoitus rajaa niiden lähdekoodin määrää. WxWidgetsin arkkitehtuuri ei rajoita alla olevan käyttöjärjestelmäkohtaisen toiminnallisuuden abstraktiotasoa millään tavalla. Käytännössä kuitenkin wxWidgetsin kaikki toteutukset on tehty toisen ikkunointikirjaston toiminnallisuuden pohjalle. Tämä vähentää huomattavasti tarvittavan lähdekoodin määrää. Ongelmakohtana luonnollisesti on käyttöliittymän samankaltaisuus erilaisten käyttöjärjestelmien ja kirjastojen välillä.

Liitteiden 1-3 tuloksista nähdään että Qt:ssa muuhun toiminnallisuuteen nähden suhteellisen pienen kirjaston ytimen ympärille on rakennettu suurempia käyttötarkoituksen mukaan jaoteltuja moduuleja. Koska Qt on ikkunointikirjasto, merkittävä osa Qt:n lähdekoodista keskittyy graafisen käyttöliittymän tarjoavaan moduliin. Tämän lisäksi kokonaista internet-selainta vastaavan toiminnallisuuden tarjoava moduli on suhteellisen suuri. Muuta toiminnallisuutta käsittelevät modulit ovat suhteellisen pieniä ja käyttötarkoitukseltaan tarkemmin rajattuja. Toisin kuin Qt, GTK+ sisältää suhteellisen suuren yhteisen ytimen. Lisäksi C++ käyttöön tarvittavat rajapinnat vievät kohtuullisen paljon tilaa. GTK+:n käyttämät riippuvuudet ovat pieniä. WxWidgets rakentuu suhteellisesti suuren ytimen ympärille, joka on kuitenkin huomattavasti Qt:n ydintä pienempi. WxWidgetsin ytimen jälkeen suhteellisesti suurin osa on pääasiallisen graafisen toiminnallisuuden sisältävä moduuli, muut moduulit ovat erittäin pieniä.

Qt:n pienikokoinen ydin mahdollistaa kohtuullisen pienikokoisten ei graafisten sovellusten luomisen. Graafisen toiminnallisuuden tai web-sisällön lisääminen Qt-sovellukseen kasvattaa sovelluksen kokoa suhteellisen paljon. Muiden modulien osalta Qt on samalla tasolla muiden kirjastojen kanssa. GTK+:n tapauksessa tilanne on erilainen. Suhteellisen suuren ytimen ympärille on mahdollista rakentaa runsaasta

määrästä erilaisia pieniä kirjastoja. Koska ylimääräiset kirjastot ovat suhteellisen yleiskäyttöisiä, ne mahdollistavat suuren valinnan vapauden erilaisten vaihtoehtojen väliltä. Yleiskäyttöisyyden ansiosta on mahdollista jättää koko GTK+:n liittämättä sovellukseen. C++-rajapintojen suuri koko on ongelma ja tekee GTKmm-sovelluksesta helposti melko suuren. WxWidgets on käännetyn lähdekoodin osalta erittäin pieni. Sovelluksen mukana toimitettavien kirjastojen vaatima tila tuskin nousee ongelmaksi useimmissa tapauksissa. Joskin wxWidgetsin ytimen suhteellisesti suurehko koko asettaa suhteellisen korkean rajan minimaaliselle sovelluksen koolle.

Linuxissa ja Windowsissa Qt:n käännetty lähdekoodi pysyy muihin kirjastoihin nähden suhteellisesti saman kokoisena. Qt:n Mac OS X toteutus on suhteellisesti huomattavasti suuremman kokoinen kuin muiden kirjastojen. GTK+:n C-pohjaisten kirjastojen koko on suhteellisen suuri Windowsissa, todennäköisesti johtuen Visual studion C-kääntäjän erilaisesta toiminnasta. C++ pohjainen Gtkmm lähdekoodi on suhteellisesti saman kokoista eri käyttöjärjestelmien välillä. WxWidgets on suhteellisesti saman kokoinen Windowsilla ja Mac OS X:lla. Linuxissa wxWidgetsin käännetty koko kasvaa huomattavasti.

8.2 Yhteensopivat käännösympäristöt

Kirjastojen ilmoittama kääntäjäyhteensopivuus varmistettiin kääntämällä kirjastojen lähdekoodi alla olevilla kääntäjillä. Valitettavasti Borland C++ kääntäjää ei ollut saatavilla sen vaatiman kaupallisen lisenssien osalta. Borlandin osalta oletettiin kirjastojen oman ilmoituksen yhteensopivuudesta pitävän paikkansa.

Taulukosta 9 on nähtävissä että kaikki kääntäjät ovat yhteensopivia GCC:n kanssa. Koska kyseessä ovat avoimen lähdekoodin kirjastot ei liene yllätys että ne ovat yhteensopivia juuri avoimen lähdekoodin GCC:n kääntäjän kanssa. Erityisesti Linux ja Unix maailmaan syntyneiden GTK+:n ja Qt:n tapauksessa GCC on varmasti ollut se ensimmäinen kääntäjä, jolla kirjastot on käännetty. Tuki Windowsin Visual Studio kääntäjälle on varmasti alunperin ennemmin wxWidgetsin ominaisuus, wxWidgets jopa

toimittaa lähdekoodiensa mukana Visual Studion projektitiedostot valmiiksi. GTK+:n ja Qt:n tapauksessa projektitiedostot joutuu itse luomaan ohjeiden avustuksella. Ilmoitettu tuki Borlandin kääntäjälle on ainoastaan wxWidgetsin ominaisuus. Tuen salaisuus muihin kirjastoihin verrattuna lienee WxWidgetsin sisältämä hyvin lähellä C++-standardia oleva lähdekoodi.

Taulukko 9: Kirjastojen yhteensopivuus eri käännösympäristöjen kanssa

	Qt	GTK+/Gtkmm	wxWidgets
GCC	x	x	x
Visual studio	x	x(Luvun 5.2 rajoituksin)	x
Borland C++			x(ei voitu varmistaa)

8.3 Ei-näkyvät komponentit

Ei näkyvien komponenttien analyysi tehtiin Littlefairin väitöskirjassaan[2] kehittämällä CCCC-työkalulla. Esiteltävät komponentit valittiin arkkitehtuuria käsittelevissä kappaleissa esitellyistä luokista. Koska kaikki kirjastot olivat toteuttaneet ei-näkyvät komponenttinsa käyttöjärjestelmästä riippumattomalla tavalla, käyttöjärjestelmäkohtaisia toiminnallisuuksia ei tarvinnut tässä tapauksessa tutkia.

Liitteen 4 McCaben metriikan tuloksia esittelevässä taulukossa esitellyt tulokset saatiin syöttämällä CCCC:lle taulukossa mainitut luokat. CCCC:lle asetettiin jokaista luokkaa kohden parametreiksi luokan toteutuskieli. Käsiteltävät tiedostot valittiin tutkimalla lähdekoodista, missä tiedostoissa sijaitsisi oleellinen osa rajapinnan toteutuksesta. Liitteen 4 Chidamber & Kemererin menetelmän tuloksia esittelevässä taulukossa esiteltyjen luokkien oliosuunnittelun laatua tutkittiin analysoimalla CCCC:n avulla koko sitä kirjaston erilliskirjastoksi erotettavissa olevaa osaa joka toteuttaa ei-näkyvien komponenttien toiminnallisuuden. GTK+:n komponenteille ei voitu tehdä kokeita, koska ne seuraavat GTK+:n omaa oliosyntaksia C++ syntaksin sijasta. Liitteen 4 Henry & Kafuran menetelmien tuloksia esittelevän taulukon tulokset saatiin samoin kuin

saman liitteen Chidamberin & Kemererin tuloksia käsittelevän taulukon.

Liitteen 4 sisältämistä McCaben menetelmällä saaduista tutkimustuloksista voitiin todeta että wxWidgetsin näkymättömien komponenttien metodit ovat kaikista yksinkertaisimpia, tämän jälkeen tulee GTK+ ja sitten Qt. Qt:n ongelmaksi, McCaben analyysissä, muodostuu sen merkkijono-luokan algoritminen monimutkaisuus. Luokan monimutkaisuuden todennäköisenä syynä on sen toteuttaman implisiittisen jakamisen mukanaan tuoma monimutkaisuus. Muissa kirjastoissa kuin Qt:ssa ei vastaavaa toiminnallisuutta ole. GTK+:n merkkijono-luokka toteuttaa itse oman versionsa merkkijonojen käsittelystä. WxWidgetsin merkkijonojen toteutus nojaa toteutettavan alustan toteutukseen ja lisää siihen hieman omaa toiminnallisuuttaan. Tästä johtuen wxWidgetsin toteutus on huomattavasti GTK+:n vastaavaa yksinkertaisempi. Assosiatiivisten taulukoiden toteutuksessa Qt on yksinkertaisin, tämän jälkeen tulee GTK+ ja sitten wxWidgets. Yhtenä syynä tähän voidaan pitää wxWidgetsin GTK+ ja Qt:sta poikkeavaa tapaa toteuttaa kopiointi kirjoitettaessa myös säiliöluokille. Eräänä syynä Qt:n yksinkertaisuuteen GTK+:n verrattuna voisi on sen lähdekoodin perustuminen GTK+:n C:n sijasta aidosti oliopohjaiseen C++:n, oliopohjainen kieli mahdollistaa listan taulukon alkioiden käsittelemisen pienemmällä määrällä lähdekoodia. Listojen toteutuksessa tämä on myöskin nähtävissä sillä GTK+ sisältää monimutkaisimman toteutuksen, wxWidgets on hieman GTK+:aa yksinkertaisempi ja Qt on huomattavasti yksinkertaisin.

Liitteen 4 Chidamber & Kemerer:n menetelmän tuloksista voidaan muöskin nähdä että olio-suunnittelun osalta wxWidgetsin toteuttaa normaalit ja assosiatiiviset-listat vähemmän ylläpidettävämmin kuin Qt. Qt:n listat ovat wxWidgetsin listoja paremmin kapseloituja eli sisältävät vähemmän yhteyksiä muihin komponentteihin. Tämän lisäksi ne sisältävät vähemmän painotettuja metodeita ja ovat täten vähemmän sovelluskohtaisia. Merkkijonojen osalta Qt:n merkkijonot sisältävät enemmän sisäistä monimutkaisuutta painotettujen metodien muodossa. WxWidgetsin merkkijonot sisältävät sen sijaan Qt:a enemmän sisäisiä kytköksiä kirjaston muuhun toiminnallisuuteen ja ovat näin vähemmän itsenäisiä. Rakenteellisten metriikoiden osalta Qt:n merkkijonojen toteutus sisältää enemmän ulostulovuota mutta wxWidgetsin

selkeästi enemmän sisääntulovuota. WxWidgetsin selkeä sisääntulovuon osuus kertoo siitä että merkkijonotiedoston koodia käytetään paljon hyväksi muissa wxWidgetsin osissa. Qt:n osalta taas kyseessä on komponentti, joka on suhteellisen itsenäinen ja kapseloitu eikä tarvitse paljoakaan lähdekoodia muista kirjaston osista. Normaalien ja assosiatiivisten listojen osalta wxWidgets ja Qt sisältävät samanlaisen sisääntulovuon, mutta ulostulovuo on wxWidgetsin osalta hieman suurempi. Tästä voidaan päätellä WxWidgetsin listojen olevan Qt:n vastaavaa toteutusta vaikeammin erotettavissa muusta kirjaston lähdekoodista. Ja näin ollen kirjaston lähdekoodin ylläpito muodostuu vaikeammaksi.

8.4 Käyttöliittymäkomponentit ja niiden hallinta

Käyttöliittymäkomponenttien staattiset metriikat laskettiin käyttöliittymäkomponenttien rajapinnan toteuttavista luokista ja metodeista. Analysoitavaksi valittiin arkkitehtuuria käsittelevissä kappaleissa esitellyt luokat. Analyysi tehtiin CCCC:n avustuksella. Liitteen 5 McCaben menetelmän tuloksia käsittelevässä taulukossa esitellyt luokat valittiin niin että samassa tiedostossa sijaitsevat luokat otettiin mukaan tutkimukseen. Lisäksi huomioitiin mahdolliset luokkien yksityiset toteutukset. CCCC:lle syötettiin lisäparametriksi tiedoston toteutuskieli. Koska McCabe ei käsittele tiedoston ulkopuolisia suhteita anayysitehtiin vain taulukossa mainituista tiedostoista. GTK+/Gtkmm tapauksessa otettiin poikkeuksellisesti myös huomioon että yhteisiä kantaluokkia on kaksi. Liitteen 5 Chidamberin ja Kemererin menetelmän tuloksia käsittelevässä taulukossa esiteltyjen luokkien tutkimustulokset saatiin laskemalla CCCC:n avulla luokat sisältävän moduulin kaikki kytkennät ja painotetut metodit. Näin saatiin selville moduulin sisäinen toteutus. Liitteen 5 Henryn & Kafuran menetelmän tuloksia käsittelevän taulukon tulokset saatiin samoin kuin saman liitteen Chidamberin ja Kemererin menetelmiä käsittelevän taulukon.

Liitteestä 5 nähdään että proseduurien osalta monimutkaisimman kantaluokan omistaa GTK+, tämän jälkeen tulevat järjestyksessä wxWidgets, Gtkmm ja Qt. GTK+:n monimutkaisuuden taustalla lienee kantaluokan C-pohjaisuus, olio-toiminnallisuus joka

tulee C++:n mukana on täytynyt toteuttaa kantaluokkaan proseduurien avulla. Tämä sama monimutkaisuus heijastuu sitten Gtkmm toteutukseen, joka on vain yhteensopivuusrajapinta C-toteutukseen. WxWidgetsin ja Qt:n välille ero voisi syntyä esimerkiksi erilaisista luokkarakenteista. Kun katsotaan luokkien oliosuunnitteluun liittyviä tuloksia ero on helpompi ymmärtää, Qt:n kantaluokka toteuttaa isomman osan toiminnallisuudesta muualla kuin kantaluokassaan, tämä on nähtävissä esimerkiksi siitä että Qt:n kantaluokalla on wxWidgetsiä suurempi moduulin kytkentä ja pienempi painotettujen metodien määrä. Oliosuunnittelun tuloksista voidaan myöskin päätellä Gtkmm:n kahden kantaluokan olevan yhteen laskettuna suhteellisesti monimutkaisempi kuin kahden muun kirjaston kantaluokkien, syynä tähän lienee jo aiemmin mainittu sidonta C-pohjaiseen toiminnallisuuteen. Myöskin korkea modulin kytkennän määrä kertoo että Gtkmm:n on muitten kirjastojen kantaluokkia huomattavasti monimutkaisempi. Rakenteelliset metriikat paljastavat Qt:n kantaluokan nojaavan voimakkaasti ulkopuoliseen lähdekoodiin, mutta muiden luokkien käyttävän hyvän vähän Qt:n kantaluokan lähdekoodia. Tästä voidaan päätellä luokan sisältämän lähdekoodin aiempiin metriikkoihin perustuen kantaluokan olevan yksinkertaisen pitkälti muiden luokkien lähdekoodin varassa. WxWidgetsin kantaluokka on muita kantaluokkia itsellisempi eikä näin ollen nojaa muiden luokkien lähdekoodiin paljoakaan. Gtkmm:n kahden kantaluokan toteukset näyttävät samoista syistä kuin edellä on mainittu vähiten itsellisiltä.

Pohjapiirustus-luokkien osalta, pelkkiä proseduureja tarkastellessa, GTK+ ja Gtkmm ovat ylläpidettävimmät. Qt on GTK+:a tällä osa-alueella jäljessä, hieman vähemmän kuin wxWidgets. Oliosuunnittelun ja rakenteen osalta Qt on kuitenkin Gtkmm:ää edellä ja wxWidgets on kaikkein vaikeammin ylläpidettävä. WxWidgetsin ja Qt:n pohjapiirustus-luokkien tapauksessa suuri osa monimutkaisuudesta tulee modulien kytkennästä muihin moduleihin ja pienempi osa itse metodien monimutkaisuudesta. GTK+:n ja Gtkmm:n tapauksessa modulien oma sisäinen monimutkaisuus muodostaa suuremman osan. Gtkmm:n toteutus on kuitenkin metodien monimutkaisuudesta huolimatta edellä Qt:n ja etenkin wxWidgetsin vastaavia.

8.5 Asynkronisten viestien toteutus

Analyysitulokset saatiin laskemalla CCCC:n avulla kaikkien asynkronisten viestien toteutukseen oleellisesti osallistuvien luokkien tulokset. Luokat valittiin tutkimalla lähdekoodista ja luokkien kommenteista oikeat luokat. Liitteessä 6 sijaitsevat proseduurien, olio-suunnittelun ja rakenteellisten metriikoiden tulokset saatiin samalla menetelmällä kuin kappaleen 8.4 tulokset.

Johtuen kirjastojen täysin erilaisista lähestymistavoista asynkronisten metodien tutkimus on hieman vaikeaa muttei kuitenkaan mahdotonta. WxWidgetsin lähestymistapa on toteuttaa asynkroniset viestit muutamalla yksinkertaisella jokaisen luokan mukana kulkevalla kokonaislukumuuttujalla ja yhdellä pienellä luokalla. Tästä johtuen wxWidgetsin toteutustapa ei ole kovinkaan monimutkainen. WxWidgetsin asynkronisten viestien toteutus onkin kaikkien muiden metriikoiden paitsi rakenteellisten mukaan selkeästi yksinkertaisin. Proseduurien osalta Qt:n toteutus on selkeästi monimutkaisin, tosin Qt:n signaalien toteutus sisältyy erottamattomalla tavalla Qt:n introspektion toteutukseen ja näin ollen mitattuun kokonaisuuteen kuuluu muutakin toiminnallisuutta kuin vain asynkroniset viestit. Gtkmm:n sisältämä asynkronisten viestien toteutus on proseduurien osalta Qt:n ja wxWidgetsin välistä, joskin sisältää ainoastaan tämän toiminnallisuuden. Oliosuunnittelultaan Qt:n introspektion toteutus on lähellä Gtkmm:n vastaavaa, mutta on kuitenkin hieman monimutkaisempi kokonaisuus, erityisesti painotettujen metodien osalta. Qt:n introspektion toteutus ei sisällä ollenkaan ulostulovuota ja onkin rakenteellisten metriikoiden osalta yksinkertaisin. Gtkmm:n toteutus on sisääntulovuon osalta suhteellisesti saman suuruinen, mutta sisältää enemmän ulostulovuota ja on näin ollen monimutkaisempi. WxWidgetsin tapahtumien käsittely sisältää huomattavan määrän ulostulovuota ja paljastaa näin vähäisen lähdekoodinsa olevan rakenteellisesti monimutkaista ja hyvin riippuvaista wxWidgetsin muusta lähdekoodista.

8.6 Omat muokatut komponentit

Analyysitulokset laskettiin käyttöjärjestelmäkohtaisesti CCCC:n avulla. Analyysi täytyi tehdä jokaiselle käyttöjärjestelmälle erikseen, koska Qt:ssa ja wxWidgetissä luokat eroavat täysin erilaisten piirtorutiinien takia toisistaan. GTK+/Gtkmm toteuttaa oman piirtonsa Cairon avustuksella, jolloin Cairo kätkee oman rajapintansa taakse käyttöjärjestelmäkohtaisen toiminnallisuuden.

Liitteessä 7 sijaitsevat McCaben metriikoilla mitatut tulokset saatiin mittaamalla ensin kaikissa käyttöjärjestelmissä käytössä olevat yhteiset osat ja tämän jälkeen mitattiin kukin käyttöjärjestelmäkohtainen osa. Käyttöjärjestelmäkohtaiset tiedostot valittiin sisällön sekä lähdekoodin kommenttien parusteella. Koska McCaben avustuksella ei arvioida tiedostojen ulkopuolisia riippuvuuksia, CCCC:lle annettiin syötteeksi taulukossa listatut tiedostot. C-kielen erilaisuus otettiin huomioon antamalla CCCC:lle C-kielen käyttöä tukevat parametrit. Liitteessä 7 esitellyt olio-suunnittelua käsittelevät tulokset saatiin samoin kuin proseduureille, analyysi tehtiin kuitenkin koko kirjaston osan laajuudelta, koska olio-pohjaiset menetelmät edellyttävät riippuvuuksien analyysiä. Liitteessä 7 esitellyt rakenteellisten metriikoiden tulokset saatiin samoin kuin olio-pohjaisten menetelmien tulokset.

Pelkästään proseduurien monimutkaisuutta mitattaessa, Qt on kaikissa käyttöjärjestelmissä monimutkaisin ja täten vaikeimmin ylläpidettävä. WxWidgets on puolestaan Gtkmm:aa monimutkaisempi Windowsissa ja Linuxissa, Mac Os X:ssa Gtkmm on wxWidgetsiä monimutkaisempi. Gtkmm:n yksinkertaisuuden proseduurien osalta selitää Gtkmm:n käyttämä Cairo kirjaston käyttö, korkeampi abstraktiotaso mataloittaa piirtorutiinien monimutkaisuutta. Olio-suunnittelun laatua arvioitaessa tilanne kääntyy toisin päin, Gtkmm-on olio-suunnittelun näkökulmasta selkeästi monimutkaisin kirjasto. Runsas painotettujen metodien määrä ja kytkentä selittää Gtkmm:n proseduurien matalaa cyklomaattista kompleksisuutta. Suuri osa kantaluokan toiminnasta tehdään jossain muualla kuin itse luokan lähdekoodissa. Qt:n on olio-suunnittelunsa osalta wxWidgetsiä laadukkaampi Windows ja Linux

käyttöjärjestelmissä, wxWidgetsin saadessa paremman arvosanan mac OS X toteutuksensa osalta. Qt:n ja WxWidgetsin oliosuunnittelun välinen ero vaikuttaisi olevan suunnilleen sama kuin niiden proseduurien toteutusten välinen ero. Rakenteelliset metriikoiden paras tulos paljastaa Qt:n näkyvän kantaluokan toteutuksen muita paremman kapseloinnin ja tästä seuraavan suuremman sisäisen monimutkaisuuden. Tuloksista voidaan nähdä myöskin rakenteellisten metriikoiden olevan lähes samassa suhteessa toisiinsta käyttöjärjestelmä kohtaisia toiminnallisuuksia tutkittaessa. Minkään kirjaston kohdalla käyttöjärjestelmä kohtaisilla toteutuksilla ei näytä olevan juurikaan vaikutusta kirjaston rakenteeseen. Gtkmm:n wxWidgetsiä huomattavasti korkeampi ulostulovuo kertoo todennäköisesti näkyvän kantaluokan vahvasta nojaamisesta ulkoiseen toiminnallisuuteen etenkin Cairoon ja täten heikentyneestä kapseloinnista. Myös Gtkmm:n luonne rajapintana GTK+:n toiminnallisuuteen varmasti selittää osan tästä korkeasta määrästä. WxWidgetsin Gtkmm:ää suurempi sisääntulovuo kertoo näkyvän kantaluokan olevan olemassaolon ehdoton edellytys kirjaston muille luokille ainakin verrattuna muihin tässä työssä esiteltyihin kirjastoihin.

8.7 Piirtorutiinien toteutus

Piirtorutiinien toteutusta arvioitiin analysoimalla kirjaston koko piirtorutiinit toteuttava järjestelmä. Qt:n kohdalla arvioitiin kaikki matalan tason piirron tekevät luokat. GTK+:n kohdalla arvioitiin kaiken piirron suorittava Cairo kirjasto, sekä tähän liittyvät C++ rajapinnat. WxWidgetsistä arvioitiin kaikkien alustojen omia piirtorajapintoja käyttävät piirtokontekstiluokat.

Liitteessä 8 esitellyt proseduurien analyysitulokset saatiin CCCC:n avustuksella käsittelemällä kaikki ensimmäisessä sarakkeessa mainitut tiedostot yhdellä suorituskerralla. Cairon C pohjaisuus otettiin huomioon analyysissä. WxWidgetsin tapauksessa kaikkien alustojen toteutukset on lisätty samoihin tiedostoihin, joten kaikki jouduttiin arvoimaan yhdessä. Normaalisti makrojen vaikutuksesta osa alustakohtaisesta toiminnallisuudesta jäisi käyttämättä. Olio-suunnittelua ja rakenteellisia metriikoita

mittaavat tulokset saatiin samalla tavoin kuin proseduureillekin.

Liitteestä 8 nähdään että GTK+:n käyttämä Cairo sisältää selkeästi monimutkaisimmat proseduurit. Ero Qt:n piirtorutiineihin on kaksinkertainen ja wxWidgetsin rutiineihin kymmenkertainen. Cairon proseduurien monimutkaisuus on selitettävissä sillä että toisin kuin Qt ja wxWidgets sen piirtorutiinit ovat hyvin yleiskäyttöisiä, eivät vain pelkkien ikkunointikirjaston komponenttien piirtämiseen tarkoitettuja. Qt:n proseduurit ovat puolestaan wxWidgetsin vastaavia monimutkaisempia erilaisen abstraktiotasonsa takia, Qt käyttää mahdollisimman matalan tason piirtorutiineja piirtonsa toteuttamiseen. Oliosuunnittelun osalta, kun tutkitaan pelkästään painotettuja metodeita tilanne on täsmälleen samanlainen kuin proseduurienkin kohdalla. Eroa sen sijaan syntyy kun tutkitaan moduleiden kytkentää. WxWidgetsin proseduurien matala kompleksisuus selittyy erityisesti wxDC luokan sisältämä suurella kytkennällä. Matala kompleksisuus siis kompensoituu suuremmalla jaetun toiminnallisuuden määrällä. Qt saavuttaa suurinpiirtein saman suuruisen kytkennän Cairon kanssa. Mutta suhteessa painotettujen metodien määrään Cairon luokat ovat kaikkein vähiten kytkettyjä. Cairon luokat ovat siis hyvin kapseloituja, vaikka sisältävät suuren määrän monimutkaista lähdekoodia. Qt puolestaan edustaa välimuotoa Cairon ja wxWidgetsin väliltä. Qt on enemmän itsenäinen kuin wxWidgets, mutta kuitenkin vähemmän kuin Cairo. Rakenteellisten metriikoiden osalta kirjastot vaikuttavat hyvin tasapäisiltä, pieniä eroja kuitenkin on. sisääntulovuon osalta Cairo sisältää eniten vuota, ero johtunee runsaasta määrästä lähdekoodia, jota muut kirjaston osat käyttävät hyväkseen. WxWidgets sisältää hieman enemmän sisääntulovuota kuin Qt. Todennäköinen selitys tälle on WxWidgetsin piirtoluokkien suurempi kytkentä muuhun kirjaston toiminnallisuuteen. Qt kuitenkin sisältää kolmikon eniten ulostulovuota. Todennäköisesti Qt:n sisältämä korkea kytkennän määrä selittyy enemmän ulostulovuolla, tämä puolestaan kertoo Qt:n toteutuksen olevan wxWidgetsiä ja Cairoa enemmän painottunut käyttämään piirtorutiinien ulkopuolisia luokkia hyväkseen. WxWidgets ja Cairo sisältävät vastaavaa toiminnallisuutta jonkin verran mutta kuitenkin melkein kaksi kertaa vähemmän kuin Qt. Tästä voidaan päätellä Cairon ja WxWidgetsin käyttävän piirtorutiiniluokkien lähdekoodia hyväksi enemmän muussa lähdekoodissaan kuin Qt.

8.8 Analyysitulosten yhteenveto

Taulukossa 10 on näkyvissä tiivistelmä kappaleen 8 analyysituloksista.

Taulukko 10: Tiivistelmä merkittävimmistä analyysituloksista

Ominaisuus	**Qt**	**GTK+/Gtkmm**	**wxWidgets**
Käännetyn lähdekoodin viemä tila	Suuri, paljon ominaisuuksia	Keskisuuri, paljon irrallisia itsenäisiä osia	Pieni, yhtenäinen kokonaisuus
Merkkijonot	Monimutkainen toteutus	Keskitason toteutus	Ylläpidettävä yksinkertainen toteutus
Asynkroniset ja normaalit listat	Ylläpidettävä yksinkertainen toteutus	Keskitason toteutus	Monimutkainen toteutus
Kantaluokan toteutus	Yksinkertainen vahvasti riippuvainen kantaluokka	Keskitasoinen vahvasti riippuvainen kantaluokka	Monimutkainen, mutta itsenäinen kantaluokka
Asynkronisten viestien toteutus	Monimutkainen, oliosuunnittelun osalta riippuva ja rakenteeltaan itsenäinen	Monimutkainen ja melko voimakkaasti riippuva toteutus	Hyvin yksinkertainen, mutta voimakkaasti riippuvainen kirjaston muusta toiminnallisuudesta
Piirtorutiinien toteutus	Keskitasoinen ylläpidettävyys, kohtuullisen riippuvainen	Yleiskäyttöinen, hyvin kapseloitu, mutta hyvin monimutkainen	Hyvin yksinkertainen, mutta erittäin riippuvainen kirjaston muusta toiminnallisuudesta

9. YHTEENVETO

Tämän tutkimuksen tarkoituksena oli löytää vastauksia siihen mikä on avoimen lähdekoodin kirjastojen toteutuksen tämän hetkinen taso. Vastauksia toteutukseen lähdettiin etsimään McCaben, Henryn&Kafuran ja Chidamberin & Kemererin esittämillä staattisilla menetelmillä. Varsinaiset mittaukset tehtiin käyttämällä hyväksi Tom Littlefairin väitöskirjassaan kehittämää työkalua, sekä mittaamalla käännettyjen kirjastojen eri osien viemä tila eri käyttöjärjestelmissä. Mitattujen tulosten merkitystä arvioitiin kahdesta suunnasta. Yhtäältä vertaamalla kirjastojen suunnilleen samaa asiaa toteuttavia ominaisuuksia toisiinsa. Ja toisaalta vertaamalla niitä kullekin kirjastolle ominaisiin arkkitehtuurisiin ratkaisuihin.

Qt tarjoaa sovelluksille yhtenäisen rajapinnan ja on kehitettävän sovelluksen kannalta yksi kokonaisuus. Qt toteuttaa itse kaiken tarvitsemansa toiminnallisuuden. Qt:n pienikokoinen ydin mahdollistaa kohtuullisen pienikokoisten ei-graafisten sovellusten luomisen. Graafisen toiminnallisuuden tai web-sisällön lisääminen Qt sovellukseen kasvattaa sovelluksen kokoa suhteellisen paljon muihin kirjastoihin verrattuna. Qt:ssa on muita monimutkaisempi merkkijonoluokan toteutus, mutta erittäin laadukkaat säiliöluokat. Qt:n yhteinen kantaluokka nojaa vahvasti kirjaston muihin osiin ja on lähdekoodiltaan suhteellisen yksinkertainen, pohjapiirustus-luokkien osalta Qt on keskitasoa. Qt sisältää erittäin monimutkaisen, mutta erittäin paljon toiminnallisuutta sisältävän asynkronisten viestien toteutuksen. Qt:lla tehdyt omat luokat sisältävät muita kirjastoja laajemman toiminnallisuuden, mutta ovat paremmin kapseloituja. Qt sisältää kirjaston muuhun toiminnallisuuteen voimakkaasti liitetyn suhteellisen laajan piirrosrajapinnan.

GTK+:n toiminnallisuus on toteutettu useiden toisistaan irrallisten osakokonaisuuksien avulla. GTK+ koostuu yleiskäyttöisistä ja melko pienistä osista, joka mahdollistaa sovelluksen koon pitämisen suhteellisesti pienenä. GTKmm:n tarjoamat C++-rajapinnat ovat kuitenkin kohtuullisen suuria. GTK+:n merkkijono-luokka on keskitasoa ja säiliö-luokkien toteutus muita monimutkaisempi. Gtkmm:n yhteinen-kantaluokka on vaikeasti

ylläpidettävä, mutta pohjapiirustus-luokat ovat vastaavasti erittäin ylläpidettävät. Gtkmm:n osana jaeltava asynkronisten viestien toteutus on WxWidgetsiä monimutkaisemmin toteutettu, mutta Qt:a yksinkertaisempi. GTK+/Gtkmm:n avustuksella tehdyt omat kantaluokat ovat hyvin ylläpidettäviä, mutta hyvin riippuvia Cairon toiminnallisuudesta. GTK+/Gtkmm piirtorutiinit perustuvat erittäin laajaan ja monimutkaiseen Cairoon.

WxWidgets tarjoaa sovelluksille yhtenäisen rajapinnan, jonka toteutus voi perustua yhtä hyvin toiseen ikkunointikirjastoon kuin matalan tason piirtorajapintaankin. WxWidgets on käännetyn lähdekoodin osalta erittäin pieni. Joskin WxWidgetsin ytimen suhteellisesti suurehko koko asettaa suhteellisen korkean rajan minimaaliselle sovelluksen koolle. WxWidgetsin merkkijonoluokka on varsin ylläpidettävä ja säiliö-luokat keskitasoa. WxWidgetsin pohjapiirustusluokka on tarpeettoman monimutkainen ja yhteinen-kantaluokka on hyvin kapseloitu mutta monimutkainen. WxWidgets sisältää erittäin yksinkertaisen asynkronisten viestien toteutuksen, joka on voimakkaasti liitetty kirjaston muuhun toiminnallisuuteen. WxWidgetsillä tehdyt omat komponentit ovat keskitasoisia ja Mac OS X:n osalta kaikkein parhaiten ylläpidettäviä. WxWidgets sisältää suppean ja helposti ylläpidettävän piirtorajapinnan.

Tutkimuksessa esitelty analyysi kirjastojen tilastosta auttanee sopivaa kirjastoa etsiviä kehittäjiä arvioimaan kirjastojen toteutusten laatua. Erityisesti kirjaston ylläpidettävyys-analyysi lienee arvokasta mahdollisia ohjelmistorajapintojen muutoksia ennakoitaessa. Tutkimus kaipaa lisätutkimusta erityisesti ohjelmistorajapintojen laadukkuuden, dokumentaation laadukkuuden ja sekä suorituskyvyn osalta. Staattisten menetelmien rinnalle myöskin kaivataan esimerkiksi käyttäjätutkimukseen perustuvia analyysejä. Esimerkiksi Littlefairin kehittämä dikotominen kysymysanalyysi[2] voisi olla hyvin paljastava, jos se tehtäisiin kaikkia kolmea kirjastoja käyttäneille kehittäjille.

Lähteet

1. Diomidis Spinellis, A tale of four kernels, International conference on software engineering, 2008, sivut 381-390

2. Tim Littlefair, An investigation into to the use of software code metrics in the industrial software development environment, Ph.D Thesis, Edith Cowan University, 2001, sivut 10 - 83, 121 – 142, 178 – 186, 223 - 231

3. Thomas J McCabe, A complexity measure, IEEE transactions on software engineering, vol. se-2, no.4, 1976, sivut 308-320

4. Sallie Henry, Dennis Kafura, Software Structure Metrics Based on Information Flow, IEEE transactions on software engineering, vol. se-7, no.5, 1981, sivut 510-518

5. Shyam R Chidamber, Chris F Kemerer, Towards a metrics suite based for object oriented design, Object-Oriented Programming: Systems, Languages and Applications (OOPSLA), vol. 26, no. 11, 1991, sivut 197-211

6. G. Booch, Object Oriented Development, IEEE Transactions on Software Engineering, vol. se-12, no. 8, 1986, sivut 211-221

7. Ashley Beard, A survey on open source software licenses, Journal of Computing Sciences in Colleges, no. 4, 2007, sivut 205-211

8. Jay Kruizenga, Enlightenment—the Next Generation of Linux Desktops, Linux Journal, no.7, October 2008

9. Jasmin Blanchette, Mark Summerfield, C++ GUI programming with Qt 3, 1st

edition, Prentice Hall, 2004

10. GTK usein kysytyt kysymykset[Internet]
    http://www.gtk.org/faq [Viitattu 15.7.07]

11. Julian Smart and Kevin Hock , Cross-Platform GUI Programming with wxWidgets,1st edition, Prentice hall, 2005

12. wxWidgets lisenssin kuvaus [Internet].
    http://www.wxwidgets.org/about/newlicen.htm [Viitattu1.7.07]

13. Qt 4.2 whitepaper, Trolltech Co, 2006

14. Qt 4.2 reference manual, Trolltech Co, Luokkakuvaukset, 2006

15. GTK virallinen sivusto [Internet].
    http://www.gtk.org [Viitattu 10.08.07]

16. Epävirallinen GTK:n Windows sivusto [Internet]
    http://www.gimp.org/~tml/gimp/win32/downloads.html[Viitattu 17.8.07]

17. Gtkmm C++- rajapinnan sivusto [Internet]
    http://www.gtkmm.org/index.shtml [Viitattu 15.9.07]

18. Gtkkmm dokumentaatio [Internet]
    http://www.gtkmm.org/documentation.shtml [Viitattu 21.9.07]

19. Glib dokumentaatio [Internet]
    http://developer.gnome.org/doc/API/2.0/glib/ [Viitattu 28.9.07]

20. Gtkmm ohjelmointi opas [Internet]
    http://www.gtkmm.org/docs/gtkmm-2.4/docs/tutorial/html/ [Viitattu 28.9.07]

21. Gtk ohjelmointi opas [Internet]
http://www.gtk.org/tutorial [Viitattu 28.9.07]

22. Glade käyttöopas [Internet]
http://glade.gnome.org/manual/ [Viitattu 24.10.07]

23. Cairon kuvaus Cairon kotisivulla [Internet]
http://cairographics.org/ [viitattu 31.10.07]

24. wxWidgets dokumentaatio [Internet]
http://www.wxwidgets.org/manuals/stable/wx_contents.html [Viitattu 10.10.07]

25. WxWidgets wiki [Internet]
http://www.wxwidgets.org/wiki/index.php/Main_Page [viitattu 7.11.07]

26. wxWidgets luokkien kuvaukset ja esimerkit [Internet]
http://www.wxwidgets.org/manuals/stable/wx_classesbycat.html
[Viitattu 10.10.07]

27. wxWidgets tapahtumien käsittelyn kuvaus [Internet]
http://www.wxwidgets.org/manuals/stable/wx_eventhandlingoverview.html
[Viitattu 17.10.07]

28. Gettext käyttöopas [Internet]
http://www.gnu.org/software/gettext/manual/gettext.html [Viitattu 24.10.07]

LIITE 1. Käännetyn lähdekoodin viemä tila Windowsissa

Käännettyjen Qt:n kirjastojen viemä tila.

Kirjaston nimi	*Koko(kt)*
QtCore	2089
QtGUI	7763
QtNetwork	895
QtSql	184
QtSvg	263
QtXml	328
Qt3Support	2254
QtTest	94
QtMultimedia	100
QtOpenGL	596
QtWebkit	9673
Yhteensä:	**24239**

Käännettyjen GTK+:n kirjastojen viemä tila.

Kirjaston nimi	*Koko(kt)*
LibGtk	4792
GObject	308
GLib	1076
Pango	328
ATK	148
GDK	830
GDK-pixbuf	147
Yhteensä:	**7629**

käännettyjen GTK+:n riippuvuuksien viemä tila.

Kirjaston nimi	*Koko(kt)*
Cairo	840
GNU libiconv	33
libpng	219
libjpeg	125
libexpat	140
Yhteensä	**1357**

käännettyjen GTK+:n C++-rajapintojen viemä tila.

Kirjaston nimi	*Koko(kt)*
LibGtkmm	2454
LibGlibmm	496
LibPangomm	160
LibAtkmm	241
LibGdkmm	266
Yhteensä:	**3617**

GTK+:n riippuvuuksien C++-rajapintojen viemä tila.

Kirjaston nimi	*Koko(kt)*
LibCairomm	100
Libxml++	380
LibSigc++	278
Yhteensä	**758**

WxWidgetsin kirjastojen viemä tila.

Kirjaston nimi	***Koko(kt)***
wxAui	305
wxBase	1067
wxNet	114
wxRichText	725
wxXML	117
wxCore	2678
wxAdvanced	659
wxMedia	91
wxGL	8
wxHTML	458
wxODBC	8
wxQA	97
Yhteensä	**6327**

LIITE 2. Käännetyn lähdekoodin viemä tila Linuxissa

Käännettyjen Qt:n kirjastojen viemä tila.

Kirjaston nimi	*Koko(kt)*
QtCore	3072
QtGUI	12697
QtNetwork	1126
QtSql	324
QtSvg	452
QtXml	363
Qt3Support	3686
QtTest	191
QtMultimedia	140
QtOpenGL	1002,8
QtWebkit	22630
Yhteensä:	**45683**

Käännettyjen GTK+:n kirjastojen viemä tila.

Kirjaston nimi	*Koko(kt)*
LibGtk	4132
GObject	279
GLib	792
Pango	291
ATK	126
GDK	687
GDK-pixbuf	111
GDK-X11-pixbuf	66
Yhteensä:	**6484**

Käännettyjen Cairon ja GTK+:n riippuvuuksien viemä tila.

Kirjaston nimi	***Koko(kt)***
Cairo	520
GNU libiconv	200
libpng	154
libjpeg	146
libexpat	162
Yhteensä	**1182**

Käännettyjen GTK+:n C++-rajapintojen viemä tila.

Kirjaston nimi	***Koko(kt)***
LibGtkmm	4352
LibGlibmm	354
LibPangomm	184
LibAtkmm	303
LibGdkmm	302
Yhteensä:	**5495**

Cairoon ja tärkeiden riippuvuuksien C++-rajapintojen viemä tila.

Kirjaston nimi	***Koko(kt)***
LibCairomm	128
Libxml++	1331
LibSigc++	22
Yhteensä	**1481**

WxWidgetsin kirjastojen viemä tila.

Kirjaston nimi	***Koko(kt)***
wxAui	569
wxBase	1512
wxNet	243
wxRichText	1172
wxXML	214
wxCore	5816
wxAdvanced	1136
wxMedia	71
wxGL	77
wxHTML	870
wxQA	217
Yhteensä	**11897**

LIITE 3. Käännetyn lähdekoodin viemä tila Mac OS X:ssa

Käännettyjen Qt:n kirjastojen viemä tila.

Kirjaston nimi	*Koko(kt)*
QtCore	2743
QtGUI	11321
QtNetwork	1280
QtSql	274
QtSvg	393
QtXml	442
Qt3Support	2962
QtTest	184
QtMultimedia	193
QtOpenGL	893
QtWebkit	23517
Yhteensä:	**44202**

Käännettyjen GTK+:n kirjastojen viemä tila.

Kirjaston nimi	*Koko(kt)*
LibGtk	3837
GObject	254
GLib	852
Pango	266
ATK	123
GDK	565
GDK-pixbuf	102
GDK-X11-pixbuf	61
Yhteensä:	**6060**

Käännettyjen Cairoon ja GTK+:n riippuvuuksien viemä tila.

Kirjaston nimi	***Koko(kt)***
Cairo	541
GNU libiconv	1085
libpng	180
libjpeg	135
libexpat	164
Yhteensä	**2105**

Käännettyjen GTK+:n C++-rajapintojen viemä tila.

Kirjaston nimi	***Koko(kt)***
LibGtkmm	3189
LibGlibmm	369
LibPangomm	176
LibAtkmm	299
LibGdkmm	299
Yhteensä:	**4332**

Cairon ja tärkeiden riippuvuuksien C++-rajapintojen viemä tila.

Kirjaston nimi	***Koko(kt)***
LibCairomm	201
Libxml++	1380
LibSigc++	29
Yhteensä	**1610**

WxWidgetsin kirjastojen viemä tila.

Kirjaston nimi	***Koko(kt)***
wxAui	446
wxBase	1166
wxNet	217
wxRichText	1075
wxXML	66
wxCore	3853
wxAdvanced	794
wxMedia	102
wxGL	82
wxHTML	696
wxODBC	152
wxQA	156
Yhteensä	**8805**

LIITE 4. Mittaustulokset ei-näkyville komponenteille

Proseduurien mittaustulokset McCaben menetelmällä. Tulokset ei-näkyville komponenteille.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
gdataset.h, gdataset.c	GData	78	GTK+	Assosiatiivinen taulukko
glist.h, glist.c	GList	115	GTK+	Lista
gstring.h, gstring.c	GString	119	GTK+	Merkkijono
list.h, list.cpp	wxListBase, wxStringList	100	wxWidgets	Lista
string.h, string.cpp	wxString	29	wxWidgets	Merkkijono
hash.h, hash.cpp	wxHashTable	86	wxWidgets	Assosiatiivinen taulukko
qlist.h, qlist.cpp	QSet, QVector, QList	53	Qt	Lista
qmap.h, qmap.cpp	QMap, QMapData	27	Qt	Assosiatiivinen taulukko
qstring.h, qstring.cpp	QString	573	Qt	Merkkijono

Olio-suunnittelun mittaustulokset Chidamberin ja Kemererin menetelmällä. Analyysitulokset ei-näkyville komponenteille.

Luokka	Painotetut metodit	Modulin kytkentä
QList	0	2
QListData	13	0
QSet	17	2
QVector	0	4
QString	75	18
QMap	0	1
QMapData	7	0
wxListBase	24	7
wxStringList	9	3

wxString	0	137
wxHashTable	27	5

Rakenteellisten metriikoiden mittaustulokset Henryn ja Kafuran menetelmällä. Analyysitulokset ei-näkyville komponenteille.

Luokka	**sisääntulovuo**	**ulostulovuo**
QList	2	0
QListData	0	0
QSet	0	2
QVector	4	0
QString	10	8
QMap	1	0
QMapData	0	0
wxListBase	1	6
wxStringList	1	2
wxString	137	0
wxHashTable	0	5

LIITE 5. Mittaustulokset käyttöliittymäkomponenteille

Proseduurien mittaustulokset McCaben menetelmällä. Kaikkien komponenttien kantaluokat ja pohjapiirustus-luokat.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
qobject.cpp, qobject.h, qobject_p.h	QObject	8	Qt	Kaikkien Qt:n luokkien kantaluokka
qlayout.cpp, qlayout.h, qlayout_p.h	QLayout	8	Qt	pohjapiirustus-luokka
qlayoutitem.h, qlayoutitem.cpp	QLayoutItem	120	Qt	pohjapiirustus-luokka
object.h, object.cpp	wxObject	41	wxWidgets	Kaikkien wxWidgetsin luokkien kantaluokka
sizer.h, sizer.cpp	wxSizer, wxSizerItem	316	wxWidgets	pohjapiirustus-luokka
gobject.h, gobject.c	GObject	252	GTK+	Kaikkien GTK+ luokkien yhteinen kantaluokka, C-pohjainen ei standardi olio-toteutus
object.h, object.cc	Object	19	glibmm	Kaikkien gtkmm luokkien kantaluokka
objectbase.h, objectbase.cc	ObjectBase	14	glibmm	Kaikkien luokkien ja rajapintojen kantaluokka
gtklayout.h, gtklayout.c	GtkLayout	85	GTK+	pohjapiirustus-luokka
layout.h, layout.cpp	GtkLayout	14	gtkmm	pohjapiirustus-luokka

Rakenteellisten metriikoiden mittaustulokset Chidamberin & Kemererin menetelmällä. Analyysitulokset kaikkien komponenttien kantaluokalle ja pohjapiirustus-luokalle.

Rajapinta	Painotetut metodit	Modulin kytkentä
QObject	8	27
QLayout	2	5
QLayoutItem	10	2
wxObject	15	17
wxSizer	42	9
wxSizerItem	20	10
Object	11	8
ObjectBase	22	21
GtkLayout	16	16

Rakenteellisten metriikoiden mittaustulokset Henryn ja Kafuran menetelmällä. Analyysitulokset kaikkien komponenttien kantaluokalle ja pohjapiirustus-luokalle.

Luokka	sisääntulovuo	ulostulovuo
QObject	0	30
QLayout	1	4
QLayoutItem	1	1
wxObject	3	14
wxSizer	4	5
wxSizerItem	7	3
Object	5	3
ObjectBase	5	16
GtkLayout	5	2

LIITE 6. Mittaustulokset asynkronisien viestien toteutukselle

Proseduurien mittaustulokset McCaben menetelmällä. Analyysitulokset asynkroniset viestit toteuttaville luokille.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
qmetaobject.cpp, qmetaobject.h, qmetaobject_p.h, qmetatype.h, qmetatype.cpp	QMetaObject, QMetaType, QMetaMethod, QMetaProperty	830	Qt	Qt:n introspektion toteuttavat luokat
Kaikki libsigc++ tiedostot	Kaikki sigc++ luokat, tärkeimmät sigc::signal, sigc::slot	156	Gtk+/Gtkmm	Gtk+ ja Gtkmm asynkronisen viestijärjestelmän toteuttavat luokat
event.h, event.cpp	wxEvtHandler	2	WxWidgets	WxWidgetsin tapahtumien käsittelyn hoitavat luokat

Olio-suunnittelun mittaustulokset Chidamberin & Kemererin menetelmällä. Analyysitulokset asynkroniset viestit toteuttaville luokille.

Rajapinta	Painotetut metodit	Modulin kytkentä
QMetaType	9	3
QMetaProperty	26	2
QMetaMethod	10	4
Sigc::Signal	10	3
Sigc::Slot	12	6
wxEvtHandler	2	10

Rakenteellisten metriikoiden mittaustulokset Henryn & Kafuran menetelmällä. Analyysitulokset asynkroniset viestit toteuttaville luokille.

Luokka	sisääntulovuo	ulostulovuo
QMetaType	3	0
QMetaProperty	2	0

QMetaMethod	4	0
Sigc::Signal	3	0
Sigc::Slot	3	3
wxEvtHandler	1	9

LIITE 7. Mittaustulokset omille muokatuille komponenteille

Proseduurien mittaustulokset McCaben menetelmällä. Näkyvien komponenttien kantaluokkien käyttöjärjestelmäriippumattomat osat.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
qwidget.h, qwidget.cpp, qwidget_p.h	QWidget	59	Qt	Kaikkien näkyvien komponenttien yhteinen kantaluokka
widget.h, widget.cc	Widget	57	GTK+/Gtkmm	Kaikkien näkyvien komponenttien yhteinen kantaluokka
control.h, control.cpp	wxControl	12	WxWidgets	Kaikkien näkyvien komponenttien yhteinen kantaluokka

Proseduurien mittaustulokset McCaben menetelmällä. Näkyvien komponenttien kantaluokat Windows versioissa.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
qwidget.h, qwidget.cpp, qwidget_p.h, qwidget_win.cpp	QWidget	262	Qt	Toteuttaa QWidgetin windows version
control.h, control.cpp	wxControl	62	wxWidgets	Toteuttaa wxControlin windows kohtaisen toiminnallisuud

				en

Proseduurien mittaustulokset McCaben menetelmällä. Näkyvien komponenttien kantaluokat Linux versioissa.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
qwidget.h, qwidget.cpp, qwidget_p.h, qwidget_x11.cpp	QWidget	767	Qt	Toteuttaa QWidgetin Linux version
control.h, control.cpp	wxControl	41	wxWidgets	Toteuttaa wxControlin Linux kohtaisen toiminnallisuuden

Proseduurien mittaustulokset McCaben menetelmällä. Näkyvien komponenttien kantaluokat Mac OS X versioissa.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
qwidget.h, qwidget.cpp, qwidget_p.h, qwidget_mac.mm	QWidget	59	Qt	Toteuttaa QWidgetin Mac OS X version
control.h, control.cpp	wxControl	7	wxWidgets	Toteuttaa wxControlin Mac OS X kohtaisen toiminnallisuuden

Olio-suunnittelun mittaustulokset Chidamberin ja Kemererin menetelmällä. Näkyvien komponenttien yhteiset kantaluokat.

Rajapinta	Painotetut metodit	Modulin kytkentä
QWidget	8	3
Widget	48	41
wxControl	4	6

Olio-suunnittelun mittaustulokset Chidamberin ja Kemererin menetelmällä. Näkyvien komponttien yhteiset kantaluokat Windowsissa.

Rajapinta	Painotetut metodit	Modulin kytkentä
QWidget	8	3
wxControl	14	12

Olio-suunnittelun mittaustulokset Chidamberin ja Kemererin menetelmällä. Näkyvien komponttien yhteiset kantaluokat Linuxissa.

Rajapinta	Painotetut metodit	Modulin kytkentä
QWidget	8	3
wxControl	18	16

Olio-suunnittelun mittaustulokset Chidamberin ja Kemererin menetelmällä. Näkyvien komponttien yhteiset kantaluokat Mac OS X:ssa.

Rajapinta	Painotetut metodit	Modulin kytkentä
QWidget	8	3
wxControl	4	8

Rakenteellisten metriikoiden mittaustulokset Henryn ja Kafuran menetelmällä. Näkyvien komponttien yhteiset kantaluokat.

Luokka	sisääntulovuo	ulostulovuo
QWidget	3	0
Widget	11	30
wxControl	6	0

Rakenteellisten metriikoiden mittaustulokset Henryn ja Kafuran menetelmällä. Näkyvien komponttien yhteiset kantaluokat Windowsissa.

Luokka	sisääntulovuo	ulostulovuo
QWidget	3	0

wxControl	11	1

Rakenteellisten metriikoiden mittaustulokset Henryn ja Kafuran menetelmällä. Näkyvien komponttien yhteiset kantaluokat Linuxissa

Luokka	**sisääntulovuo**	**ulostulovuo**
QWidget	3	0
wxControl	14	2

Rakenteellisten metriikoiden mittaustulokset Henryn ja Kafuran menetelmällä. Näkyvien komponttien yhteiset kantaluokat Mac OS X:ssa.

Luokka	**sisääntulovuo**	**ulostulovuo**
QWidget	3	0
wxControl	8	0

LIITE 8. Mittaustulokset piirtorutiineille

Proseduurien mittaustulokset McCaben menetelmällä. Tulokset piirtorutiinit toteuttaville luokille.

tiedostot	Toteuttaa rajapinnat	Kaikkien metodien cyclomaattinen kompleksisuus McCaben mukaan	Kirjasto	Käyttö-tarkoitus
Kaikki src/gui/painting/ alta löytyvät tiedostot	Tärkeimmät rajapinnat: QPainter, QPaintDevice, QPaintEngine	4696	Qt	Piirtorutiinien toteutus, alustariippu-mattomasti ja alustakohtaisesti
Kaikki Cairon tiedostot	Ei olio-pohjaista rajapintaa	10169	Cairo(Gtk+)	Piirtorutiinien toteutus alustariippumatt omasti C-kielellä
Kaikki Cairomm:n tiedostot	Tärkeimmät rajapinnat: Context, Surface	256	Cairo(Gtkmm)	Cairon C++ rajapinta
dc.h, dcclient.h, dcscreen.h, dcmemory.h, sekä toteutus-tiedostojen alustakohtaiset versiot	wxDC, wxClientDC, wxScreenDC, wxMemoryDC	726	wxWidgets	wxWidgetsin piirtorajapinnan muodostavat luokat

Olio-suunnittelun analyysitulokset Chidamberin & Kemererin menetelmällä tärkeimmille piirtorutiinien rajapinnoille

Rajapinta	Painotetut metodit	Modulin kytkentä
QPaintDevice	4	10
QPaintEngine	28	23
QPainter	1	5
Context	114	22
Surface	24	18
wxDC	71	30
wxClientDC	1	0

wxScreenDC	5	2
wxMemoryDC	7	3

Rakenteellisten metriikoiden analyysitulokset Henryn & Kafuran menetelmällä tärkeimmille piirtorutiinien rajapinnoille.

Luokka	**sisääntulovuo**	**ulostulovuo**
QPaintDevice	2	8
QPaintEngine	17	6
QPainter	0	5
Context	22	0
Surface	9	9
wxDC	19	11
wxClientDC	0	0
wxScreenDC	2	0
wxMemoryDC	3	0