Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Matti Nieminen

Opinnäytetyö

Qt-ohjelmointitekniikat Java-osaajan näkökulmasta

Työn ohjaaja FL Paula Hietala
Työn tilaaja Kilosoft Oy, yhteyshenkilönä talousjohtaja Kimmo Hakkarainen
Tampere 6/2009

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä	Matti Nieminen
Työn nimi	Qt-ohjelmointitekniikat Java-ohjelmoijan näkökulmasta
Sivumäärä	48
Valmistumisaika	Kesäkuu 2009
Työn ohjaaja	Paula Hietala
Työn tilaaja	Kilosoft Oy

Tiivistelmä

Tämän opinnäytetyön tarkoituksena on esitellä Qt-ohjelmointia Java-ohjelmoijan näkökulmasta. Työ voi toimia myös oppaana Qt-sovelluskehityksen aloittamiseen. Työssä käsitellään tärkeimpiä Qt-ohjelmointitekniikoita samalla esittäen, miten niitä on käytetty Mobres-TJ-nimisen sovelluksen luomiseen Kilosoft Oy:n sisäiseen käyttöön.

Qt-osaajien tarve tulee kasvamaan, ja tästä syystä luotiin harjoitus, jonka tarkoituksena on antaa mahdollisuus kokeilla Qt:ta monipuolisesti. Näkökulmaksi valitsin oman Java-taustani, jota tuodaan esille Qt:n ja Javan samankaltaisuuksien vuoksi: molemmat sovelluskehitysympäristöt ovat alustariippumattomia ja helppokäyttöisiä.

Qt-tekniikat ja -luokkakirjasto osoittautuivat helpoksi tavaksi tuottaa graafisella käyttöliittymällä varustettuja sovelluksia. Useat opinnäytetyön esimerkit todistavat, että Qt tuo klassista C++-ohjelmointia selkeästi lähemmäksi Javaa tarjoten ohjelmoijalle valmiita ratkaisuja C++-ohjelmoinnin yleisimpiin kompastuskiviin. Näin opinnäytetyöni rohkaisee epäilevää ohjelmoijaa tutustumaan Qt-kehitysympäristöön.

Qt on kokenut muutoksia juuri ennen työni tekemistä, joten yritin käyttää työssäni tuoretta aineistoa Qt:n osalta. Qt-tutkimuksen lisäksi jouduin tutustumaan myös hyvin paljon C++-ohjelmointiin ja kehityin myös jonkin verran Java-ohjelmoijana.

Avainsanat Qt, Java, Ohjelmointi

TAMK University of Applied Sciences
Business Information Systems

Writer	Matti Nieminen
Thesis	Qt Programming from Java Programmer's Point of View
Pages	48
Graduation time	June 2009
Thesis Supervisor	Paula Hietala
Co-operating Company	Kilosoft Oy

Abstract

The purpose of my thesis is to demonstrate Qt programming from a Java programmer's point of view. This thesis can also work as a guide when starting programming with Qt. It describes the most important aspects of Qt programming by demonstrating how I have used Qt framework to create a program called Mobres-TJ for internal use of Kilosoft Oy.

I chose Qt because the demand for Qt programmers will likely grow in the future. For this reason, Mobres-TJ was implemented as an exercise that gave me an opportunity to try Qt programming diversely. My past experience with Java was chosen as the point of view, because I constantly compare Qt to Java because of their similarities as platform-independent and user friendly programming environments.

Qt programming and its class library turned out to be an easy way to develop GUI programs. Several examples in my thesis will prove that Qt brings the classic C++ programming closer to Java offering complete solutions for the programmers who find C++ challenging to use. In a way, my thesis invites sceptical programmers to try Qt framework.

Qt had gone through changes just before I started working with it, so I tried to use as fresh material as possible in my studies regarding Qt. In addition to studying Qt I had to get acquainted with C++ and my programming skills and knowledge in Java also improved a lot.

Keywords Qt, Java, Programming

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Sisällysluettelo

1 Johdanto

Aloitin ohjelmoinnin tutustumalla Java-kieleen muutama vuosi ennen tämän työn tekemistä ja vuonna 2008 olin jo Kilosoft Oy:llä harjoittelijana. Tänä aikana olen tutustunut enimmäkseen Javaan ja olen päässyt vain kokeilemaan muita ohjelmointikieliä ja -tekniikoita. Kilosoftilla kuitenkin työskentelee Java-ohjelmoijien lisäksi useiden muiden ohjelmointikielten asiantuntijoita. Minua on aina kiinnostanut ohjelmointi hyvin laajasti, mutta osittain kurssitarjonnan vuoksi minusta on tullut Java-osaaja.

Uusien ohjelmointikielien ja -ympäristöjen omaksuminen on usein kuitenkin haastavaa ilman koulutusta, ja usein uusia osaamisalueita pitää pystyä hallitsemaan nopealla aikataululla. Opinnäytetyössäni olen perehtynyt Qt-sovelluskehitysalustaan Java-tausta mielessäni. Uskon vahvasti siihen, että Qt tulee olemaan käytetyimpiä ohjelmointiympäristöjä niin Suomessa kuin maailmanlaajuisesti ja tarkoitukseni on ottaa varaslähtö kyseisen tekniikan opiskeluun ja näin varmistaa asemani työmarkkinoilla valmistumisen jälkeen. Lisäksi Kilosoft saa arvokasta Qt-itseopiskelumateriaalia ja työntekijän, jolla on ainakin Qt:n perusteet hallinnassa.

Qt:n tehtävä on ollut tarjota alustariippumaton luokkakirjasto graafisten sovellusten tekemiseen. Nykyään se tarjoaa kuitenkin kokonaisen kehitysympäristön ja helpottaa C++-ohjelmointia suuresti. Qt on tunnettu pitkään varsinkin avoimen lähdekoodin käyttäjien piireissä, mutta sen suosio todennäköisesti kasvaa räjähdysmäisesti myös mobiiliohjelmoijien keskuudessa vuonna 2009, sillä Nokia osti Qt:n omistavan norjalaisen ohjelmistoyhtiön, Trolltechin.

Tällä hetkellä Qt on saatavilla Windowsille, Linuxille, Mac OS X:lle ja joihinkin sulautettuihin järjestelmiin. Nokia aikoo lisäksi tuoda Qt:n omiin Symbian S60-alustan matkapuhelimiinsa. Tämän työn kirjoitushetkellä on mahdollista asentaa erilaisia Qt-demoja osaan Nokian matkapuhelimista, mutta sitä ei paketoida osaksi Nokian uusia matkapuhelinmalleja ainakaan toistaiseksi. Siitä huolimatta useat mobiilialan yritykset etsivät Qt-osaajia jo nyt.

Opinnäytetyötä tehdessäni olen harjoituksenomaisesti suunnitellut ja toteuttanut sovelluksen nimeltä Mobres-TJ Kilosoftin omaan käyttöön. Tarkoituksena on tuottaa Qt-osaamista ja -itseopiskelumateriaalia kirjallisen osuuden toimiessa yhdessä tuotetun sovelluksen kanssa johdatuksena Qt-ohjelmointitekniikoihin, jota esimerkiksi Kilosoftin Java-osaajat voivat hyödyntää.

Työni tarkoituksena onkin tutkia Qt-ohjelmistokehityksen perusteita, kun taustalla on suhteellisen kattava Java-osaaminen. Opinnäytetyöni tulee keskittymään Javan ja Qt:n eroavuuksiin ja yhtäläisyyksiin sekä tuomaan niitä esille Java-ohjelmoijan näkökulmasta. Opinnäytetyössä esitellään toteuttamani sovellus ja käsitellään Qt-sovelluskehitystä yleisellä tasolla ja käytännönläheisesti, sekä yritetään hieman raottaa, mitä pinnan alla tapahtuu. Myöhemmin työssä esitellään vaihtoehtoja erilaisille Javasta tutuille ohjelmointitekniikoille ja esitellään Mobres-TJ:ssä käytettyjä Qt-tekniikoita.

Opinnäytetyön lukija, jolla on Java-kokemusta, saa opinnäytetyöstäni eniten irti, mutta ammattikorkeakoulutason tietämys olio-ohjelmoinnista ja ohjelmistosuunnittelusta riittää. Pyrin käsittelemään C++-osuudet mahdollisimman suppeasti, mutta totuus on se, että Qt-sovelluskehitystä tehdään oletusarvoisesti C++-kielellä, ja ainakin perustietämys sen ohjelmointitekniikoista ja -tavoista on välttämätöntä. Käsittelen työssäni lähinnä niitä tekniikoita, jotka itse miellän kuuluviksi olio-ohjelmoinnin perusteisiin. Tarkoituksena ei kuitenkaan ole käsitellä Javaan tai C++:aan liittyviä asioita, mikäli Qt ei tarjoa niihin uutta näkökulmaa tai toimintatapaa.

2 Mobres

Kilosoft Oy päätti kehittää Mobres-nimisen sovellusperheen Qt-harjoituksena. Mobres-perheen avulla on tarkoitus hallinnoida yrityksen mobiileja resursseja, kuten lähettejä, paketteja, asiakkaita ja osoitteita. Esimiesten on kyettävä lisäämään ja hallinnoimaan näitä resursseja ja lähetit puolestaan kuljettavat paketit oikeisiin osoitteisiin, kuittaavat lähetyksen ja vastaanottavat seuraavan tehtävän.

Lähetit toimivat siis pääsääntöisesti kentällä, joten järjestelmää pitää pystyä käyttämään tehokkaasti myös toimiston ulkopuolelta. Sovellusten suunnitteluprosessin aikana Symbianin Qt-tuki oli vielä erittäin varhaisessa vaiheessa, joten laitteeksi tähän tehtävään valittiin Nokia N810 Internet Tablet Linux-alustansa vuoksi, sillä laitteeseen on mahdollista asentaa Qt-sovelluksia. Kyseisessä laitteessa on myös GPS-piiri, jonka avulla laite voidaan paikantaa.

Näin heti lähtötilanteeksi otettiin Qt:n tuleva rooli kannettavissa laitteissa ja mahdollisuus käyttää samaa sovellusta eri alustoilla. Suunnitteluhetkellä ei ollut mahdollisuutta varmistaa sovellusten toimivuutta Nokian matkapuhelimella, joten jouduimme luottamaan siihen, että sovellusten kääntäminen matkapuhelimelle onnistuu tulevaisuudessa suoraan.

2.1 Mobresin eri osat

Kenttäkäyttöön luotavan Mobres-sovelluksen nimeksi annettiin Mobres-M ja sen toteutti eräs toinen opiskelija opinnäytetyönään. Mobres-M:n haasteita ovat käyttöliittymän toteuttaminen pienelle kosketusnäytölle, sekä Internet- ja GPS-yhteyden luotettava hallinnointi. Sovelluksen tarkoitus on yksinkertaisuudessaan ottaa yhteys toimistolla sijaitsevaan palvelimeen, kuitata tehdyt kuljetukset, päivittää lähetin sijainti GPS:n avulla ja noutaa toimistolla sijaitsevalta palvelimelta seuraavan kohteen tiedot.

Mobiilikäytön lisäksi tuotettiin lähinnä PC-käyttöön tarkoitettu sovellus, joka sai nimekseen Mobres-TJ. Tämä sovellus tarkoitettiin esimieskäyttöön lähettien, pakettien ja asiakkaiden hallintaa varten. Mobres-TJ:n toteuttaminen oli oma osuuteni Mobres-tuoteperheestä ja sain lähes vapaat kädet sen tekemiseen. Tarkoituksena oli tutustua Qt:n

luokkakirjastoon, sen tarjoamiin käyttöliittymäelementteihin, karttanäkymän luomiseen ja tietokantayhteyden ja -hakujen järkevään toteuttamiseen.

Alussa pohdittiin myös mahdollisuutta luoda Mobres-S-niminen sovellus palvelimelle, jossa Mobres-tuoteperheen käyttämä tietokanta on. Näin mahdollisesti raskaat operaatiot saataisiin pois päätelaitteilta palvelinkoneelle, mikä olisi selkeä etu, kun asiakasohjelmaa on tarkoitus käyttää heikkotehoisella mobiililaitteella. Mobres-S:n luomisesta luovuttiin ainakin alkuvaiheessa, mutta koko sovellusperhe päätettiin toteuttaa siten, että palvelinsovelluksen lisääminen olisi myöhemmin mahdollisimman helppoa.

2.2 Käytetyt teknologiat

Pääteknologiana toimii tietysti Qt. Qt:ta käytetään käyttöliittymän ja sovelluslogiikan luomiseen. Ikään kuin kaupan päälle sovelluksesta tulee yhteensopiva kaikille Qt:n tukemille alustoille. Qt:n luokkakirjasto tarjoaa sovelluksia varten käyttöliittymäkirjaston lisäksi myös valmiit työkalut esimerkiksi tietokantayhteyksiä varten.

Palvelinkoneeseen asennettiin palvelinversio Ubuntu Linuxista, jonka uusin versio asennushetkellä oli 8.04. Palvelimelle asennettiin MySQL-tietokanta, sillä se oli minulle jo entuudestaan tuttu ja siihen liittyen on tarjolla erittäin paljon dokumentaatiota ja muuta materiaalia. MySQL on tyypillinen relaatiotietokanta, jossa tietoa tallennetaan tauluihin, jotka voivat olla yhteydessä toisiinsa tiettyjen sarakkeiden välityksellä. Tauluilla on useita sarakkeita, joiden arvot ovat määrätty tietotyyppien avulla. Sekä Mobres-M että Mobres-TJ käyttävät samaa tietokantaa.

Mobres-M toteutetaan Nokian Linux-pohjaiselle kämmenlaitteelle, jossa käyttöjärjestelmänä on Debianiin perustuva Maemo Linux. Maemo-ohjelmointia tehdään yleensä C-kielellä GTK+-grafiikkakirjaston avulla, mutta siihen on mahdollista asentaa Qt-tuki (maemo.org – Intro: Software Platform, 2009) Pohjimmiltaan Maemo on hyvinkin lähellä mitä tahansa Linux-jakelua, sillä se koostuu suurimmaksi osaksi samoista komponenteista. Näitä komponentteja täydentää Hildon UI-kehitysympäristö, joka sisältää kämmenlaitteelle sopivia käyttöliittymäkomponentteja.

3 Mistä Qt-ohjelmistokehityksessä on kyse?

Tässä luvussa työtäni käsitellään Qt-ohjelmoinnin yleistä problematiikkaa, kehitystyökalujen asentamista ja ohjelmointityön aloittamista. Luvussa käsitellään myös Qt-luokkakirjaston lyhyttä historiaa ja sen tulevaisuuden näkymiä. Alusta asti työssäni tutkitaan Qt:ta Javan avulla.

3.1 Qt:n historia ja tunnettavuus

Qt on suhteellisen lyhyessä ajassa noussut tuntemattomasta norjalaisesta C++-luokkakirjastosta yhdeksi maailman käytetyimmistä C++-kehitysympäristöistä. Qt:lla on lisäksi todennäköisesti erittäin pitkä ja valoisa tulevaisuus edessä sekä työpöytä- että mobiilisovelluksissa.

3.1.1 Mikä on Qt?

Qt (lausutaan “cute”) on usealla alustalla toimiva, graafisten C++-sovellusten luomiseen tehty olioperustainen kehitysympäristö ja luokkakirjasto. Se tarjoaa laajan käyttöliittymäkirjaston, mutta sitä voidaan aivan hyvin käyttää myös esimerkiksi kirjastokomponettien tekemiseen, sillä käyttöliittymäkirjasto on vain yksi Qt:n useista moduuleista. Qt:sta löytyy lisäksi XML-dataa, verkkoyhteyksiä, tietokantoja, kansainvälistämistä ja lokalisointia sekä esimerkiksi 2D- ja 3D-grafiikkaa käsittelevät luokkakirjastot (Qt Software Products, 2009). Lisäksi sovelluskehittäjille on tarjolla erilaisia makroja ja tekniikoita, joiden tarkoitus on yksinkertaistaa C++-ohjelmointia.

Qt on varteenotettava vaihtoehto, kun tarvitsee luoda graafinen C++-sovellus, josta pitää pystyä kääntämään versio useammalle alustalle, kuten Windowsille, Linuxille tai Macille. Tulevaisuudessa samojen sovellusten pitäisi toimia myös Symbian S60-alustalla, kunhan ne käännetään asianmukaisesti uudestaan.

3.1.2 C++ Qt-ohjelmoinnin perustana

Haavard Nordilla ja Eirik Chambre-Engillä oli visio alustariippumattomasta graafisten sovellusten kehitysympäristöstä C++:lle jo ennen 1990-lukua. Vuonna 1991 Nord kirjoitti ensimmäiset luokat tästä kehitysympäristöstä, joka nykyään tunnetaan nimellä

Qt. Nimensä Qt sai hieman myöhemmin Q-kirjaimen kauniin kirjoitusasun ja siihen liitetyn t-kirjaimen mukaan, joka tulee englanninkielisestä sanasta toolkit. Samoihin aikoihin Chambre-Eng ja Nord perustivat yhtiön nimeltä Quasar Technologies, joka nykyään tunnetaan nimellä Trolltech (Blanchette & Summerfield 2008, xix.)

Huhtikuun 20. päivä vuonna 1995 Qt 0.90 julkaistiin. Jo tällöin Qt oli käyttökelpoinen sekä Windowsilla että Unix-järjestelmillä tarjoten samat rajapinnat molemmille alustoille. Qt alkoi todella saamaan julkisuutta, kun Matthias Ettrich päätti käyttää Qt:ta pohjana KDE:lle, joka on nykyään yksi suurimmista Linux-työpöytäympäristöistä. Ettrich liittyi myöhemmin osaksi Trolltechia (Blanchette & Summerfield 2008, xx). Vaikka alussa Qt:ta on tarjottu erikseen avoimen lähdekoodin lisenssillä ja kaupallisella lisenssillä, Qt:n versiosta 4.5 lähtien sovelluskehittäjät voivat ladata ja asentaa Qt:n LGPL-lisenssillä, joka mahdollistaa myös kaupallisten sovellusten tekemisen ilman Qt:n lisenssimaksujen maksamista.

Vuonna 2008 Nokia osti Trolltechin ja muutti sen nimen Qt Softwareksi. Nokia on ilmoittanut tavoitteekseen hakea vauhtia Symbian-ohjelmistokehitykseen Qt:n avulla (Nokia, 2009). Vaikka Symbian onkin toistaiseksi maailman käytetyin matkapuhelin-alusta, sen sovelluskehitysympäristöä pidetään sekavana ja vaikeana hallita. Symbian-puhelinten osuus vähenee jatkuvasti, ja uudet tulokkaat, kuten Applen iPhone ja Android-alustan puhelimet, valtaavat alaa erittäin nopeasti. Nokia uskoo saavansa ohjelmistokehittäjät kiinnostumaan matkapuhelinsovellusten tekemisestä Qt:lla, sillä periaatteessa samojen Qt-sovellusten pitäisi toimia niin PC:llä kuin mobiililaitteessakin. On myös mahdollista, että Qt:n hankinta on vain nopea elvytys Symbianille ja koko alusta korvataan tulevaisuudessa esimerkiksi Linuxiin perustuvalla käyttöjärjestelmällä, jossa Qt on erittäin keskeisessä tehtävässä.

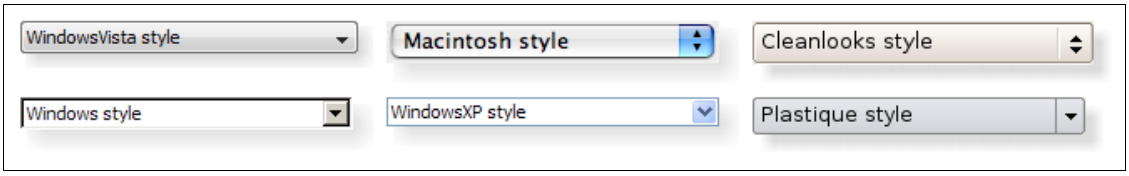
Koska Qt on C++-luokkakirjasto ja Qt-ohjelmistokehitystä tehdään pääasiassa C++:lla, on ainakin sen perusteet hallittava hyvin. Osata tulisi ainakin osoittimet, viitteet, operaattoreiden ylikuormitus, perintä ja muistinhallinta.

3.1.3 Missä Qt:ta käytetään?

Qt voi olla monille täysin tuntematon tekniikka, mutta sillä on tehty useita tavallisille

käyttäjille suunnattuja sovelluksia. Muutamana esimerkkinä voidaan mainita Opera-selain, Skype, Google Earth sekä jo mainittu KDE-työpöytäympäristö. Kaikista näistä ohjelmista on saatavilla ilmaisversio usealle alustalle. Ainoastaan KDE on suunnattu alun perin Unix/Linux-käyttäjille, mutta sen Windows- ja Mac-versiot ovat nykyään jo varsin käyttökelpoisia.

Kaikki edellä mainitut ohjelmat ovat alustariippumattomia, ja vaikka niiden käyttöliittymä on toiminnaltaan yhtenäinen alustasta riippumatta, niiden ulkonäkö vaihtelee hieman, sillä graafiset Qt-ohjelmat käyttävät oman alustansa natiiveja kirjastoja hyödykseen (Qt 4 .4 Whitepaper 2008, 40). Kuviossa 1 on esitetty Qt:n pudotusvalikko (QComboBox) eri alustoilla.



Kuvio 1. Qt:n pudotusvalikon ulkoasu eri alustoilla (mukaillen Qt Reference Documentation, 2008)

3.2 Qt ja Java yhdessä

Qt:lle on olemassa myös Java-tuki, Qt Jambi. Java-taustaiselle ohjelmoijalle Qt Jambin pitäisi olla helppo tapa omaksua Qt:n luokkakirjasto ja sen tarjoamat ohjelmointitekniikat. Tässä opinnäytetyössä ei kuitenkaan käsitellä Qt Jambia, sillä tarkoitus on valmistautua Qt:n käyttämiseen Nokian matkapuhelimissa, jolloin Qt Jambi ei todennäköisesti ole käytettävissä. Tämän lisäksi mainittakoon, että Java-ohjelmoijien ei tulisi pelätä Qt-ohjelmointia C++:lla, sillä Qt:n sanotaan olevan erittäin helppo luokkakirjasto, joka tarjoaa valmiit ratkaisut esimerkiksi muistin siivoamiseen samaan tapaan kuin Java.

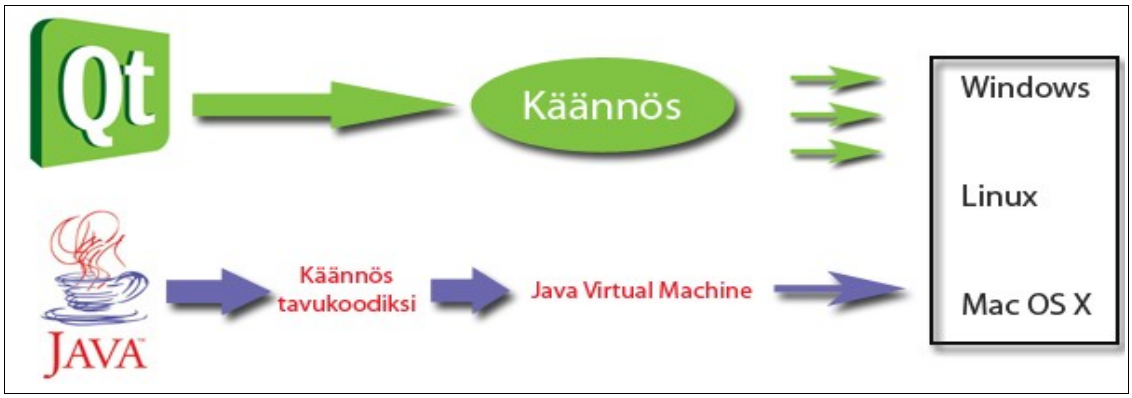
3.3 Alustariippumattomuus

Sekä Java että Qt ovat kieliä, joista käännetään binäärimuotoinen sovellus ajoa varten. Molemmat kielet tähtäävät myös alustariippumattomuuteen ja siirrettävyyteen. Alusta-

riippumattomuutta tavoitellaan kuitenkin täysin erilaisilla ratkaisuilla. Java-koodia ei käännetä suoraan alustalle natiiviksi binääriksi, vaan siitä käännetään tavukoodia (Haggar Peter, 2001). Tavukoodi on eräänlainen välivaihe käännökselle, sillä sitä tulkataan eteenpäin järjestelmälle. Javan työpöytäversiossa (Java SE, Java Standard Edition) tulkkaamisesta vastaa virtuaalikone nimeltä JVM eli Java Virtual Machine.

Java-kääntäjän muodostama tavukoodi on aina samanlaista käännettiinpä koodi millä alustalla tahansa. Vasta virtuaalikone tulkkaa käännöksen kullekin alustalle sopivaksi (Niemeyer & Knudsen 2005, 4). Virtuaalikone on ikään kuin alikäyttöjärjestelmä, sillä se hallitsee käyttöjärjestelmältä saatuja resursseja itsenäisesti ja jakaa niitä eteenpäin ohjelmilleen. Tästä syystä Javassa on useita vasta ajonaikana hyödynnettäviä ominaisuuksia, joita ei voida hyödyntää kielissä, joista käännetään staattiset binäärit (Niemeyer & Knudsen 2005, 6). Java-sovelluksia pidetään yleisesti myös turvallisina käyttää, sillä ne toimivat virtuaalikoneen sisällä omassa ympäristössään.

Myös Qt on alusta alkaen ollut alustariippumaton kehitysympäristö. C++ on kuitenkin puhtaasti käännettävä kieli. Kääntäjä tuottaa sen alustan binäärejä, jolla lähdekoodit käännetään ja Qt ei tässä suhteessa tarjoa mitään uutta: koodia ei tulkata, eikä siitä käännettä väliaikaista tavukoodia. Alustariippumattomuus saavutetaan siis tekemällä useampi käännös jokaiselle alustalle erikseen (Molkentin 2007, 19). Seuraavaan kuvaan on tiivistetty Qt:n ja Javan käännösprosessit lähdekoodeista valmiiksi ohjelmiksi.



Kuvio 2. Qt:n ja Javan avulla lähestytään alustariippumattomuutta erilaisista näkökulmista.

3.4 Kehitysympäristö

Qt-sovelluskehitystä varten on tarjolla useita työkaluja niin koodin kuin graafisten osuuksien tekemiseen. Qt-sovelluskehitystä varten SDK:n hankinta on ainoa pakollinen vaihe ja sen voi ladata Qt Softwaren sivuilta Windowsille, Linuxille ja Macille. Linux-käyttäjien kannattanee kuitenkin asentaa Qt SDK oman pakettienhallintansa kautta.

Qt Softwaren sivuilta saatavan SDK:n mukana tulee Qt-ohjelmointia varten kokonainen ohjelmointiympäristö (IDE, Integrated Development Enviroment), jonka nimi on Qt Creator. Qt Creator julkaistiin tämän opinnäytetyön kirjoituksen aikana Qt:n version 4.5 myötä (Products – Qt – A cross-platform application and UI framework, 2009). Se on siis suhteellisen uusi tuote, mutta siitä huolimatta Qt Softwaren sivujen mukaan suositelluin vaihtoehto Qt-sovelluskehitykseen, sillä sen avulla voi suunnitella ja tehdä ohjelmakoodin lisäksi graafiset komponentit WYSIWYG-tyylisen (What You See Is What You Get) editorin avulla samaan tapaan kuin Javassa käyttöliittymiä voi luoda Netbeansilla.

Lisäksi Qt Softwaren sivuilla tarjotaan mahdollisuus ladata Qt-lisäosa Eclipseen ja Visual Studioon. Lisäosan asennuksen jälkeen Qt-kehitys onnistuu myös näillä suosituilla ohjelmointiympäristöillä. Varsinkin Java-ohjelmoijille tuttu Eclipse voikin tarjota turvallisen tavan tutustua Qt-luokkakirjastoon tutun ympäristön siivittämänä. Tosin näiden lisäosien avulla ei ole mahdollista tehdä sovelluksen graafisia osuuksia muuten, kuin kirjoittamalla ne suoraan koodiksi. Tähän apua tarjoaa Qt Designer -niminen sovellus, jolla voi tehdä graafiset osuudet samaan tapaan kuin Qt Creatorilla. Toisin kuin Qt Creatorilla, Qt Designerilla ei kuitenkaan voi tehdä sovelluksen koodiosuuksia.

Mobres-TJ:tä luodessani kokeilin edellä mainitsemiani apuohjelmia nopeasti, mutta suurimman osan koodista olen kirjoittanut tekstieditorilla ja kääntänyt komentoriviltä. Apuohjelmat tuntuivat tekevän suuren osan kääntämiseen ja graafisten osuuksien toteutukseen liittyvästä työstä, joten niiden käyttäminen oikeassa tuotantoympäristössä on suositeltavaa. Mobres-TJ:tä tehtiin kuitenkin harjoituksen vuoksi, ja koin yhdessä Qt-tiimin kanssa saavani enemmän harjoitusta ja ymmärrystä Qt-sovelluskehitykseen tekemällä kaiken itse komentoriviltä käsin.

3.5 Qt-sovelluksen kääntäminen

Jos käytössä on jokin aiemmin tässä työssä mainittu ohjelmointiympäristö, ohjelman kääntäminen ja ajaminen tapahtuvat suoraviivaisesti ohjelmointiympäristön käyttöliittymän avulla. Tässä luvussa perehdytään Qt-sovelluksen kääntämiseen yksityiskohtaisemmin ja tutustutaan kääntämisessä tarvittaviin komentorivityökaluihin.

3.5.1 Kääntäjä ja sen käyttö

Qt:n SDK:n asennuksen mukana saadaan tarvittavat komentoriviohjelmat Qt-sovelluskehitystä varten. Lisäksi tarvitaan luonnollisesti C++-kääntäjä ja jokin tekstieditori koodin kirjoittamista varten.

Qt:n mukana tuleva qmake-ohjelma on tärkeimpiä apuvälineitä projektin kääntämistä varten. Java-ohjelmoijille todennäköisesti tuttuja työkaluja ovat Ant ja Maven, jotka huolehtivat projektin kääntämisestä xml-tiedostoon määriteltyjen ohjeiden mukaisesti. QMake ajaa saman asian Qt-projektissa: sen toiminta määritellään projektitiedostossa ja myöhemmin sen avulla luodaan projektille makefile, jossa varsinainen käännös on määritelty (Blanchette & Summerfield 2008, 594). Toisin kuin Maven, qmake ei lataa projektin riippuvuuksia verkosta, sillä kaikki moduulit ovat valmiina Qt:n SDK:ssa, ja ne otetaan käyttöön projektitiedostossa määrittelemällä. Qt:n ydin- ja käyttöliittymäkirjastoja ei tarvitse erikseen määritellä, sillä ne ovat jokaisessa projektissa mukana automaattisesti.

Projektitiedoston runkoa ei tarvitse kirjoittaa itse, sillä qmake luo sen, kun lähdekoodit sisältävässä kansiossa suoritetaan komento ”qmake -project”. Tällöin kansioon luodaan alustariippumaton projektitiedosto, jonka tiedostopääte on .pro. Tätä komentoa on tarkoitus käyttää vain, kun projektitiedostoa ei ole. Tällöin qmake luo seuraavan esimerkin (Koodiesimerkki 1) mukaisen projektitiedoston.

```
#############################################################
# Automatically generated by qmake (2.01a) Tue Nov 25 11:02:47 2008
#############################################################

TEMPLATE = app
TARGET =
DEPENDPATH += . lib widgets
INCLUDEPATH += . lib widgets

# Input
HEADERS += mainwindow.hh \
                widgets/login.hh \
SOURCES += main.cpp \
                mainwindow.cpp \
                widgets/login.cpp \
```

Koodiesimerkki 1. Mobres-TJ:n projektitiedoston runko

Seuraavaksi qmake suoritetaan antamalla sille parametriksi juuri luotu projektitiedosto, esimerkiksi ”qmake mobrestj.pro”, jolloin qmake luo käytetylle alustalle sopivan makefilen. Makefile tulee uusia aina, kun uusia moduuleja tai tiedostoja otetaan käyttöön. Kun makefile on luotu, ohjelma voidaan kääntää normaalisti make-komennolla. Windowsilla komentojen toimiminen suoraan komentokehotteesta vaatii ympäristömuuttujien asettamisen tai MinGW-ohjelmiston asentamisen koodin kääntämistä varten (Blanchette & Summerfield 2008, 5).

Qt on pohjimmiltaan C++-luokkakirjasto ja Qt-ohjelmistokehityksessä syntyvät tiedostot ovat samoja, mitä normaalissa C++-kehityksessä syntyy muutamaa lisäystä lukuun ottamatta, kuten projektitiedostoa ja erilaisia xml-resurssitiedostoja, joita esimerkiksi Qt Creatorilla ja Qt Designerilla voi tuottaa.

3.5.2 Mitä pinnan alla tapahtuu?

Tulevissa esimerkeissä tullaan huomaamaan, että Qt-koodi ei ole aina standardin C++:n mukaista. Tyypillisenä esimerkkinä tästä ovat luvussa 4.2.2 käsiteltävät signal-slot -mekanismit. Jotta Qt-koodi saadaan C++-kääntäjän ymmärtämään muotoon, täytyy apuna käyttää moc-apuohjelmaa (Meta-Object Compiler). Moc on oma ohjelmansa, mutta qmake kutsuu sitä tarvittaessa automaattisesti (Blanchette & Summerfield 2008, 22).

Nimestään huolimatta moc ei ole kääntäjä: käytännössä moc vain tutkii jokaisen luokan, joka on peritty Qt:n kantaluokaksi tarkoitetusta QObject-luokasta tai sen aliluokasta. Lisäksi luokissa tulee käyttää Q_OBJECT makroa, jotta moc tunnistaa luokan Qt-luokaksi. Moc etsii näistä luokista tietyt avainsanat, funktiot ja makrot, jotka se korvaa C++-kääntäjälle sopivalla koodilla ja kirjoittaa uudet lähdekooditiedostot moc_-alkuisiin tiedostoihin. Kyseiset tiedostot löytyvät projektikansiosta ja niitä voi vapaasti tutkia, mutta niiden muokkaamisesta ei ole mitään hyötyä, sillä tiedostot kirjoitetaan kääntämisen yhteydessä uudelleen (Molkentin 2007, 56).

Q_OBJECT -makro on pakollinen, kun luokassa käytetään signal-slot -mekanismeja, tai esimerkiksi lokalisoinnissa käytettävää tr()-funktiota. Sen tehtävä on yhdessä moc:n kanssa luoda C++-kääntäjälle sopivat toteutukset kyseisistä tekniikoista ja funktioista. Qt:n dokumentaation mukaan Q_OBJECT -makroa tulee käyttää jokaisessa QObjectin perivässä luokassa, vaikka kyseisiä tekniikoita ei suoraan käytettäisikään (Qt Reference Documentation, 2008). Makro otetaan käyttöön kirjoittamalla Q_OBJECT heti luokan esittelyn alkuun, kuitenkin esittelylohkon sisäpuolelle. Tässä opinnäytetyössä on myöhemmin koodiesimerkki, jossa kyseistä makroa käytetään.

4 Ohjelmointitekniikat

Tässä luvussa vertaillaan niitä Javan ja Qt:n ohjelmointitekniikoita, jotka itse miellän ohjelmoinnin perusteiksi. Lisäksi luvussa käsitellään Qt:n ainutlaatuisia mekanismeja, joilla C++-ohjelmointia on pyritty yksinkertaistamaan. Luvussa käsitellään lisäksi widgettejä, joilla tarkoitan tässä opinnäytetyössä yksittäisiä käyttöliittymäkomponentteja, joita yhdistelmällä käyttöliittymä rakennetaan.

Widget-käsitteen lisäksi tässä luvussa käsitellään rajapintoja. Jotta lukijan ei olisi vaikea hahmottaa millaisesta rajapinnasta on kyse, viitataan tässä työssä Javan rajapintaan (Interface) ohjelmointirajapinta-käsitteellä, kun rajapinta-käsitteellä tarkoitetaan tässä työssä mitä tahansa rajapintaa.

4.1 Hello Word!

Perinteisesti ohjelmointikielen tai -ympäristön opiskelu alkaa ”Hello World”-ohjelman tekemisellä. Alla on esimerkkiohjelman koodi, joka suoritettaessa luo ikkunan, jossa lukee ”Terve maailma. Tämä on Qt:ta!”. Koska Qt on nimenomaan graafisten sovellusten tekemiseen tarkoitettu kehitysympäristö, tehdään tulostus suoraan graafiseen ikkunaan, ei komentoriville.

```
#include <QApplication>
#include <QLabel>

int main(int argc, char* argv[])
{
   QApplication app(argc, argv);
   QLabel* label = new QLabel(”Terve maailma. Tämä on Qt:ta”);
   label->show();
   return app.exec();
}
```

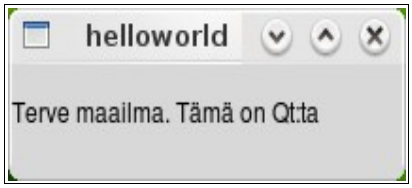
Koodiesimerkki 2. Hello World!

Koodi on syntaksiltaan normaalia C++:aa. Lähdekooditiedostoon sisällytetään QApplication- ja QLabel-luokat, jotka ovat ohjelmassa tarvittuja Qt-luokkakirjaston luokkia. Qt ohjelma alkaa aina QApplication-luokan olion luomisesta, joka saa parametriksi main()-funktioon tulleet mahdolliset komentoriviparametrit. Sen jälkeen luodaan QLabel-

luokan olio, johon viitataan osoittimella. Käyttämissäni lähteissä ensimmäiset QLabel-luokan oliot luodaan hieman eri tavalla: lähteessä Blanchett & Summerfield (2008) QLabel luodaan dynaamisesti heap-muistiin ja siihen viitataan osoittimen avulla, kun taas lähteessä Molkentin (2007) QLabel luodaan suoraan stack-muistiin.

Itse päädyin luomaan kaikki oliot heap-muistiin ja viittaamaan niihin osoittimilla, sillä projektissa luotavat Qt-luokkakirjaston oliot voivat olla kooltaan hyvinkin suuria, jolloin stack-muisti voi loppua kesken. Näin varaudutaan myös Qt:n Symbian-versioon, jossa stack-muistin määrä on erittäin pieni. Lisäksi olioiden luominen stack-muistiin ei ole aina turvallista. Javassa stack-muistiin ei ole edes mahdollista luoda olioita.

Seuraavaksi ohjelmassa kutsutaan QLabel-luokan olion show()-funktiota, joka piirtää widgetin ruudulle. Lopuksi vielä kutsutaan alussa luodun app-olion exec()-funktiota, mikä aloittaa tapahtumienkäsittelyn. Näin ohjelma jää odottamaan käyttäjän syötteitä ja muita tapahtumia (Blanchette & Summerfield 2008, 4). Ohjelmaa suoritettaessa näytölle piirretään seuraavan kuvion (Kuvio 3) mukainen ikkuna.



Kuvio 3. Ensimmäinen Qt-ohjelma

4.2 Tiedon välittäminen luokalta toiselle

Qt:n mekanismi tiedon välittämiseen luokalta toiselle on ainutlaatuinen ja kyseinen tekniikka käsitellään käyttämissäni aineistossa ensimmäisten kappaleiden joukossa. Lisäksi se on oivallinen esimerkki siitä, miten Qt laajentaa normaalia C++-koodia ja näistä syistä se käsitellään myös tässä opinnäytetyössä näin varhaisessa vaiheessa.

4.2.1 Takaisinkutsut ja kuuntelijat

Takaisinkutsuja (callback) on käytetty oliokielten syntymästä lähtien. Takaisinkutsulla tarkoitetaan nimensä mukaisesti jonkin toisen olion tai komponentin kutsumista. Kun luokan funktiossa kutsutaan toisen luokan funktiota, jossa kutsutaan taas ensimmäisen

luokan funktiota, puhutaan takaisinkutsusta. Takaisinkutsua voidaan käyttää esimerkiksi tilanteessa, jossa luokan täytyy pystyä ilmoittamaan toiselle luokalle jonkin metodin suorituksen olevan valmis. Tekniikan käyttäminen ei edellytä siis mitään muuta, kuin että kaksi luokkaa ovat tietoisia toisistaan. Javassa takaisinkutsut toteutetaan yleensä ohjelmointirajapinnoilla (Niemeyer & Knudsen 2005, 171).

Toinen yleinen tapa välittää tietoa luokalta toiselle on kuuntelija-luokkien käyttäminen. Ne odottavat valmiustilassa tiettyä tapahtumaa, ja sen tapahtuessa toimivat tietyllä tavalla. Javassa esimerkiksi MouseListener-ohjelmointirajapinta on yleinen käyttöliittymissä käytetty hiirieleiden kuuntelija (Niemeyer & Knudsen 2005, 582). Kyseisellä rajapinnalla on useita metodeita esimerkiksi hiiren painikkeen painallusta ja vapautusta varten. Ohjelmoijan tehtäväksi jää määrittää, mitä edellä mainituissa tilanteissa tapahtuu.

4.2.2 Signaalit ja slotit

Qt-ohjelmoinnissa edellä mainittuja tekniikoita ei tarvitse käyttää, sillä Qt hyödyntää niin sanottua signal-slot -mekanismia. Oliot voivat lähettää signaaleja ilmoittaessaan tapahtumasta tai tilansa muutoksesta. Signaalit yhdistetään sopiviin sloteihin tai toisiin signaaleihin connect(...)-funktiolla. Pääasiassa sloteja kutsutaan automaattisesti, kun siihen yhdistetty signaali lähetetään, mutta niitä voidaan kutsua kuten tavallisia funktioitakin. Slotit toteutetaan kuin normaalit metodit, mutta signaaleille ei kirjoiteta toteutusta lainkaan (Blanchette & Summerfield 2008, 20).

Yhdistämiseen tarvitaan signaalin lähettävän ja vastaanottavan olioiden osoittimet ja signaalin ja slotin nimet. Tarvittaessa useita signaaleja voidaan yhdistää yhteen slotiin ja toisinpäin, sekä signaaleita voidaan yhdistää signaaleihin. Koodiesimerkissä 3 on esimerkki connect(...)-funktion käytöstä. Esimerkissä closeButton-painikkeen clicked()-signaali yhdistetään kyseessä olevan luokan accept()-slotiin. Näin saadaan aikaan toiminnallisuus, jossa painiketta napsauttamalla kyseessä oleva ikkuna suljetaan.

```
//Yhdistetään close-painikkeen napsauttaminen dialogin sulkemiseen.
connect(closeButton, SIGNAL(clicked()), this, SLOT(accept()));
```

Koodiesimerkki 3. Signaalin ja slotin yhdistäminen connect(...)-funktiolla

Connect(...)-funktiolla on siis neljä parametriä: lähettäjä, lähettäjän signaali, vastaanottaja ja vastaanottajan slot. Lähettäjän ja vastaanottajan tulee olla osoittimia QObjectiin tai sen aliluokkaan. This-osoittimen käyttö on sallittua ja usein jopa erittäin suotavaa.

Signaalin ja slotin välinen yhteys voidaan poistaa (Molkentin 2007, 35). Poistamista tarvitaan harvoin, sillä Qt purkaa yhteyden olion tuhoutuessa. Purkaminen tapahtuu kutsumalla disconnect(...)-funktiota samoilla parametreillä, millä connect(...)-funktiotakin kutsuttiin.

Hyvin usein signaalien pitää välittää arvoja sloteille. Esimerkkinä mainittakoon Mobres-TJ:hin toteutettu signaali kirjautumistietojen välittämiseen. Kyseisen slotin kutsuminen ei riitä, vaan slotille tulee välittää ainakin käyttäjän syöttämät käyttäjätunnus ja salasana. Signaalien ja slotien parametrien määrittely tapahtuu kuin minkä tahansa metodin parametrien määrittely.

Qt:n valmiilla widgeteillä on yleensä omat signaalinsa, joita lähetetään automaattisesti widgetin arvon tai tilan muuttuessa (Molkentin 2007, 38). Esimerkiksi Qt:n tekstikentällä (QLineEdit) on textChanged(const QString& text)-signaali, joka lähetetään tekstikentässä tapahtuvan muutoksen jälkeen automaattisesti. Signaalin parametrinä välittyy tekstikentässä oleva teksti, joka voidaan vastaanottaa ja käsitellä slotissa (Qt Reference Documentation, 2008).

Signaalin ja siihen yhdistetyn slotin parametrien tietotyyppien tulee olla samat, sillä tyyppimuunnoksia ei ole mahdollista tehdä. Tarpeen vaatiessa luokka täytyy periyttää ja halutusta slotista pitää toteuttaa oma versio, jossa parametrin tyyppi otetaan huomioon. Lisäksi parametrien määrä voi aiheuttaa ongelmia: slot voi vastaanottaa vähemmän parametrejä, kuin signaali lähettää. Parametrilistan ylijäämät parametrit yksinkertaisesti jäävät huomiotta ja niillä ei ole vaikutusta ohjelmaan. Tästäkin huolimatta tietotyyppien tulee olla samoja, ja parametrien keskinäinen järjestys vaikuttaa lopputulokseen (Molkentin 2007, 38). Jos signaali välittää kokonaisluvun ja liukuluvun, se voidaan hyvin kytkeä slotiin, joka vastaanottaa yhden kokonaisluvun. Virheellinen kytkentä tapahtuu vasta, kun kyseistä signaalia yritetään kytkeä slotiin, jonka ensimmäinen parametri on liukuluku.

Signaalien ja slotien virheellinen yhdistäminen ei aiheuta käännösvirhettä johtuen niiden ajonaikaisesta käsittelystä. Ohjelma voi kuitenkin suorituksen aikana tulostaa konsoliin varoituksia, mikäli connect(...)-funktiota on käytetty väärin (Molkentin 2007, 39). Ohjelma ei kaadu, mutta signaali ja slot jäävät yhdistämättä ja ohjelma ei toimi halutusti.

Tähän mennessä on käsitelty luokkakirjaston tarjoamia valmiita signaaleja ja sloteja. Qt:ssa on kuitenkin mahdollisuus määritellä niitä myös itse. Lisäksi signaalien lähettämiseen voidaan vaikuttaa. Omat signaalit ja slotit määritellään luokkien esittelytiedostoissa omien lohkojensa alla seuraavan esimerkin (Koodiesimerkki 4) mukaisesti. Tässä esimerkissä on määritelty Mobres-TJ:n kirjautumisikkunassa tarvittavat signaalit ja slotit.

```
signals:
    //Signaali, jossa lähetetään käyttäjän antamat kirjautumistiedot.
    void sendLogin(const QString& host, const QString& db, const QString& user,
    const QString& passwd);

private slots:
    //Slot, jota kutsutaan kun login-painiketta napsautetaan.
    void onLoginButtonPressed();
```

Koodiesimerkki 4. Mobres-TJ:n kirjautumiseen tarvittavat signaalit ja slotit

Luokan toteutukseen slotit määritellään kuin ne olisivat tavallisia funktioita. Kirjoitetut slotit saadaan käyttöön yhdistämällä ne sopiviin signaaleihin ja lähettämällä signaaleita emit-operaattorilla. Koodiesimerkissä 4 esitellyt signaalit ja slotit yhdistetään toisiinsa seuraavasti: login-painiketta napsauttamalla painike lähettää pressed()-signaalin, joka yhdistetään esimerkissä esiteltyyn onLoginButtonPressed()-slotiin. Koodiesimerkissä 5 on osa kyseisen slotin toteutuksesta: slotissa tarkastetaan, että käyttäjänimi- ja salasanakenttä eivät ole tyhjiä ja lähetetään sendLogin(...)-signaali. Lopuksi widget suljetaan.

```
        ...
        //Lähetetään signaali ja suljetaan ikkuna.
        emit sendLogin("192.168.11.10", "mobres", username, password);
        this->close();
```

Koodiesimerkki 5. Signaalin lähettäminen ohjelmoijan toimesta

Signaaleja ja sloteja käsiteltäessä ja niitä yhdistellessä ohjelmoijan tulee varmistua siitä, että ohjelman ei ole mahdollista mennä signaali-slot -silmukkaan. Tälläisiä tilanteita syntyy todennäköisemmin, mikäli signaaleja lähetetään niihin yhdistetyistä sloteista. Tällöin signaalin lähettäminen kannattaa suojata sopivalla if-tarkastuksella, jolla varmistetaan, että signaalin lähettäminen on todellakin tarpeellista.

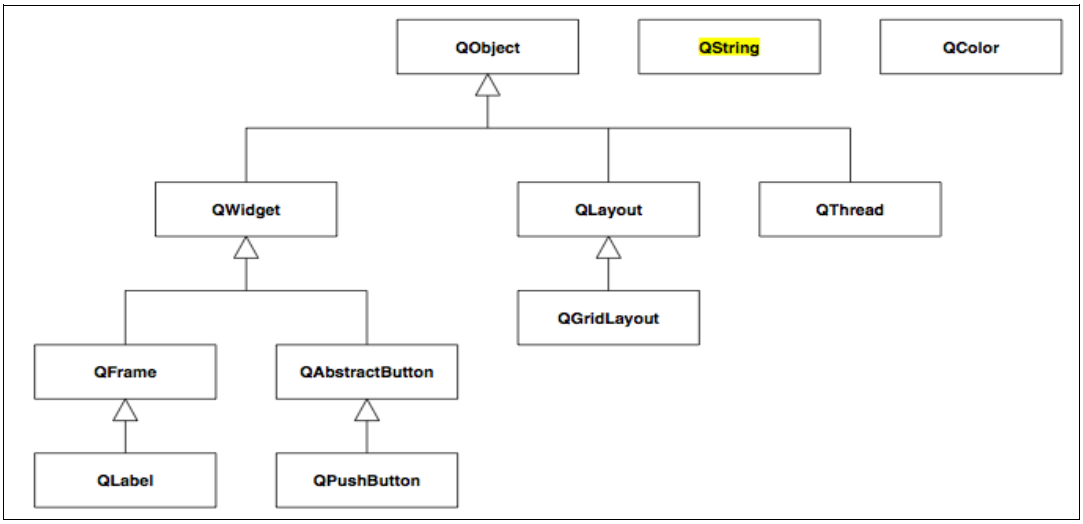
4.3 Luokkahierarkia ja periytyminen

Luokkahierarkia ja perintä ovat Javassa ja C++:ssa hyvinkin erilaisia. Javassa ne ovat yksinkertaisia ja osittain automatisoituja asioita, kun taas C++:ssa niistä pitää huolehtia itse. Tässä luvussa tutustaan Qt:n luokkahierarkiaan, tärkeimpiin widget-luokkiin ja käsitellään perinnän tärkeyttä Qt-ohjelmoinnissa.

4.3.1 Qt:n perintähierarkia

C++:sta Javaan siirtyvä ohjelmoija saattaa ihmetellä, miksi luokilla on metodeita, joita ohjelmoija ei ole itse kirjoittanut. Jokaisella luokalla on esimerkiksi toString()-metodi, joka tulostaa oliosta saadun hajautusarvon, jos metodia ei ole kuormitettu. Tämä johtuu siitä, että Javassa kaikki luokat perivät automaattisesti Object-luokan ja saavat käyttöön sen tarjoamat metodit. C++:ssa taas luokalla ei ole kantaluokkaa, jos ohjelmoija ei ole sellaista määritellyt.

Qt:ssa tilanne on enemmän Javaa muistuttava; hyvin suuri osa Qt:n luokkakirjaston luokista perii QObject-nimisen luokan. Perintä voi tapahtua pitkänkin hierarkian kautta kuten Javassakin. Toisin kuin Javassa, Qt:ssa tähän sääntöön on muutama poikkeus (Kuvio 4): esimerkiksi QString-luokka ei peri mitään luokkaa, sillä se ei tarvitse signaaleja tai sloteja eikä myöhemmin esiteltävää lapsi-vanhempi -mekanismia.

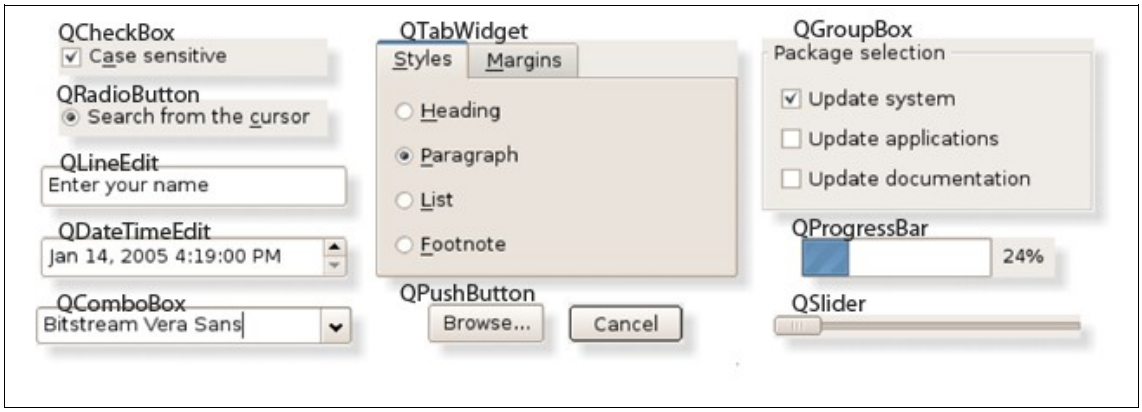


Kuvio 4. Qt:ssa kaikki luokat eivät peri kantaluokkaa (Molkentin 2007, 41)

Qt-ohjelmoijan ei ole teoriassa pakko periyttää mitään luokkaa kirjoittaessaan omia luokkiaan, kun taas Javassa periyttämiseen käytettävän extends-määren käyttämättä jättäminen aiheuttaa Object-luokan perimisen. Vaikka Qt-ohjelmoinnissa QObject-luokan perintää ei ole automatisoitu, tulee ohjelmoijan muistaa periyttää jokin Qt-luokka: QObject on hyvä vaihtoehto, jos parempaa kohdetta ei löydy. Ilman perintää Qt:n edut jäävät täysin hyödyntämättä, eikä tehtyä luokkaa voida oikeastaan edes pitää Qt-luokkana.

4.3.2 Tärkeimmät widgetit

Suuri osa Qt:n luokkakirjastosta on erilaisia widgettejä, joiden yhdistelmiin monimutkaisemmatkin käyttöliittymät yleensä perustuvat. Tärkeimmät widgetit on esitelty hyvin Qt 4.4 Whitepaperissa (Kuvio 5).



Kuvio 5. Tärkeimmät Qt:n widgetit (mukaillen Qt 4.4 Whitepaper 2008, 5)

Kuviossa 5 on mielestäni keskeisimpiä widgettejä, joita Qt-ohjelmoijan tulisi osata käyttää ilman jatkuvaa dokumentaation selaamista. Yleensä graafiset käyttöliittymät koostuvat juuri näistä widgeteistä. Hyvin pitkälle pääsee jo normaalin tekstirivin tarjoavalla QLineEditillä, tavallisen painikkeen tarjoavalla QPushButtonilla, valintaruudun tarjoavalla QCheckBoxilla ja valintapainikkeen tarjoavalla QRadioButtonilla. Kuvassa ei ole eritelty QLabel-widgettiä, joka on normaali nimiö.

4.3.3 Widgettien ja dialogien luominen periyttämällä

Perinnällä on erittäin suuri merkitys Qt-ohjelmoinnissa. Omia widgettejä tehdään täydentelemällä ja yhdistelemällä Qt:sta löytyviä luokkia. Lisäksi useat luokat ovat jätetty hieman avonaisiksi, jotta ohjelmoija voi halutessaan periyttää ne ja toteuttaa puuttuvat osat haluamallaan logiikalla. Toisin sanoen useista luokista löytyy virtuaalisia metodeita.

Qt-ohjelmoija tarvitsee usein graafisten elementtien yhdistämistä yhdeksi toimivaksi widgetiksi, jolloin tulee periyttää QWidget-luokka (Blanchette & Summerfield 2008, 107). Useista widgeteistä koostetaan yksi widget laittamalla ne uuden widget-luokan ilmentymämuuttujiksi. Koodiesimerkissä 6 on osa Mobres-TJ:n kirjautumisikkunan muodostajan toteutuksesta. Esimerkissä alustetaan widgetit, joista Login-widget koostuu.

```
Login::Login(QWidget* parent, const QString debug) : QWidget(parent)
{
   //Alustetaan tekstikentät
   username_line = new QLineEdit(this);
   username_line->setFixedWidth(260);
   password_line = new QLineEdit(this);
   password_line->setFixedWidth(260);
   password_line->setDisabled(true);
   password_line->setEchoMode(QLineEdit::Password);
   //Alustetaan nimiöt
   username_label = new QLabel(tr("Username"));
   password_label = new QLabel(tr("Password"));
   //Alustetaan login-painike
   login_button = new QPushButton(tr("login"));
   login_button->setFixedWidth(260);
   ...
```

Koodiesimerkki 6. Osa Login-luokan muodostajasta

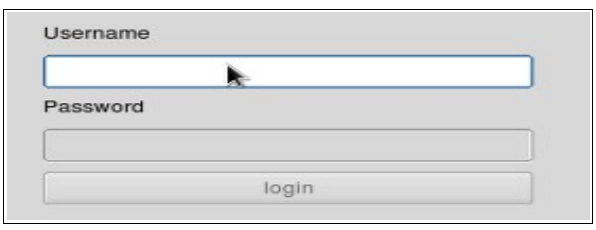
Widgettien alustaminen ei yksin riitä, vaan ne pitää asetella lomakkeeksi. Ilmentymämuuttujaksi tarvitaan siis vielä sopiva QLayout-luokan aliluokka. QLayout-luokalla on useita aliluokkia, mutta Mobres-TJ:ssä niistä käytettiin lähinnä kahta luokkaa: QLayoutista on periytetty QBoxLayout-niminen luokka, josta on edelleen periytetty luokat QVBoxLayout, jolla widgetit asetellaan päällekäin ja QHBoxLayout, jolla widgetit asetellaan vierekkäin (Qt Reference Documentation, 2008). Näiden kahden luokan avulla ja niitä yhdistelemällä voidaan widgetit asetella lähes miten tahansa, ja tästä syystä en itse koskaan tarvinnut muita QLayoutin aliluokkia. Koodiesimerkissä 7 esitetään edellisessä koodiesimerkissä alustettujen widgettien taittaminen.

```
...
//Luodaan pystysuuntainen layout ja keskitetään se vaakatasolla
login_layout = new QVBoxLayout(this);
login_layout->setAlignment(Qt::AlignHCenter);
login_layout->addStretch();

//Asetellaan widgetit layouttiin
login_layout->addWidget(username_label);
login_layout->addWidget(username_line);
login_layout->addWidget(password_label);
login_layout->addWidget(password_line);
login_layout->addWidget(login_button);
login_layout->addStretch();
...
```

Koodiesimerkki 7. Login-luokan widget-muuttujien taittaminen

Widgettejä lisätään QLayout-luokkaan tai sen aliluokkiin addWidget(...)-metodilla. Lisäksi koodiesimerkissä kutsutaan QLayout-luokan addStretch()-metodia kahdesti. Kyseinen metodi varaa tilaa QLayoutille mahdollisimman paljon: tässä yhteydessä sitä on käytetty widgetin keskittämiseen. Sitä voitaisiin myös käyttää widgettien asettelemiseen eri puolelle QLayouttia. Koodiesimerkissä taitettu Login-widget näyttää kuvion 6 mukaiselta.



Kuvio 6. Sisäänkirjautuminen Mobres-TJ:hin.

Login-widget on tarkoituksella luotu yksinkertaiseksi taittaa, mutta jos nimiöt haluttaisiin asetella tekstikenttien viereen, tarvitsisi QVBoxLayoutin sisään asetella myös QHBoxLayout-luokan olioita addLayout()-metodilla, joten koodiesimerkin pituus kasvaisi ainakin kaksinkertaiseksi. Tästä syystä Qt Creatorin tai Qt Designerin käyttäminen on enemmän kuin suotavaa varsinkin monimutkaisempien käyttöliittymien suunnittelussa, mutta kuten mainittu, Mobres-TJ:n tapauksessa widgetit on kirjoitettu koodiksi harjoituksen vuoksi.

Jos tarkoitus on täydentää jotakin valmista widgettiä, eikä koostaa uusia, kannattaa periä suoraan täydennettävä widget. Näin voidaan luoda esimerkiksi tekstikenttä, johon voidaan syöttää vain numeroita. Periytetty luokka tarvitsee ilmentymämuuttujaksi osoittimen sopivaan lisätoiminnallisuutta tarjoavan luokan olioon. Edellä mainitussa esimerkkitapauksessa lisätoiminnallisuus saataisiin QRegExpValidator-luokasta.

Widgettiä, joka avautuu omaan ikkunaansa, kutsutaan dialogiksi. Kun halutaan tehdä dialogi, tulee luokka periyttää QDialog-luokasta. Tällöin aliluokalla on kaikki dialogin ominaisuudet, ja kutsuttaessa show()-funktiolla se avautuu omaan ikkunaansa (Blanchette & Summerfield 2008, 19). Äskeinen esimerkki voidaan muuttaa dialogiksi erittäin helposti vaihtamalla yliluokka QWidgetistä QDialogiksi. Lisäksi ohjelmoijan tehtäväksi jää lisätä halutessaan painike dialogin sulkemista varten. Mobres-TJ:tä varten luotiin erilaisia dialogeja esimerkiksi uusien asiakkaiden ja pakettien lisäämistä varten.

4.3.4 Mahdollisuus moniperintään

Moniperintä tarkoittaa luokan periyttämistä useammasta kuin yhdestä kantaluokasta. Java-ohjelmoijille termi on usein tuttu, vaikka Javassa ei aitoa moniperintää olekaan. Java on luotu helpoksi kieleksi, ja moniperintä aiheuttaa väärin käytettynä ongelmia. Javassa on kuitenkin mahdollisuus käyttää ohjelmointirajapintoja normaalisti vain moniperinnän mahdollistamien toiminnallisuuksien toteuttamiseen.

Javassa ohjelmointirajapinnat saavat sisältää vain staattisia vakiomuotoisia muuttujia (static, final) ja pelkkiä metodeiden esittelyitä: metodit pitää toteuttaa ohjelmointirajapinnan toteuttavassa luokassa. Vaikka ei ole mahdollista periä kuin yksi luokka, ohjelmointirajapintoja voidaan toteuttaa kuitenkin useampia.

Moniperintää pidetään yleisesti vaikeana hallita ja tästä syystä sitä ei ole Javassa. Vaikka itse en ole vielä tarvinnut moniperintää Qt-ohjelmoinnissa, kannattaa muistaa, että se on täysin sallittua, vaikka Qt:n lähes puumainen luokkahierarkia ja laaja luokkakirjasto poistavat lähes kokonaan tarpeen käyttää moniperintää.

4.4 Muistinhallinta

Java-ohjelmoijat yleensä pelkäävät C++-ohjelmoinnin aloittamista kuullessaan kauhutarinoita muistivuodoista, joita etsittiin viikkotolkulla. C++-ohjelmoijien suusta ei taas aina kuule kovin mairittelevia kuvauksia Javan automaattisesta roskienkeruusta. Tässä luvussa esitellään lyhyesti molemmat ääripäät ja tutustutaan Qt:n muistinhallintaan molemmista näkökulmista.

4.4.1 Javan automaattinen roskienkeruu

Ainoastaan Javaan tutustunut ohjelmoija harvoin kiinnittää huomiota muistin siivoamiseen tai sen käsittelyyn, sillä Java hoitaa kaiken automaattisesti. Javassa ei ole oikeita osoittimia, joten mahdollisuutta viitata tiettyyn kohtaan muistia ei ole. Lisäksi Javassa on roskienkeruu (garbage collector), jonka vuoksi luotuja olioita ei tarvitse itse poistaa. Roskienkeruu on automaattinen järjestelmä, joka pitää kirjaa muistin sisällöstä ja käytöstä poistuneista olioista (Niemeyer & Knudsen 2005, 13). Kun olioon ei enää viitata, roskienkeruu osaa vapauttaa sen muistista. Ohjelmoijalle tämä tarkoittaa sitä, että käytöstä poistuneille olioille ei tarvitse tehdä mitään. Javan roskienkeruu toimii taustalla ja sen ajo tapahtuu yleensä tarpeen vaatiessa sopivassa kohdassa ohjelman suoritusta siten, että kerralla saadaan aikaan mahdollisimman paljon. Roskienkeruun ajo näkyy selkeästi Java-ohjelmissa tarvittavan muistin määrän laskemisena ja ohjelman suorituksen hetkellisenä hidastumisena, jos olioita on paljon.

4.4.2 C++:ssa siivoamisesta pitää huolehtia itse

C++:sta roskienkeruu puuttuu ainakin toistaiseksi. Tästä syystä C++:ssa luokkiin kirjoitetaan yleensä hajotin (destructor, destruktori). Hajotinta kutsutaan automaattisesti oliota tuhottaessa delete-operaattorilla, kuten muodostinta kutsutaan oliota luodessa new-operaattorilla. Hajottimen kirjoittaminen on erityisen tärkeää, kun luokassa on

ilmentymämuuttujia, jotka alustetaan new-operaattorilla. Tällöin tilaa varataan heap-muistista, jota C++:ssa ei siivota automaattisesti. Pääsääntöisesti kaikki new-operaattorilla luodut oliot pitää vapauttaa muistista itse (Hietanen 2004, 245).

Hajotinta voidaan käyttää luonnollisesti vain ilmentymämuuttujien vapauttamiseen. Jos jossakin lohkossa esitellään uusi osoitinmuuttuja ja sen osoittamaan muistipaikkaan alustetaan jotakin, täytyy delete-operaattoria käyttää ennen lohkosta poistumista. Jos näin ei tehdä, osoitinmuuttuja vapautetaan stack-muistista lohkosta poistuttaessa, mutta sen viittaamassa muistipaikassa olevaa sisältöä ei vapauteta. Osoittimen vapautuessa menetetään ainut keino tämän muistipaikan vapauttamiseen ja ohjelmassa on muistivuoto.

Delete-operaattorin käyttäminen ei suinkaan tarkoita olion tuhoamista ja muistin tyhjentämistä, vaan kyseisen osan vapauttamista muuhun käyttöön. Sen jälkeen kyseisessä kohdassa muistia saattaa olla mitä vaan, joten mahdollisiin osoitinmuuttujiin, jotka osoittavat juuri vapautettuun osaan muistia, kannattaa alustaa NULL.

C++:n muistinhallinta on siis täysin ohjelmoijan vastuulla ja kokemattomalle ohjelmoijalle muistinhallinta voi osoittautua erittäin haastavaksi, sillä muistivuotoja voi olla hankala etsiä ja testata. Pienissä muutaman olion ohjelmissa muistinhallinnan voi unohtaa, mutta satoja tai tuhansia olioita käsittelevissä sovelluksissa muistinhallinnan unohtaminen johtaa laajoihin muistivuotoihin, jolloin ohjelma kuluttaa tahattomasti muistia ja käyttäytyy mahdollisesti epävakaasti.

4.4.3 Qt huolehtii osittain muistin siivoamisesta

Qt pitää osittain huolta muistin siivoamisesta ja turhista olioista. QObject-luokka sisältää toiminnallisuuden, jonka avulla oliot muodostavat hierarkian lapsiolioistaan (child-parent mechanism). Ohjelmoijan tulisi välittää QObject-luokan muodostajaan parametrina osoitin johonkin toiseen QObject-luokan tai sen aliluokan olioon. Tästä oliosta tulee uuden olion vanhempi (parent). Näin oliot muodostavat puumaisen rakenteen, jossa yhdellä oliolla on useita lapsia joilla voi edelleen olla omia lapsiolioita. QObject-luokan olioilla on siis tiedossa sekä oma vanhempansa että kaikki lapsiolionsa. Muodostajan parametrin oletusarvo on 0, joten parametriä ei ole pakko välittää.

Mobres-TJ:n päänäkymän navigointivalikko on toteutettu useilla QTreeWidgetItem-luokan olioilla ja sopii siksi hyvin esimerkiksi: kyseinen luokka käyttää lapsi-vanhempi -mekanismia puumaisten valikkojen luomiseen. Siksi QTreeWidgetItem sopii hyvin esimerkiksi kansiorakenteiden mallintamiseen, ja Mobres-TJ:tä varten luodussa navigointivalikossa on käytetty samaa menetelmää. Muodostajassa (Koodiesimerkki 8) alustetaan yli kymmenen QTreeWidgetItem-luokan oliota ja niiden muistiosoite säilötään ilmentymämuuttujiin. Jokainen olio saa muodostimessa vanhemman.

```
Navigation::Navigation(QWidget* parent) :
   QTreeWidget(parent)
{
   //Asetetaan näkymään yksi sarake ja piilotetaan otsikko
   this->setColumnCount(1);
   this->setHeaderHidden(true);

   //Luodaan QTreeWidgetItemit
   database = new QTreeWidgetItem(this);
   database->setText(0, tr("Database"));
   deliveries = new QTreeWidgetItem(database);
   deliveries->setText(0, tr("Deliveries"));
   clients = new QTreeWidgetItem(database);
   clients->setText(0, tr("Clients"));
   users = new QTreeWidgetItem(database);
   users->setText(0, tr("Users"));
  …
```

Koodiesimerkki 8. Vanhemman välittäminen oliota alustettaessa

Ensimmäinen alustettu QTreeWidgetItem saa vanhemmaksi Navigation-luokan olion this-osoittimen avulla. Loput esimerkissä luodut QTreeWidgetItem-luokan oliot saavat tämän ensimmäiseksi luodun olion vanhemmakseen ja näin QTreeWidgetItemit muodostavat puumaisen hierarkian. Kuviossa 7 on syntynyt navigaatiovalikko.

Kuvio 7. Mobres-TJ:n Navigation-widget

Mitä hyötyä lapsi-vanhempi -mekanismin käytöstä on sitten muistin siivoamisen kannalta? Kun Qt:ssa poistetaan olio delete-operaattorilla aivan kuin C++:ssa, poistetaan olion kaikki lapset ja niiden lapset, kunnes yhtään lapsiolioita ei ole jäljellä (Blanchette & Summerfield 2008, 28). Tästä syystä Navigation-luokka ei tarvitse ollenkaan hajotinta. Hajottimelle on siis käyttöä ainoastaan kun heap-muistiin on luotu olioita, joilla ei ole vanhempaa tai ne eivät ole QObjectin aliluokkia.

Lapsiolioita hyödyntävästä tekniikasta on muitakin etuja. Kaikilla QObjectista perityillä luokilla on metodit vanhemman ja lapsien palauttamiseen. Esimerkiksi parent()-metodi palauttaa osoittimen olion vanhempaan, jota taas voidaan käyttää hyväksi esimerkiksi signaalien ja slotien yhdistämisessä.

4.5 Tietorakenteet

Java-ohjelmoijan harvoin tarvitsee miettiä taulukoita käyttäessään mitä oikeastaan tapahtuu. C++-ohjelmoija yleensä ymmärtää, että taulukkomuuttuja on vain osoitin taulukon ensimmäiseen soluun. Taulukko-operaattori [] kääntyy siis *-operaattoriksi automaattisesti. Tästä syystä taulukoita voidaan iteroida normaalisti kokonaislukujen avulla kuten Javassa, tai käyttää *-operaattoria. Lisäksi taulukoille täytyy niiden luomisen yhteydessä varata muistia yleensä heap-muistista, sillä tarvittava muistin määrä voi olla hyvinkin suuri. Taulukoiden käsittely on siis aidosti muistiavaruuden käsittelyä, joten taulukot käyttäytyvät samalla tavalla käytettiin Qt:ta tai ei.

Qt:n luokkakirjasto sisältää kuitenkin useita monipuolisia ja dynaamisia tietorakenteita erilaisiin tilanteisiin ja tarkoituksiin. Ne ovat geneerisiä ja toimivat kaikilla alustoilla samalla tavalla. Lisäksi tavallisia taulukoita käytettäessä ei voitaisi hyödyntää Qt:n lapsi-vanhempi mekanismia, joten muistin siivoamisesta pitäisi huolehtia itse. Mobres-TJ:tä tehdessäni en tarvinnut Qt:n tietorakenteita. Tein siitä huolimatta muutaman harjoituksen, sillä tietorakenteiden osaaminen kuuluu mielestäni minkä tahansa ohjelmointikielen tai -ympäristön perusteisiin. Lisäksi Qt:n tietorakenteiden iterointi tarjoaa mielenkiintoisia näkökulmia Java-ohjelmoijille.

4.5.1 Tietojäsenten keskinäiseen järjestykseen perustuvat tietorakenteet

Qt:n oman dokumentaation mukaan QList<T> on yleensä paras valinta yleiseksi tietorakenteeksi. Sisäisesti tietorakenne toimii kuin taulukko, joten sen avulla voidaan suorittaa erittäin nopeita hakuja. Sitä voidaan iteroida iteraattorin avulla tai käyttää taulukko-operaattoria ja kokonaislukuja alkioiden palauttamiseen. QListin loppuun lisääminen on erittäin nopeaa, ja se suoriutuu Qt:n dokumentaation mukaan erittäin nopeasti elementtien lisäämisestä listan keskiosaan, kun alkioita on alle tuhat kappaletta (Qt Reference Documentation, 2008).

QListiin voidaan lisätä elementtejä <<-operaattorilla tai append(const& T value)-funktiolla. Elementtien poistaminen tapahtuu removeFirst()-, removeLast()-, tai removeAt(int i)-funktiolla. Ensimmäisen elementin saa palautettua first()-funktiolla ja viimeisen last()-funktiolla, mutta helpoin tapa elementtien palauttamiseen on käyttää taulukko-operaattoria. Koodiesimerkissä 9 luodaan yksinkertainen kokonaislukuja säilövä QList.

```
//Luodaan QList ja laitetaan sinne luvut 1,200 ja -3.
QList<int> list;
list << 1<< 200 << -3;
//Luetaan QListin toinen alkio muuttujaan.
int x = list[1];
```

Koodiesimerkki 9. QList-tietorakenteen alustus ja käsitteleminen

QListille on tarjolla vaihtoehtoja. Kun tarvitaan taulukko, jonka koon täytyy mukautua erilaisiin tarpeisiin, voidaan käyttää QVector<T>-luokkaa. QVector<T> toimii kuin minkä tahansa muun ohjelmointikielen vektori: se on kokonsa puolesta dynaaminen ja sen loppuun on helppo lisätä uusia elementtejä, kun taas alkuun tai keskelle lisääminen on raskas operaatio, sillä uudelle elementille tilaa tehtäessä koko vektorin loppuosaa pitää siirtää. Javassa Vector-luokkaa pidetään hitaana, koska sitä voi käyttää turvallisesti myös säikeiden kanssa. Suositeltavampaa on yleensä käyttää taulukkolistaa (ArrayList) (Niemeyer & Knudsen 2005, 367).

QVectorille voidaan myös esittelyn yhteydessä antaa koko, tai jättää se antamatta. Jos koko annetaan, voidaan QVectoriin lisätä elementtejä kuin taulukkoon []-operaattorilla. Jos kokoa ei anneta, tulee elementtejä lisätä append(...)-funktiolla, jolloin vektorin

kokoa kasvatetaan tarpeen vaatiessa. QVectoria käydään läpi taulukko operaattorilla, kuten QListiakin.

Qt tarjoaa myös QLinkedList<T> -nimisen rakenteen, joka vastaa Javan linkitettyä listaa. Sen käyttäminen on suositeltavaa, jos uusia elementtejä lisätään jatkuvasti listan keskivaiheille. QLinkedListiin lisätään elementtejä samalla tavalla kuin QListiinkin. QLinkedList<T> ei kuitenkaan tue []-operaattoria, joten iteraattorin käyttö on pakollista tietorakenteen läpi käymiseksi koodiesimerkin 10 mukaisesti.

```
//Luodaan QLinkedList ja laitetaan sinne luvut 1 ja 3.
QLinkedList<int> list;
list << 1 << 3;

//Oho.. luku 2 unohtui välistä. Lisätäänpä se listaan lukujen 1 ja 3 väliin.
QLinkedList<int>::iterator iter = list.find(3);
lista.insert(iteraattori, 2);
```

Koodiesimerkki 10. QLinkedList-tietorakenteen alustaminen ja iterointi

4.5.2 Tietorakenteiden iterointi

Qt-ohjelmoinnissa tietorakenteita on mahdollista iteroida kahdella tavalla. Voidaan käyttää tehokkaita C++ Standard Libraryn (STL) iterointiluokkia ja -tekniikoita tai vaihtoehtoisia hieman helppokäyttöisempiä Java-iteraattoreita. Jokaiselle iteroitavalle tietorakenteelle on yhteensä neljä iteraattoria, kaksi Java- ja kaksi STL-iteraattoria. Toinen iteraattoreista on aina vain luku -tyyppiä ja toisella on mahdollista muokata tietorakenteen jäseniä. Mutable-sana iteraattorin nimessä viittaa aina iteraattoriin, jolla voidaan myös muokata tietorakennetta.

Iteraattorityypit ovat käyttörajapinnoiltaan samankaltaisia, mutta niiden toiminnassa on silti eroja. Java-iteraattori osoittaa aina kahden tietojäsenen väliin ja STL-iteraattori osoittaa suoraan tietojäsentä. Tärkeää on muistaa, että Java-iteraattoreita käytettäessä poistamiseen tarkoitettu remove()-funktio poistaa aina elementin, jonka yli ollaan viimeksi iteroitu (Blanchette & Summerfield 2008, 277). Poistamisen kohde riippuu siis siitä, ollaanko iteroitu eteenpäin vai “peruutettu”, vaikka iteraattori olisikin samassa kohdassa. Koodiesimerkistä 11 havaitaan, että Java-tyylisen iteraattorin käyttörajapinta on identtinen oikean Java-iteraattorin kanssa.

```
//Luodaan lista ja listaiteraattori.
QList<int> list;
list << 1 << 2 << 3;
QListIterator<int> iter(list);
//Iteroidaan listaa
while( iter.hasNext() )
{
   int temp = i.next();
   ...
}
```

Koodiesimerkki 11. Qt:n Java-tyylisen iteraattorin käyttörajapinta

Jokaisella tietorakenteella, jolla on Java-iteraattorit, on myös kirjoittamiseen ja lukemiseen tarkoitetut STL-iteraattorit. Nämä iteraattorit osoittavat aina suoraan tietorakenteen elementtiin. STL-iteraattorit käyttävät selkeämmin hyväkseen osoittimia. Tietorakenteessa voidaan liikkua ++- ja -– -operaattoreilla, ja tietojäsenten sisältö saadaan *-operaattorilla koodiesimerkin 12 mukaisesti.

```
//Luodaan lista ja listaiteraattori
QList<int> list;
list << 1 << 2 << 3;
QList<int>::iterator iter = list.begin();
//Iteroidaan listaa
while( iter != list.end() )
{
   int x = *iter;
   ...
   ++iter;
}
```

Koodiesimerkki 12. Qt:n STL-iteraattorin käyttörajapinta

Javassa on versiosta 1.5 lähtien ollut mahdollisuus käyttää foreach-silmukkaa tietorakenteiden läpikäymiseen. C++:sta kyseinen silmukkarakenne puuttuu, mutta Qt-ohjelmoija voi sitä halutessaan käyttää. Kun silmukkaan mennään, tietorakenteesta otetaan kopio (Blanchette & Summerfield 2008, 280). Tehdyt muutokset eivät siis tallennu tietorakenteeseen, eivätkä alkuperäiseen tietorakenteeseen mahdollisesti samaan aikaan tapahtuvat muutokset vaikuta silmukkaan (Koodiesimerkki 13).

```
//Luodaan lista
QList<QString> list;
list << ”Ensimmäinen” << ”Toinen” << ”Kolmas”;
//Iteroidaan foreachilla
foreach(i, list)
{
   //Ei vaikuta tietorakenteeseen: EI TOIMI!
   I = ” ”;
}
```

Koodiesimerkki 13. Foreach-silmukka Qt:ssa

4.5.3 Hajautukseen perustuvat tietorakenteet

Qt tarjoaa lisäksi myös hajautusarvoon ja avain-arvo -pareihin perustuvia tietorakenteita. Tietorakenteeseen tallennetaan avain, jonka avulla itse tallennettava arvo haetaan. Tärkeimmät hajautusarvoon perustuvat tietorakenteet Qt:ssa ovat QMap<K, T> ja QHash<K, T>, joissa K on avaimen tietotyyppi ja T tallennettavan arvon tietotyyppi.

Molempien luokkien käyttörajapinta on lähes samanlainen: tietojäseniä voidaan asettaa []-operaattorilla tai insert(...)-funktiolla. Taulukko-operaattoria voidaan käyttää myös tiedon noutamiseen, mutta tämä ei ole suositeltavaa, sillä jos haetaan avainta, jota ei ole olemassa, sellainen luodaan ja se saa tyhjän arvon (Blanchette & Summerfield 2008, 282). Tämän välttämiseksi tietoa tulisi hakea aina value()-funktion avulla.

Koodiesimerkissä 14 esitetään edellä mainitut tavat käsitellä näitä kahta tietorakennetta. Vaikka esimerkissä käytetäänkin QMapia, samat funktiot ja toimintaperiaatteet toimivat myös QHashilla.

```
//Luodaan QHash, josta tehdään englanti-suomi -sanakirja.
QHash<QString, QString> dict;
//Tallennetaan tietoa kahdella eri tavalla.
dict[”car”] = ”auto”;
dict.insert(”boat”, ”vene”);
//Haetaan tietoa sanakirjasta.
QString word = dict.value(”car”);
```

Koodiesimerkki 14. Avain-arvo -parien tallentaminen QHash-tietorakenteeseen

Avaimen tietotyypiksi kannattaa valita kokonais- tai liukuluku, merkkijono tai osoitin. Avaimet saadaan palautettua QList-tietorakenteessa keys()-funktiolla ja arvot values()-funktiolla. Tietorakenteista on peritty myös versiot, joissa yhdelle avaimelle voidaan tallentaa useampia arvoja: QMultiMap ja QMultiHash (Blanchette & Summerfield 2008, 283).

Kahdesta edellä mainitusta tietorakenteesta QHash on hieman nopeampi ja se onkin suositeltavampi vaihtoehto sanakirja-tyylisen tietorakenteen luomiseen, jos tietorakennetta iteroitaessa avainten järjestyksellä ei ole väliä (Blanchette & Summerfield 2008, 283). Jos avaimet täytyy saada järjestykseen, QMap on parempi valinta.

Avain-arvo -pareja voidaan iteroida Java-tyylisillä iteraattoreilla. Java-iteraattoreiden next()- ja previous()-funktiot palauttavat olion, jossa on sekä avain, että sen arvo. Avaimen saa oliosta key()-funktiolla ja arvon value()-funktiolla.

4.6 Säikeet

Sovelluksissa tapahtuu usein aikaa vieviä ja mahdollisesti raskaita toimenpiteitä. Käyttäjälle tämä näkyy ohjelman käyttöliittymän tai muun toiminnan pysähtymisenä, ja ohjelma lakkaa vastaamasta. Näissä tilanteissa aikaa vievät operaatiot tulisi laittaa omaan säikeeseensä. Säie on ohjelman oma haara, jota suoritetaan samaan aikaan muiden säikeiden kanssa. Säieohjelmointi on laaja alue ja tässä luvussa on tarkoitus antaa vain suppea kuvaus siitä, miten moniajo toteutetaan Qt:n avulla. Säikeistämisellä on kuitenkin myös omat ongelmansa, sillä ne toimivat erilaisissa tilanteissa erilaisilla tavoilla. Esimerkiksi yksiytimisellä prosessorilla useamman säikeen suorittaminen voi olla hidasta verrattuna moniydinprosessoriin.

4.6.1 Säikeen luominen periyttämällä

Javassa säikeitä varten on olemassa Runnable-ohjelmointirajapinta, joka sisältää ainoastaan run()-nimisen metodin esittelyn. Kyseinen ohjelmointirajapinta toteutetaan ja run-metodin runkoon määritellään kaikki toiminnallisuus, joka halutaan suorittaa omassa säikeessään. Syntyneestä luokasta luotu olio annetaan Thread-luokan oliolle parametriksi muodostajassa. Sen jälkeen kutsutaan tämän olion start()-funktiota, mikä

käynnistää säikeen ja sen toimintaa ja ajastusta voidaan hallita esimerkiksi sleep()-metodilla (Niemeyer & Knudsen 2005, 251).

Rajapinnan toteuttamisen lisäksi voidaan uusi säie ottaa käyttöön perinnän avulla. Javassa Thread-niminen luokka toteuttaa Runnable()-rajapinnan jo valmiiksi ja ohjelmoija voi halutessaan yksinkertaisesti periyttää ja kuormittaa sen run-metodin. Säikeen voi sitten käynnistää suoraan kutsumalla syntyneen luokan start()-metodia (Niemeyer & Knudsen 2005, 254). Vaikka Thread-luokan periminen on hieman yksinkertaisempaa, rajapinnan toteuttaminen on kuitenkin suositeltavampaa: Javassa ei ole moniperintää, joten jos luokka perii Thread-luokan, se ei voi periä enää mitään muuta luokkaa.

Qt:n tapa säikeistää sovellus muistuttaa Javan perimistapaa. Säikeistettävän toiminnallisuuden sisältävä luokka periytetään QThread-luokasta ja sen run()-funktio kuormitetaan ja säie käynnistetään start()-funktiolla. Mikäli säikeen toiminta halutaan pysäyttää, tulee ohjelmoijan itse huolehtia sopivan boolean-tyyppisen muuttujan ja sen asetus- ja palautusmetodien ohjelmoinnista. Oletuksena run()-metodi suoritetaan kokonaisuudessaan, eikä sen keskeyttäminen onnistu luotettavasti. QThreadin terminate()-metodi kyllä lopettaa säikeen, mutta ei kumoa säikeen tekemiä muutoksia tai siivoa muistia (Qt Reference Documentation, 2008).

Alkuperäinen tarkoitukseni oli käyttää säikeitä Mobres-TJ:ssä tietokantahakujen suorittamiseen, mutta loppujen lopuksi koko tietokantaosuus toteutettiin käyttäen Qt:n valmiita MVC-arkkitehtuuriin perustuvia tietokantaluokkia. Näin säikeille ei jäänyt ohjelmassa tarvetta, eikä niitä hyödynnetä ollenkaan. Koodiesimerkissä 15 on kuitenkin esimerkkisäikeen esittelytiedosto malliksi omien säikeiden luomiseen.

```
class Thread : public QThread
{
   Q_OBJECT
   public:
      Thread();
      void stop();

   protected:
      void run();

   private:
      volatile bool stopped();
};
```

Koodiesimerkki 15. Ohjelma voidaan säikeistää periyttämällä QThread-luokka

Thread-luokalla on muodostaja, jossa stop-ilmentymämuuttuja voitaisiin alustaa false-arvolla. Säikeen käynnistäminen start()-funktiolla aloittaa run()-metodin suorittamisen. Mikäli kyseisessä metodissa on silmukkarakenne, kannattaa silmukassa tutkia stopped-muuttujan arvoa, jotta säikeen suoritus voidaan turvallisesti lopettaa kutsuttaessa stop()-funktiota, jonka tehtävä on asettaa stopped-muuttujan arvoksi true. Ennen run()-funktion päättymistä kannattaa kuitenkin stopped-muuttujalle antaa arvoksi false, jos säiettä halutaan käyttää myöhemmin uudestaan.

4.6.2 Säikeen tilan tarkastelu

Säikeen tila voidaan palauttaa useammalla metodilla: jos säie on lopettanut toimintansa, isFinished()-metodi palauttaa true-arvon ja isRunning()-metodi false-arvon. Palautus-arvot ovat päinvastaisia, jos säikeen tilaa tarkastellaan isRunning()-metodilla. Lisäksi säie lähettää signaaleja, kun sen tila olennaisesti muuttuu. Säikeen käynnistyessä lähetetään started()-signaali, lopettaessa finished()-signaali ja terminate()-metodilla lopettaessa terminated()-signaali (Qt Reference Documentation, 2008). Näin säikeen toiminta voidaan yhdistää muuhun toiminnallisuuteen normaalisti slotien ja connect(...)-funktion avulla. Luonnollisesti voidaan määritellä myös omia signaaleja, joita voidaan lähettää säikeen sisällä.

4.6.3 Säikeiden synkronointi

Säikeiden ajastamista ja suorittamista ohjaa käyttöjärjestelmä ja kun käytössä on useampia säikeitä, käyttöjärjestelmä voi suorittaa osan jostakin säikeestä ja sen jälkeen vaihtaa suoritettavaa säiettä. Ohjelman seuraavalla suorituskerralla säikeiden saama suoritusaika voi olla erilainen. Näin ollen muuttujat, joita useampi säie käsittelee samaan aikaan voivat aina hetkellisesti saada ohjelman toiminnan kannalta virheellisiä arvoja. Mikäli kahden säikeen tulee voida luotettavasti käsitellä yhteisiä muuttujia, täytyy säikeet synkronoida (Molkentin 2007, 241).

Qt:ssa on tarjolla useita luokkia kuten QMutex, jonka avulla osa koodista voidaan lukita lock()-funktiolla ja vapauttaa unlock()-funktiolla. QMutex-luokalla voidaan estää säikeen pääsy haluttuun osaan koodia, joten se ei sovi kaikkiin tarkoituksiin: jos muuttujien arvo pitää pystyä lukemaan kahdesta säikeestä samaan aikaan, mutta kirjoittaminen ei saa onnistua, QMutex-luokka voi olla huono valinta synkronointiin. Synkronointiin on onneksi tarjolla muitakin luokkia (QReadWriteLock, QSemaPhore, QWaitCondition), joihin synkronointia toteuttaessa kannattaa tutustua (Blanchette & Summerfield 2008, 343).

4.6.4 Säikeistämisen vaihtoehdot

Säikeitä ei tarvitse aina käyttää: joskus ohjelmoitessa nousee esille mahdollisuus, että käyttäjien toimiin ei pystytä vastaamaan, koska koodissa on esimerkiksi jokin raskas silmukka, voi tilanteen ratkaista ilman säikeitäkin. Kun Qt-ohjelma aloitetaan, kutsutaan QApplication-luokan exec()-funktiota, joka käynnistää silmukan, jossa kutsutaan tasaisin väliajoin QApplication::processEvents()-funktiota, jonka tehtävä on tutkia mitä ohjelmassa tapahtuu ja välittää sopivat kutsut oikeille olioille (Blanchette & Summerfield 2008, 175).

QApplication::processEvents()-funktiota voidaan kutsua myös ohjelmakoodissa. Kutsun kirjoittaminen esimerkiksi raskaisiin normaalisti säikeistämistä vaativiin silmukkarakenteisiin saa ohjelman tutkimaan ja käsittelemään ohjelman tai käyttäjän aiheuttamia tapahtumia. Näin ohjelma jatkaa vastaamista käyttäjän kutsuihin raskaiden silmukkarakenteiden suorittamisen aikana ja säikeitä ei tarvita.

4.7 Qt:n moduulit ja niiden käyttöönotto

Käyttöliittymäkirjasto on vain osa Qt:n laajaa luokkakirjastoa: tarjolla on moduuleja verkkoyhteyksiä, XML-dataa, SQL-tietokantoja, 2D-, 3D-, ja vektorigrafiikkaa sekä multimediasisältöä varten. Qt:n ydin- ja käyttöliittymämoduulit ovat jokaisessa projektissa automaattisesti käytettävissä, mutta muut moduulit täytyy ottaa käyttöön määrittelemällä ne projektitiedostoon. Jokaisessa moduulissa on noin 10-30 luokkaa.

Projektitiedostoon käytettävät moduulit määritellään seuraavan esimerkin (Koodiesimerkki 16) mukaisesti. Esimerkissä projektiin on tuotu QtWebkit- ja QtSql-moduulit. Vaikka moduulien nimissä on Qt-etuliite, niitä ei käytetä projektitiedosssa.

```
QT += sql \
    webkit
```

Koodiesimerkki 16. Moduulien tuominen Qt-projektiin.

QtSql mahdollistaa yhteydet yleisimpiin SQL-tyyppisiin tietokantoihin. Tämän moduulin kulmakivinä voidaan pitää QSqlDatabase- ja QSqlQuery-luokkia, joiden avulla tietokantayhteys avataan ja SQL-lauseet suoritetaan (Qt Reference Documentation, 2008). Moduuli tukee käytetyimpiä SQL-tietokantoja ja vaikka moduuli sisältää luokkia, joilla tietokantaan päästään käsiksi ilman SQL-lauseiden käyttöä, sen käyttö vaatii perustietämyksen relaatiotietokannoista.

Toinen esimerkissä projektiin tuotu moduuli on QtWebkit. Kyseinen moduuli sisältää valmiin selainmoottorin, joka mahdollistaa normaalisti vain selaimella tarkasteltavan sisällön tuomisen Qt-ohjelmiin. Moduuli osaa renderöidä XHTML- ja HTML- ja SVG-dokumentteja ja osaa käsitellä CSS-tyylitiedostoja ja JavaScriptiä (Qt Reference Documentation, 2008). Moduuli on helposti omaksuttava, sillä se sisältää alle kymmenen luokkaa, joiden avulla voi luoda esimerkiksi selaimen tai web-sivujen luontiin tarkoitetun WYSIWYG-editorin. Tärkeimpänä osana koko moduulia voidaan pitää QWebView-luokkaa ja sen load(...)-funktiota, jolla web-sisältö voidaan ladata osaksi käyttöliittymää. Luokka on peritty QWidgetistä, joten sen voi näyttää view()-funktiolla (Qt Reference Documentation, 2008).

5 Toteutus käytännössä

Kuten opinnäytetyöni alkupuolella luvussa 2 kirjoitin, toteutin Qt-harjoituksena Mobres-TJ-nimisen sovelluksen. Koska Qt-sovelluskehittäminen oli itselleni täysin tuntematonta ja tarkoituksena oli harjoitella erilaisia tekniikoita, ryhdyin välittömästi ohjelmointityöhön. Suuri osa koodista tuli kirjoitettua useampaan otteeseen ja virheitä tuli tehtyä alkupuolella usein, mutta nämä haasteet olivat tärkeitä osaamisen kehittymiselle.

5.1 Sovitut käytännöt

Osittain Qt-harjoituksia varten luotiin sisäverkkoon oma sisäinen wiki, johon kirjoitettiin ohjeita ja tietoa muille Qt-tiimissä työskenteleville. Tällä hetkellä wikissä on kattavat ohjeet Qt:n asennukseen ja käyttöönottoon, sekä luentoja Qt:n eri osa-alueista. Wikiin ei kirjoitettu mitään projektikohtaista, ja kaikki ohjelmointivinkit kirjoitettiin yleispäteviksi ohjeiksi.

Projektikohtaista lokia ylläpidettiin versionhallinnassa. Versionhallinnaksi valittiin tiimille etukäteen tuttu ilmainen Subversion, lyhenteeltään SVN. Versionhallintaan lisättiin pienetkin ohjelmakoodiin tehdyt muutokset ja ne kommentoitiin hyvin, joten versionhallintalokista näkee projektissa esiintyneet työvaiheet ja ongelmat.

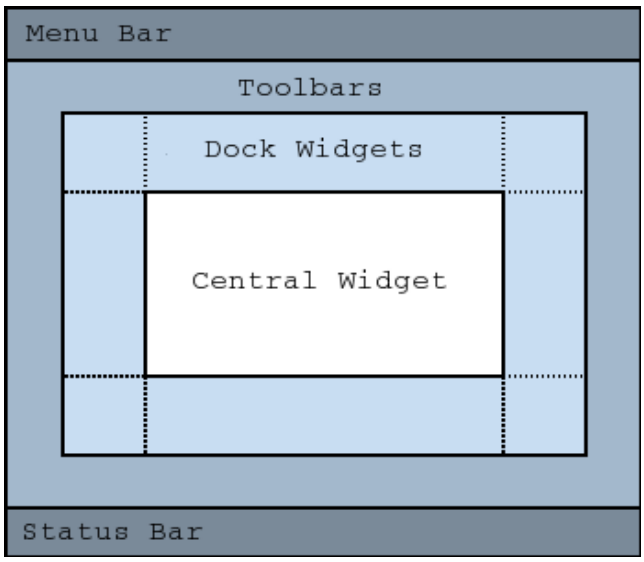
Koko tiimi käytti koodia kirjoittaessaan Tampereen teknillisen yliopiston ohjelmistotekniikan laitoksen C++-tyyliopasta. Qt-itseopiskelumateriaaliksi tuotetun koodin haluttiin olevan yhtenäistä, mutta omaa tyyliopasta ei haluttu luoda. Lisäksi kyseinen tyyliopas on todettu hyväksi ja se vastaa hyvin nykypäivän vaatimuksia. Tyyliopas on noin kaksikymmentä sivua pitkä ja sen tyylisäännöt ovat perusteltu hyvin. Tyyliopas löytyy osoitteesta http://www.cs.tut.fi/~oliot/kirja/tyyliopas/.

5.2 Käytetyt ratkaisut

Vaikka tarkoituksena oli kokeilla erilaisia Qt-luokkia ja -tekniikoita mahdollisimman monipuolisesti, kaikkea ei luonnollisesti voi sovittaa samaan sovellukseen. Tässä luvussa käsitellään niitä yksittäisiä ratkaisuja, joihin Mobres-TJ perustuu.

5.2.1 Ohjelman päänäkymä

Graafiset käyttöliittymät harvoin koostuvat yhdestä widgetistä tai joukosta erillisiä widgettejä, vaan käyttöliittymässä on myös valikkorivi, työkalurivejä, ohjelman tilasta kertova tilarivi ja mahdollisesti muita widgettejä. Qt:ssa edellä kuvatun kaltainen ohjelman päänäkymä luodaan QMainWindow-luokasta periyttämällä. QMainWindow-luokalla on valmis asettelunsa, johon on mahdollista lisätä QToolbar- QStatusbar- QMenuBar- ja QDockWidget-luokkien olioita (Qt Reference Documentation, 2008). Näiden luokkien oliot taitetaan automaattisesti osaksi QMainWindow-luokan asettelua kuvion 8 mukaisesti.



Kuvio 8. QMainWindow-luokassa widgetit taitetaan automaattisesti (Qt Reference Documentation, 2008)

Mobres-TJ:ssä QMainWindow periytettiin MainWindow-nimiseksi luokaksi. Jaoin luokan muodostajan useampaan erilliseen apufunktioon, sillä tilarivin, työkalurivin ja valikoiden alustamisen vuoksi muodostajasta olisi tullut valtavan pitkä. Alustuksen yhteydessä muodostajassa kuitenkin kutsutaan näitä apufunktioita ja luodaan Mobres-TJ:n päänäkymä valikkoineen, sekä työkalu- ja tilariveineen.

Alustuksen loppuvaiheessa luodaan myös QSplitter-luokan olio keskuswidgetiksi (central widget, katso Kuvio 8). QSplitter-luokan avulla QWidget-olioita voidaan asetella päällekkäin tai vierekkäin ja niiden kokoa ja käyttämää tilaa voidaan muuttaa vetämällä hiirellä niiden saumakohdasta. QSplitter-luokan avulla useampi widget

saadaan helposti aseteltua keskuswidgetiksi ilman QLayout-luokan käyttämistä. Koodiesimerkissä 16 on esitetty MainWindow-luokan muodostaja.

```
MainWindow::MainWindow(QWidget* parent):
   QMainWindow(parent)
{
   //Apufunktiot valikoiden ja tilarivien luomiseen.
   createActions();
   createMenus();
   createStatusBar();

   //Asetetaan ikkunan koko ja otsikko.
   setMinimumSize( 950, 600 );
   setWindowTitle(tr("Mobres-TJ"));

   //Luodaan QSplitter, johon luodaan ja lisätään Navigation-widget.
   splitter = new QSplitter(Qt::Horizontal, this);
   navi = new Navigation(this);
   navi->setMaximumWidth(200);
   splitter->addWidget(navi);

   //Luodaan Login-widget ja asetetaan se QSplitteriin
   l = new Login(this);
   splitter->addWidget(l);

   //Asetetaan QSplitter keskuswidgetiksi
   setCentralWidget(splitter);
}
```

Koodiesimerkki 16. MainWindow-luokan muodostaja

Muodostajassa kutsutaan kolmea apufunktiota ja asetetaan ikkunan koko ja otsikko. Sen jälkeen luodaan QSplitter-luokan olio, johon lisätään oliot tehdyistä Navigation- ja Login-luokkien olioista. Kirjautumisen jälkeisestä tilasta on kuvio tämän luvun lopussa.

5.2.2 MySQL-tietokantayhteys

Tietokanta suunniteltiin yhdessä koko tiimin voimin mahdollisimman yksinkertaiseksi, mutta kuitenkin niin, että samaa tietoa ei kannasta löytyisi useampaan kertaan. Tämä näkyy esimerkiksi siten, että osoitteille on oma taulunsa, eikä niitä tallenneta esimerkiksi paketit-tauluun. Paketit-taulussa on sarake, jossa on osoite-tauluun viittaava yksilöllinen id-numero, jonka avulla taulujen välille saadaan yhteys.

Alun perin oli tarkoitus kirjoittaa avustajaluokka Mobres-TJ:n tietokantayhteyksiä varten, jonka tehtävänä oli avata tietokantayhteys ja suorittaa hakuja palauttaen tulokset sopivassa tietorakenteessa. Qt:ssa on kuitenkin valmiita MVC-arkkitehtuurin (model-view-controller) mukaisia malli- ja näkymäluokkia. MVC-arkkitehtuurissa ohjelma on jaettu kolmeen eri kerrokseen: malli vastaa tallennettavasta tiedosta, näkymä sen esitystavasta ja käyttöliittymästä sekä ohjain huolehtii käyttäjältä tulevista käskyistä ja hallitsee mallia ja muokkaa sitä (Niemeyer & Knudsen 2005, 558).

QSqlRelationalTableModel on Qt:n valmis malliluokka, joka osaa muodostaa tietokannan tauluista datamallin ottaen huomioon myös taulujen väliset relaatiot. MVC-mallin mukaisesti tälle mallille voidaan luoda yksi tai useampi tapa esittää se, ja tässä tapauksessa käytin Qt:n valmista luokkaa nimeltä QTableView. Näin tietokannasta saadaan helposti taulukossa esitettävää tietoa, eikä SQL-lauseita tarvitse kirjoittaa ollenkaan, mikä edistää edelleen alusta- ja ympäristöriipumattomuutta. Koodiesimerkissä 17 on osia createPackagePanel()-funktiosta, joka luo taulukon tietokannassa olevista paketeista. Koodista on jätetty pois suuri osa, jota en ole katsonut oleelliseksi tässä yhteydessä.

```
void DeliverySummary::createPackagePanel()
{
   //malli, johon haetaan tietoa useasta taulusta
   packageModel = new QSqlRelationalTableModel(this);
   packageModel->setTable("paketit");
   //luodaan relaatio:
   //paketit-taulun lähetti-kenttään haetaan id:n sijasta lähetin nimi.
   packageModel->setRelation(Package_Messenger,
         QSqlRelation("lahetit", "id", "nimi"));
   //luodaan muita relaatioita
   ...

   //lajitellaan tieto kuvauksen mukaan
   packageModel->setSort(Package_Description, Qt::AscendingOrder);
   //annetaan tiedokannasta haetulle sarakkeelle parempi otsikko
   packageModel->setHeaderData(Package_Description, Qt::Horizontal,
                        tr("Description"));
   //luodaan näkymä, asetetaan näkymälle malli
   packageView = new QTableView(this);
   packageView->setModel(packageModel);
   ...
```

Koodiesimerkki 17. Pakettien noutaminen tietokannasta

Esimerkissä luodaan QSqlRelationalTableModel-luokan olio, joka osaa käyttää olemassa olevaa tietokantayhteyttä suoraan. Datamalli muodostetaan paketit-taulusta, jolle laaditaan relaatioita muihin tauluihin setRelation(...)-funktiolla. Esimerkissä parametrinä on paketit-taulun lähetti-sarake ja QSqlRelation-luokan olio, joiden avulla paketit-taulusta luodaan relaatio lähetit-tauluun. QSqlRelation luokka saa omassa muodostajassaan taulun ja sarakkeen nimen, johon relaatio muodostetaan, sekä sarakkeen, joka näytetään käyttäjälle.

Näkymä luodaan kutsumalla QTableView-luokan olion setModel(...)-funktiota, jolle parametriksi annetaan juuri luotu malli. Alla on vielä kuva (Kuvio 9) luodusta taulukosta, joka ylläolevassa esimerkissä luotiin.

	From	To	Priority	Description	Situation	Messenger
1	Hämeenkatu 5 B	Mäkipäänkatu 28-30	3	Osoite varmistettava	hukattu	teppo
2	Hämeenkatu 5 B	Pinninkatu 24	3	Särkyvää!	hukattu	teppo
3	Mäkipäänkatu ...	Hämeenkatu 5 B	3	Toimitettava nopeasti!	noudettu	timo
4	Mäkipäänkatu ...	6701 Collins Ave...	3	Ulkomaan paketti	asiakkaalla	seppo

Kuvio 9. QSqlRelationalTableModel esitettynä QTableView-widgetissä.

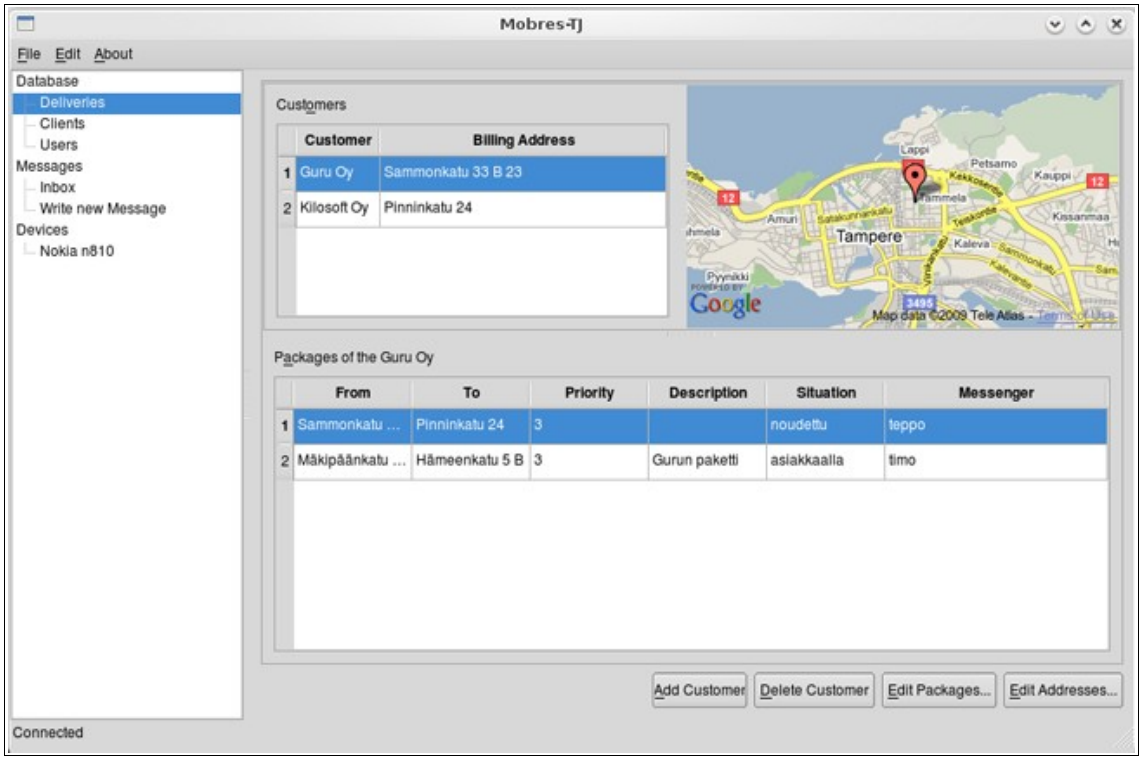
5.2.3 QtWebkitin käyttö karttaominaisuuden luomiseen

Mobres-TJ:ssä hallitaan jatkuvasti osoitteita, oli kyse sitten asiakkaan laskutusosoitteesta tai toimituksen lähde- tai kohdeosoitteesta. Tästä syystä halusin toteuttaa ohjelmaan karttaominaisuuden. Karttaa en suinkaan toteuttanut itse, vaan päätin luoda QtWebKitin avulla Google Maps -widgetin. Google Maps on Googlen tarjoama ilmainen selainpohjainen karttapalvelu, jonka avulla voi etsiä tiettyjä sijainteja tai kokonaisia reittiohjeita käyttäjän antamien syötteiden mukaan. Google Maps -widgetin luomisesta QtWebKitin avulla löytyi useampikin esimerkki, joista Qt Labsin blogista löytynyt esimerkkiluokka sopi hyvin tarkoitukseeni joidenkin muutosten jälkeen.

Google Maps tuotiin Mobres-TJ:hin perimällä jo mainittu QWebView omaksi Map-luokaksi. Luokassa on slot, joka vastaanottaa osoitteen QString-tyyppinä ja päivittää näin

karttanäkymän. Löytämäni esimerkki luki osoitteita tiedostosta, joten widgetin rajapintaa piti muuttaa hieman. DeliverySummary-luokaan lisättiin osoitin Map-luokan olioon ilmentymämuuttujana ja funktio updateGoogleMaps(), joka suorittaa tietokantahaun tauluun, joka sisältää osoitteita. Tämä haku suoritetaan aina, kun aiemmin tässä työssä käsitellyn pakettitaulukon riviä napsautetaan. Tietokannan palauttama osoite lähetetään parametrinä Map-luokan olion geoCode()-funktiolle, mikä käynnistää Map-luokan sisäisen prosessin karttanäkymän päivittämiseksi. Tämä prosessi perustuu ensin GPS-koordinaattien noutamiseen ja myöhemmin karttanäkymän päivittämiseen näiden koordinaattien perusteella.

Ennen yhteenvetoa vielä kuva (Kuvio 10) Mobres-TJ:stä kirjautumisen jälkeen. Kuvassa on nähtävissä navigointi, pakettitaulukko, muutama painike hallintaa varten sekä juuri esitelty karttaominaisuus.



Kuvio 10. Mobres-TJ: päänäkymä

6 Yhteenveto ja loppusanat

Opinnäytetyönäni tutkin Qt-ohjelmointitekniikoita Java-ohjelmoijan näkökulmasta tuottaen toimeksiantajalleni Kilosoftille Qt-itseopiskelumateriaalia ja koodiesimerkkejä erilaisista Qt-tekniikoista osana Mobres-TJ -nimistä sovellusta. Itselleni asetin tavoitteeksi Qt-ohjelmointitekniikoiden opettelemisen, jotta voisin tulevaisuudessa työskennellä Qt-projektien parissa.

Vaikka Mobres-TJ ei täysin valmiiksi hiottu sovellus olekaan, olen tyytyväinen suoritukseeni. Kilosoft on saanut arvokasta Qt-koodia ja -luentoja wiki-muodossa ja oma Qt-osaamiseni on mielestäni hyvällä tasolla. Mainittakoon, että tämän hetkinen valintani työpöytäsovelluksen luomiseen olisi Javan sijaan Qt, ja uskon pystyväni tuottamaan Qt:lla nopeammin käyttökelpoisia sovelluksia kuin Javalla, vaikka en sitä yhtä hyvin tunnekaan.

Mobres-TJ:n sijaan olisi voinut olla järkevämpää toteuttaa useampia pieniä Qt-esimerkkejä. Näin olisi ollut mahdollista kokeilla useampia Qt-tekniikoita ja tutustua luokkakirjastoon laajemmin ja mahdollisesti löytää siitä enemmän puutteita. Mobres-TJ oli mielestäni liian laaja työ toteuttaa yksin näin lyhyessä ajassa. Siitä huolimatta ohjelman perustoiminnallisuus on valmis, ja ohjelman tekeminen loppuun olisi kiinni ajasta, ei enää osaamisesta.

Aihevalintani on mielestäni onnistunut, sillä se on ajankohtainen ja valmistelee hieman lukijaansa tulevaan aikaan, jolloin Qt-sovelluksia on mahdollista kääntää matkapuhelimelle. Varsinkin Qt:n Webkit tarjoaa useita mahdollisuuksia: kun Symbianin Qt-tuki valmistuu, on vain ajan kysymys milloin tarjolla on useita erilaisia selaimia ja ohjelmia matkapuhelimelle web-sisällön näyttämiseen ja hallintaan. Ikävä kyllä tällä hetkellä näyttää siltä, että juuri Webkitin integroiminen osaksi Qt:n Symbian-versiota tuottaa eniten hankaluuksia, sillä muista moduuleista on saatavilla jo esimerkkisovelluksia.

Uskon tämän työn tarjoavan Java-taitoiselle lukijalleen yhdessä Mobres-TJ:n lähdekoodien kanssa hyvän perehdytyksen Qt-ohjelmointiin. Lisäksi uskon, että opinnäytetyöstäni välittyy alustariippumattomuuden tärkeys varsinkin alati murroksessa olevalla mobiilialalla. On mahdollista ja jopa todennäköistä, että tulevaisuudessa mobiili- ja

työpöytäsovellusten raja hämärtyy entisestään.

Tässä opinnäytetyössä tekemäni vertailu Javaan ei ole opettanut minulle vain Qt:ta: olen lisäksi kehittynyt myös Java- ja C++-ohjelmoijana. Oli mukava huomata työtä tehdessäni, kuinka paljon Qt-ohjelmointi muistuttaa Java-ohjelmointia. Esimerkiksi iterointitekniikat suorastaan antavat ymmärtää, että Java on ollut jonkinlainen malli Qt:n iterointitekniikoita kehitettäessä. Samoin Qt:n luokkakirjaston perintähierarkia on osittain samanlainen Javan kanssa: molemmissa kehitysympäristöissä on kantaluokka, joka tarjoaa perustoiminnallisuuden luokkakirjaston luokille.

Lisäksi Qt tarjoaa luokkakirjaston ohella ratkaisuja C++-ohjelmoinnin avuksi. Esimerkiksi lapsi-vanhempi -mekanismi ja signaalit ja slotit ratkaisevat joitakin ongelmakohtia ja tekevät ohjelmoinnista yksinkertaisempaa. Luokkakirjasto ja QObject-kantaluokan tarjoamat ominaisuudet mahdollistavat monimutkaistenkin sovellusten kehittämisen nopeasti, sillä koodia tarvitaan vähemmän.

Tällä hetkellä Mobres-TJ:tä ei aiota jatkokehittää, mutta Qt-luennoista laaditaan koulutuspaketti, joka sisältää diasarjan ja koodiesimerkkejä. Kun Symbianin Qt-tuki valmistuu ja saa lopullisen muotonsa, myös siihen liittyviä harjoituksia tullaan varmasti tekemään enemmänkin. Tällä hetkellä Qt:n Symbian-versioon on kuitenkin käännetty vain muutama tärkein moduuli, joten Qt-sovellusten kehitys Nokian matkapuhelimiin ei ole vielä täysin ajankohtaista.

Qt:n luokkakirjasto tarjoaa useita helposti hyödynnettäviä mahdollisuuksia oli ohjelmoijalla Java-taustaa tai ei. Lisäksi luokkakirjasto ja sen tekniikat mahdollistavat ohjelmien tuottamisen nopealla aikataululla suurimmille käyttöjärjestelmille. Tästä syystä Qt sopii mielestäni esimerkiksi ohjelmointiharrastustaan aloittaville henkilöille.

Lähteet

Blanchette, Jasmin & Summerfield, Mark 2008. C++ GUI Programming with Qt 4, Second Edition. Westford: Massachusetts.

Haggar, Peter 2001. Java bytecode: Undestanding bytecode makes you a better programmer. [online] [viitattu 6.3.2009] http://www.ibm.com/developerworks/ibm/library/it-haggar_bytecode/

Hietanen, Päivi 2004. C++ ja olio-ohjelmointi. Jyväskylä: Docendo Finland Oy

maemo.org – Intro: Software Platform 2009. [online] [viitattu 18.4.2009]. http://maemo.org/intro/platform/

Molkentin, Daniel 2007. The Book of Qt 4: Art of building Qt applications. San Francisco: No Starch Press.

Niemeyer, Patrick & Knudsen, Jonathan 2005. Learning Java. Sebastopol: O'Really Media.

Products – Qt – A cross-platform application and UI framework 2009. Nokia Corporation. [online] [viitattu 10.4.2009]. http://www.qtsoftware.com/products

Qt 4.4 Whitepaper 2008. Nokia Corporation. [online] [viitattu 24.4.2009]. http://www.qtsoftware.com/files/pdf/qt-4.4-whitepaper

Qt Reference Documentation 2008. Nokia Corporation. [online] [viitattu 24.4.2009]. http://doc.trolltech.com/4.4/classes.html

The Nokia acquisition – Qt – A cross-platform application and UI framework 2009. Nokia Corporation [online] [viitattu 10.4.2009]. http://www.qtsoftware.com/about/the-nokia-acquisition