QML QT-OHJELMOINNIN TUKENA

Lasse Pirinen

Opinnäytetyö
Huhtikuu 2011
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotannon suuntautumisvaihtoehto
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotannon suuntautumisvaihtoehto

PIRINEN, LASSE: QML Qt-ohjelmoinnin tukena
Opinnäytetyö 69 sivua
Huhtikuu 2011

Opinnäytetyön tavoitteena on esitellä QML-ohjelmointia ja sen taustalla toimivia Qt-ohjelmointitekniikoita sekä yleisesti että mobiililaite huomioonottaen. Työ voi toimia myös oppaana QML-ohjelmointiin Qt-osaajalle. Työssä käsitellään olennaisimpia Qt-ohjelmointiprosessin vaiheita ja tekniikoita QML-osaamisen tueksi sekä esitellään QML-ohjelmointikieltä yleisesti.

Kirjoitystyön ohella työstettiin QML-demosovellus, joka antaa toimeksiantajalle mahdollisuuden esitellä QML-teknologiaa demomessuilla. Se toimii myös teknologioiden esittelyn tukena opinnäytetyössä. Työn aikana tutkittiin teknologioiden historiaa ja ohjelmointiperiaatteita sekä yleisesti mitä ohjelmistosuunnittelijan tulee ottaa huomioon mobiililaitteelle ohjelmoitaessa.

Työ sisältää koodiesimerkkejä sekä Qt:sta että QML:stä ja kuvailee käytännön prosessia edellä mainittujen teknologioiden hyödyntämisessä. Työssä esitellään myös Qt- ja QML-teknologioiden välistä kommunikointia sekä valmiin sovelluksen luomiseksi tarvittavia toimia, kuten sovelluksen binääriksi kääntämistä sekä paketoimista.

Asiasanat: Mobiiliohjelmointi, Qt, QML

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Engineering

PIRINEN, LASSE: Qt software development supported by QML
Bachelor's thesis 69 pages
April 2011

The object of this thesis work is to introduce QML programming and the underlying Qt programming languages functionality both in general and from a mobile devices perspective. This work can also function as a guide to QML programming for a Qt developer.

The work contains a demonstration of the most relevant steps to take in a Qt programming process from the QML programming point of view and a presentation of the QML programming language in general. Along with this written work the author has created a QML program for the employer to use at demo events to demonstrate the QML technology. At the same time the program functions as a case study in demonstrating the QML programming language.

During the process of creating this work the author studied the history and programming principles of the technologies mentioned previously as well as what does a programmer have to take into account when designing and implementing software for a mobile device.

During the reading of this thesis you will find multiple coding examples from both Qt and QML and a description of the partical processes of creating a program with said technologies. The work also contains an explanation of communication between Qt and QML and the steps to be taken to create a complete program, such as compiling to binary and how to package a Qt program.

Keywords: Mobile programming, Qt, QML

SISÄLLYS

1 JOHDANTO

IT-ala kehittyy ja muuttuu jatkuvasti. Alalla työskentelevät ihmiset joutuvat alati opettelemaan uusia asioita ja kaikista vähiten ne, jotka työskentelevät niin sanottujen ”aallonharja-teknologioiden” parissa. Tällaisia ovat uudet ja nopeasti suosiota kasvattaneet teknologiat, joiden suosion laskun riski on aina enemmän tai vähemmän olemassa. 1990-luvulla alkunsa saanut Qt on tietyllä tavalla tällainen teknologia. Sen suosio on ollut vakaassa kasvussa syntyhetkistään asti, mutta sittemmin sen ovat ottaneet käyttöön suuret IT-alan yritykset kuten Intel ja Nokia. Tällaiset yritykset työllistävät paljon, mutta strategisien suunnanmuutosten tapahtuessa saatetaan myös menettää monia työpaikkoja.

Mobiililaite terminä ei enää käsitä vain kännyköitä ja taskutietokoneita, sillä Apple toi vastikään markkinoille oman mobiililaitteiden joukkoon liittyvän taulutietokoneen eli tabletin. Myös Nokia on viime vuosina panostanut mittavasti mobiiliteknologiaan ja tämän panoksen myötä Qt-teknologia on saavuttanut jalansijaa mobiililaitteiden keskuudessa. Tässä opinnäytetyössä käsittelen Qt:tä sekä siihen pohjautuvaa QML-teknologiaa sekä mobiililaitteelle ohjelmoimista.

Lokakuussa 2010 alkaneen harjoittelujakson aikana tutustuin Qt- ja QML-teknologioihin paremmin ja sovin tekeväni opinnäytetyön näistä teknologioista työnantajalleni. Työn tavoitteena on syventää taitojani edellämainittujen teknologioiden parissa ja esitellä niitä niin, että työ voi toimia myös oppaana näiden teknologioiden opettelussa. Tarkoituksena on saavuttaa edellämainitut tavoitteet kirjoittamalla tämä dokumentti sekä ohjelmoimalla mobiilisovellus, jota tilaaja voi käyttää IT-alan demomessuilla teknologian esittelyyn.

Tämä opinnäytetyö on suunnattu pääosin QML-kielestä kiinnostuneille ohjelmoijille, joilla on jo ennestään kokemusta vähintään pintapuolisesti C++-ohjelmoinnista sekä Qt-teknologiasta. Työssä käyn läpi tiettyjä Qt:n perusasioita, jotka pohjustavat myöhempää QML-teknologian käsittelyä. Työssä olevien teknisten asioiden ymmärtäminen vaatii aikaisempaa kokemusta ohjelmoinnista ja Qt-teknologiasta.

2 TERMIT

Tässä kappaleessa käydään lyhyesti läpi opinnäytetyön olennaisimpia termejä. Samoja termejä pyritään selittämään myöhemmin myös tekstin seassa, mutta kaikissa tapauksissa se ei ole edullista lukijalle, sillä lauseet voivat muodostua häiritsevän pitkiksi.

Työpöytälaite-termiä käytetään opinnäytetyössä mobiililaitteen vertailukohteena. Se koostuu painavista ja vaikeasti liikuteltavista osista, joka tarkoittaa sitä, että laite ei ole tarkoitettu mukana kannettavaksi. Esimerkiksi kotitietokone on työpöytälaite.

Mobiililaite tarkoittaa mukana kannettavaa laitetta. Mobiililaitteeksi voidaan ajatella kaikenlaiset kannettavat puhelimet sekä kannettavat tietokoneet, mutta tänä päivänä myös tabletit eli taulutietokoneet.

Käyttöjärjestelmä on tietokoneessa toimiva sovellus, joka sisältää muita sovelluksia. Käyttöjärjestelmä vastaa tietokoneen laiteresurssien hyödyntämisestä ja tarjoaa alustan, jonka sisällä monenlaiset sovellukset toimivat.

Lähdekoodi tarkoittaa ohjelmoijan kirjoittamaa kuvausta sovelluksen toiminnasta, jonkun tietyn ohjelmointikielen sääntöjen mukaisesti.

Kääntäjä on sovellus, joka muuntaa ohjelmoijan kirjoittaman lähdekoodin muotoon, jossa se voidaan käynnistää käyttöjärjestelmän sisällä. Usein tämä muoto on binäärikoodia.

Luokka on olio-ohjelmoinnin peruskäsite. Luokkarakenne on osa ajatusmaailmaa, jossa ohjelmoija suunnittelee loogisia kokonaisuuksia sovelluksen lähdekoodiin. Luokka edustaa yhtä tällaista kokonaisuutta ja siitä luodaan sovelluksen ajon aikana instansseja eli ilmentymiä, joita sanotaan olioksi.

Olio on luokasta luotu ilmentymä ja sen toiminnallisuus vastaa luokassa määriteltyä runkoa. Yhden luokan määrittelemän rungon mukaisia olioita voidaan luoda useita sovelluksen ajon aikana.

Perintä on olio-ohjelmoinnin käsite. Perintä tarkoittaa toiminnallisuutta, jossa kahden luokan välille luodaan isäntä-lapsi-suhde. Lapsiluokka perii isäntäluokkansa kaikki ominaisuudet ja lapsiluokasta luodut ilmentymät eli oliot voivat hyödyntää näitä ominaisuuksia sovelluksen ajon aikana. Lapsiluokka on usein erikoistunut versio isäntäluokastaan.

Funktio on osa sovelluksen toiminnallisuutta, jonka tarkoitus on suorittaa jokin tietty tehtävä. Ohjelmoija määrittelee useita funktioita sovelluksen lähdekoodiin ja nämä funktiot muodostavat lopulta kokonaisen sovelluksen toiminnallisuuden.

Ohjelmakirjasto on kokoelma funktioita ja luokkia, joita voidaan hyödyntää ohjelmistokehityksessä.

Makro on opinnäytetyössä käytetty terminä viittaamaan ohjelmointikieleen itseensä sisäänrakennettuja käännöksen aikaina tulkittavia toiminnallisuuksia. Makrojen hyödyntäminen lähdekoodissa on ohjelmoinnissa käytetty tapa ilmaista joukko sovellusohjeita lyhyesti. Toisinsanoen ohjelmoijan määrittelemä makro lähdekoodissa vastaa suurempaa toiminnallisuutta, jonka kyseinen ohjelmointikieli tarjoaa. Ohjelmoijan ei tarvitse makron avulla kirjoittaa koko toiminnallisuutta lähdekoodiin, vaan hän voi hyödyntää kielen tarjoamaa makroa.

SDK on lyhenne sanoista Software Developement Kit ja se tarkoittaa kokoelmaa työkaluista, joita ohjelmoija voi hyödyntää jonkin kielen lähdekoodin kirjoittamiseen.

Sulautettu järjestelmä tarkoittaa käyttöjärjestelmän kaltaista sovellusta ja sen yhteydessä toimivaa laitteistoa. Sulautetun järjestelmä on kokonaisuus jossa käyttöjärjestelmä on suunniteltu toimimaan jonkin tietyn laitteiston kanssa yhdessä.

Laitteistokiihdytys tarkoittaa toiminnallisuutta, jossa jokin toiminnallisuus tai sovellus on suunniteltu käyttämään suorituksessaan tiettyä laitetta yleiskäyttöisen prosessorin sijaan. Esimerkiksi peli, jonka graafinen ympäristö on suunniteltu hyödyntämään näytönohjainta emolevyssä olevan keskusprosessorin sijaan, hyödyntää laitteistokiihdytystä.

Widget (myös GUI Widget) on synonyymi käyttöliittymäkomponentille. Käyttöliittymäkomponentti on oma modulaarinen käyttöliittymän elementti, jolla on jokin tietty tarkoitus. Käyttöliittymäkomponentti voi olla esimerkiksi yksi nappi käyttöliittymässä tai vaihtoehtoisesti monimutkaisempi kokonaisuus kuten digitaalisen kellon näyttö.

RSS-syöte tarkoittaa internetistä löytyviä verkkojulkaisuja. RSS on lyhenne termeistä Really Simple Syndication. RSS-syöte on XML-muotoista dataa, joka luetaan RSS-lukijaksi kutsutulla ohjelmalla. Esimerkikki RSS-syötteestä voisi olla kokoelma uutisotsikkoja ja pieni kuvaus otsikon sisällöstä.

3 TAUSTAT

3.1 Alan tarpeet

QML on tämän työn kirjoitushetkellä hyvin uusi teknologia. Opinnäytetyön aloittamisen aikaan olennaisin tilaaja teknologian tutkimiselle ja käyttämiselle oli Nokia. Vaikka Nokialla on suuri menekki Afrikassa ja Aasiassa halvempien puhelimien saralla (Pingdom, Mobile web usage highest in Asia and Africa), on yritys panostanut myös kovasti älypuhelinkilpailuun. Nokian Symbian-alusta sai rinnalleen Linux-pohjaisen Maemon vuonna 2005 (Maemo, Evolution of Maemo), jonka jälkeen Intel ja Nokia aloittivat yhteistyön ja ilmoittivat helmikuussa 2010 järjestetyssä Mobile World Conferenssissa että Intelin kehittämä Linux-pohjainen Moblin käyttöjärjestelmä ja Nokian Maemo yhdistyvät yhdeksi uudeksi käyttöjärjestelmäksi, MeeGoksi (Nokia Conversations, Nokia and Intel create MeeGo for new era of mobile computing).

MeeGo-alusta, kuten Symbiankin, tarvitsevat onnistuneen rungon lisäksi myös kunnon ohjelmistotarjonnan. Tämän saavuttamiseksi Nokia pyrki hyödyntämään Qt- ja QML-teknologioita. Koska Qt:llä on mahdollista tuottaa sovelluksia monelle alustalle, ovat Nokia ja Intel hyvässä asemassa kannustaessaan ohjelmistokehittäjiä opettelemaan Qt:tä. Koska tavoitteena on tehdä ohjelmistokehityksestä mahdollisimman hauskaa, helppoa ja kivutonta, on Qt suunniteltu helpoksi omaksua ja näin houkuttelee yhä lisää kehittäjiä Qt:n pariin.

Tilanne muuttui Nokian osalta 11.2.2011, kun yritys ilmoitti siirtyvänsä käyttämään Microsoftin Windows Mobile 7 –käyttöjärjestelmää pääasiallisena mobiilipuhelimien käyttöjärjestelmänä (Nokia Corporation, Tiedotteet). Ilmoituksen myötä Symbian-käyttöjärjestelmän merkitys Nokialle hiipuu virallisesti tulevien vuosien kuluessa ja Intelin kanssa työstetty MeeGo-käyttöjärjestelmä toimii strategiselta osaltaan uusien mobiiliteknologioiden tutkimisen tukena. Nokian mukaan Qt- ja QML-teknologiat eivät tule olemaan kehitystyökaluja Nokian Windows-puhelimelle (Nokia Qt Blogs, Nokia new strategic direction. What is the future for Qt?). Tämä vähentää Qt- ja QML-osaamisen tarvetta Nokian suunnalta, mutta ei tee siitä täysin tarpeetonta. Nokian suunnitelmana on myydä vielä 150 miljoonaa Symbian-laitetta muutaman tulevan vuoden aikana (Nokia Qt Blogs, Nokia new strategic direction. What is the future for

Qt?) ja Nokia vakuuttaa jatkavansa Qt:n kehitystä yhdessä Digian kanssa, jolle Nokia ilmoitti myyneensä Qt:n kaupallisen puolen lisenssioikeudet marraskuussa 2011 (Nokia Qt Blogs, Qt and Digia, facts and fiction). Nokia vakuuttaa, että Symbian-alusta ei ole häviämässä pitkään aikaan ja alustalle kehittäminen Qt- ja QML-teknologioilla on vielä mahdollista ja järkevää (Nokia Conversations Youtube Channel, Nokia CTO Rich Green talks about Symbian).

3.2 Tilaajayrityksen tarpeet

Opinnäytetyön tilaajayritys työskentelee tiivissä yhteistyössä Nokian kanssa älypuhelinten saralla. Viime vuosina tapahtuneet uudistukset Qt:ssä sekä QML ja MeeGo-alusta tuovat tarpeen uudenlaisen ohjelmistokehityksen opetteluun. Vaikka Nokia on siirtymässä pääosaisena strategia-alustana Windows-pohjaisiin puhelimiin, on Qt:llä ja QML:llä vielä paikkansa myös Nokian strategiassa. Ei myöskään sovi unohtaa, että Intel toimii myös osaltaan MeeGo-käyttöjärjestelmän kehittymisen hyväksi. Tämän lisäksi Qt:n ympärillä oleva yhteisö on suuri. Nokian ja Intelin projektien lisäksi Qt:n avulla on tuotettu monia sovelluksia toisistaan hyvinkin poikkeaviin tarkoituksiin (Meego Experts, 10 Qt use cases we didn’t know about).

3.3 Omat motiivit

Oma innostukseni IT-alaan lähti aikoinaan html-kotisivujen tekemisellä. Vuosien varrella olen oppinut monia ohjelmointikieliä, joilla ei ole mitään tekemistä web-suunnittelun kanssa, mutta samalla opin paljon myös grafiikka- ja käyttöliittymäsuunnittelusta. On pitkä aika, kun olen viimeksi päässyt kunnolla työskentelmään käyttöliittymien parissa, mutta QML antaa tilaisuuden juuri siihen. Mielestäni se on helppoa, hauskaa ja, kun sen yhdistää taustalla toimiviin Qt-kirjastoihin, se on varsin tehokas tapa tuottaa ohjelmistoja sitä tukeville alustoille.

4 QT JA QML

4.1 Qt

Qt (lausutaan kuten englanninkielen sana cute /ˈkjuːt/, joka tarkoittaa söpöä) on C++:aan pohjautuva alustariippumaton kehitysympäristö. Alun perin Qt oli vain ohjelmistokirjasto graafisten käyttöliittymien toteutusta varten, mutta ajan kuluessa se on paisunut monia luokkia ja tuhansia funktioita sisältäväksi ohjelmistojen kehitysympäristöksi (Blanchette & Summerfield 2008, xix).

Qt:n taustalla toimii tavallinen C++-koodi ja Qt:llä kirjoitetun sovelluskoodin sekaan voi myös kirjoittaa tavallista C++:aa. Qt kuitenkin rikastaa C++-kieltä helppokäyttöisten funktioiden lisäksi myös tehokkailla makroilla ja mocilla eli Meta-Object Compileriksi kutsutulla koodigeneraattorilla. Meta-Object Compiler on Qt:n taustalla toimiva sovellus, joka huolehtii Qt:hen sisäänrakennetuista C++-lisäyksistä. Sen tehtävänä on muokata Qt:n sääntöjen mukaisesti kirjoitetusta sovelluskoodista natiivia C++-koodia, josta sitten tehdään oikea käynnistettävä sovellus. Qt-koodi siis muunnetaan aina aidoksi C++-koodiksi ja vasta sitten sovellukseksi. Tämä on Qt:n salaisuus sen helppokäyttöisyyteen ja monialustaisuuteen (Qt Reference Documentation, Using the Meta-Object Compiler).

Qt on moneen muuhun ohjelmointikieleen verrattuna erilainen. Kun Qt:n alkuperäiset arkkitehdit kohtasivat ongelman, he eivät vain miettineet, mikä olisi nopein tai yksinkertaisin ratkaisu. He miettivät, mikä olisi oikea ja parhain ratkaisu, jonka jälkeen he dokumentoivat tämän ratkaisun. Vaikkakin he tekivät virheitä ja osa ratkaisuista ei päätynyt lopulliseen kokonaisuuteen, he tekivät monia asioita oikein ja virheelliset ratkaisut korjattiin ja niiden ilmestyessä niitä korjataan yhä. (Blanchette & Summerfield 2008, xiii).

C++ GUI Programming with Qt 4 –kirjan alussa Matthias Ettrich on sitä mieltä, että kaiken Qt:n markkinointimateriaalin ohella tärkein syy, miksi Qt on menestyksekäs on se, että ohjelmoijat pitävät Qt:sta. Hänen sanojensa mukaan Qt:ssä on järkeä. Vaikkakin Qt:ssä on puutteita ja virheitä aika ajoin ja varsinkin sillon, jos ohjelmoija haluaa kokeilla uusimpia ja tuoreimpia ominaisuuksia, on syytä varautua ongelmiin. Qt:n

kehitysyhteisö on massiivinen sekä yritysmaailmassa että avoimen lähdekoodin kehitysyhteisön puolella.

Tämän työn kirjoitushetkellä Qt on ottamassa jälleen uusia askelia. Nokian siirtyessä Windows-alustaan puhelimissa ja jättäessään MeeGon kehityksen vähemmälle huomiolle (Nokia Corporation, Tiedotteet), on Qt ottanut askeleen kohti Googlen Android-mobiilialustaa. Avoimen lähdekoodin kehittäjä nimeltään BogDan Vatra loi ympäristön, jossa Qt-koodia on mahdollista kääntää myös Android-alustalle. Tämän lisäksi hän loi myös Ministro palvelun, joka pitää huolen siitä, että tarvittavat kirjastot on asennettu loppukäyttäjän laitteelle Qt-sovelluksen toiminnallisuuden takaamiseksi. Tämä kaikki tapahtui ilman Nokian tai Googlen tukea täysin avoimen lähdekoodin projektina (Nokia Qt Labs, Bringing Qt applications to Android – a quickstart video).

4.1.1 Qt:n lyhyt historia

Norjalaisessa teknologia-instituutissa tavanneet Haavard Nord ja Eirik Chambe-Eng saivat idean monialustaisen oliopohjaiseen ohjelmakirjastoon keskustellessaan vuonna 1990. He kehittivät Qt-kirjastoa yhdessä vuodesta 1991 alkaen harrastepohjalta ilman korvausta työssäkäyvien vaimojensa elättäminä. Vuonna 1994 miehet perustivat Trolltech-yrityksen, joka myöhemmin tuli tunnetuksi Qt:n alkuperäisenä kehittäjänä. Yrityksen nimi oli aluksi Quasar Technologies, sitten Troll Tech ja lopulta Trolltech. Trolltech toi Qt:n julkisuuteen ensimmäistä kertaa toukokuussa vuonna 1995. (Blanchette & Summerfield 2008, xix).

Qt nimessä Q-kirjain valittiin, koska Haavardin Emacs-tekstinkäsittelytyökalun fontissa kirjain näytti kauniilta. T-kirjain taas tuli sanasta toolkit, jota inspiroi samaan tapaan t-kirjainta käyttävä Xt-kirjasto, X Toolkit. Vuonna 1995 Haavard ja Chambe-Eng saivat ensimmäisen sopimuksensa, kun heidän yliopistonsa professorin välityksellä Metis-yritys tarjosi heille sopimusta kehittää ohjelmistoja Qt-kirjaston pohjalta. Näihin aikoihin Trolltech palkkasi Arnt Gulbrandsenin, joka Qt:n koodia kehittäessään loi kirjaston ympärille tänäkin päivänä arvostetun nerokkaan dokumentointijärjestelmän (Blanchette & Summerfield 2008, xix).

Lopulta vuoden 1995 kesällä Qt laitettiin julkisesti internetiin kahdella lisenssillä. Toinen oli maksullinen kaupalliseen kehitykseen tarkoitettu lisenssi ja toinen ilmainen avoimeen lähdekoodiin tarkoitettu lisenssi. Kymmeneen pitkään kuukauteen kukaan ei ostanut Qt:n kaupallista lisenssiä, mutta Metiksen kanssa tehty sopimus piti Qt:tä ja Trolltechia pinnalla. Viimein Euroopan avaruusjärjestö ESA osti vuonna 1996 kymmenen kaupallista lisenssiä ja Trolltech palkkasi tämän turvin uuden työntekijän (Blanchette & Summerfield 2008, xx).

Vuoden 1996 loppupuolella Qt:sta julkaistiin 1.0-versio ja siihen mennessä kaupallisia asiakkaita oli kahdeksan, joista jokainen oli eri maasta. Trolltech oli myynyt kaiken kaikkiaan 18 lisenssiä. Seuraavana vuonna Qt otettiin käyttöön KDE-ohjelmistokokonaisuuden kehittämiseen. KDE tunnetaan parhaiten useiden Linux-jakeluversioiden työpöytäympäristönä, mutta se on suunniteltu käytettäväksi myös muilla käyttöjärjestelmillä kuten Windowsilla ja MAC OS X:llä. KDE-päätöksen myötä Qt:sta tuli kehittäjien piireissä olennaisin ja tärkein kirjasto C++-käyttöliittymien kehittämiseen (Blanchette & Summerfield 2008, xx).

Ennen vuotta 2000 Qt:tä oli mahdollista käyttää Windows- ja Unix-pohjaisille järjestelmille kehittämiseen, mutta 2000-luvun alussa Qt:hen tehtiin tuki myös sulautettuille Linux-järjestelmille sekä Qtopia-kehitysalusta kännyköille ja kämmentietokoneille. Vuonna 1999 Qt oli voittanut LinuxWorld-palkinnon parhaasta kirjastosta ja työkalusta. Palkintojen määrä kasvoi 2000-luvun kuluessa. Muun muassa Qt:n sulautettujen järjestelmien tuki voitti ”Best Embedded Linux Solution”-palkinnon sekä vuosina 2001 että 2002 ja Qtopia-kirjasto saman palkinnon vuonna 2004 (Blanchette & Summerfield 2008, xx).

Vuonna 2001 julkaistiin Qt:n 3.0-versio ja se sisälsi tuen Windows-, Mac OS X-, Unix- ja Linux-käyttöjärjestelmille. Neljä vuotta myöhemmin eli vuonna 2005 Qt saavutti 4.0-version. Se sisälsi yli 500 luokkaa, 9000 funktiota ja kehittäjille tarjottiin mahdollisuus linkittää sovelluksensa vain niihin osiin Qt:tä, mitä he sovelluksessaan tarvitsivat. Tässä vaiheessa Qt oli kasvanut niin suureksi, että sitä ei voinut enää kutsua vain graafisten käyttöliittymien kehitystyökaluksi, vaan siitä oli tullut täysiverinen ohjelmistokehityskirjasto (Blanchette & Summerfield 2008, xx).

Kesäkuussa 2008 Nokia osti Trolltechin osana ohjelmistokehitysstrategiaansa mobiili- ja työpöytäalustoille ja syykuussa Nokia integroi Trolltechin täysin yritykseensä antaen uudelle yrityksen sisällä toimivalle osastolle nimeksi Qt Software ja myöhemmin Qt Development Frameworks (Nokia Corporation, The Nokia acquisition). Nokian myötä Qt on tullut tunnetuksi ja suosituksi myös mobiilikehittäjien keskuudessa.

Trolltechin 1990-luvulla aloittamasta C++-käyttöliittymäkirjastosta on kasvanut suosittu kehityskirjasto. Se on käynyt matkan muutaman eliittikäyttäjän joukosta työkaluksi, jota käyttävät päivittäin tuhannet maksavat asiakkaat ja kymmenettuhannet avoimen lähdekoodin kehittäjät maailmanlaajuisesti (Blanchette & Summerfield 2008, xxi).

4.1.2 Missä Qt:ta voi nähdä?

Qt:n alkuvaiheista on päästy todella pitkälle ja nykypäivänä Qt:tä todellakin käyttävät tuhannet asiakkaat sekä asiakkaisiin verrattuna moninkertainen määrä avoimen lähdekoodin kehittäjiä. Tätä työtä kirjoitettaessa, Nokian ja Intelin sekä heidän alihankkijoidensa tuotosten lisäksi loppukäyttäjälle näkyvimpiä Qt:llä tuotettuja ohjelmia ovat esimerkiksi Google Earth, Skype, VLC Media Player sekä Linux maailmassa suosittu graafinen työpöytäympäristö KDE (Nokia Corporation, Qt in use).

Nokia on omistanut Qt:n kaupallisen lisenssin myyntioikeuden siitä lähtien, kun yritys osti Trolltechin vuonna 2008. Kuten aiemmin mainittiin, marraskuussa 2011 Nokia ilmoitti myyvänsä nämä lisenssit suomalaiselle Digialle. Kaupan myötä Digialle avautuu uusia ovia muun muassa Yhdysvaltoihin sekä Norjaan ja Digia on luvannut jatkaa Qt:n kehitystä. (Nokia Qt Blogs, Qt and Digia, facts and fiction). Tähän päivään mennessä Qt:n kaupallisella puolella on tai on ollut monia merkittäviä asiakkaita kuten AMD, Asus, Canon, Euroopan avaruusjärjestö, Saab, Samsung, Siemens sekä Volkswagen (QtStudios Youtube Channel, Qt Everywhere on All Platforms). Tästä voidaan päätellä, että Qt on merkittävä tekijä ohjelmistotuotannon alalla.

4.2 QML

QML on Qt:n päälle rakennettu käyttöliittymien määrittelyyn suunniteltu kieli, jonka tarkoituksena on helpottaa ohjelmistokehittäjiä ohjelmien visuaalisen puolen toteutuksessa. Kuten aiemmin mainittiin, Qt-kirjasto suunniteltiin alusta alkaen helpoksi sekä hauskaksi omaksua ja sen toteutusratkaisut pyrittiin samaan mahdollisimman järkeviksi. QML vie tämän askeleen pidemmälle niin sanottuna deklaratiivisena kielenä, joka pohjautuu taustalla pyörivään Qt-koodiin. (Qt Reference Documentation, Qt Quick).

QML ja siihen liittyvä teknologiakokonaisuus tunnettiin aiemmin myös nimillä Qt Declarative, Declarative UI sekä myöhemmin käyttöön vakiintuneella Qt Quick –nimellä (Nokia Qt Blogs, Meet Qt Quick at Mobile World Congress). Qt Quickin avulla helposti luotavia käyttöliittymiä näkee usein kännyköissä ja muissa mobiililaitteissa, ja ne ovat omiaan houkuttelemaan kuluttajia. QML sisältää helpon syntaksinsa lisäksi ominaisuuksiltaan rikkaita käyttöliittymäelementtejä, joiden on tarkoitus auttaa kehittäjiä rakentamaan nopeasti visuaalisia kokonaisuuksia (Qt Reference Documentation, Qt Quick).

QML:än ollessa niin sanottu deklaratiivinen kieli, sen syntaksi muotoillaan niin, että koodi sisältää enemmänkin julistuksia siitä, *millaisia* asiat ovat eikä niinkään, *miten* ne toimivat. Imperatiivisissa (proseduraalisissa) kielissä muotoillaan päinvastoin (Fahland & Lübke & Mendling & Reijers & Weber & Weidlich & Zugal 2009). Tästä johtuen QML voi tuntua oudolta tai vaikeasti omaksuttavalta C++-kielen tai muun vastaavan proseduraalisten kielien ohjelmoijan näkökulmasta, mutta taas helposti lähestyttävältä deklaratiivisiin kieliin tottuneista, kuten esimerkiksi web-sivustojen tekijöistä, jotka tekevät töitä CSS- ja Javascript-teknologioiden parissa.

Kirjoitushetkellä QML:stä löytyy vielä muutamia puutteita, kuten alustakohtaiset widgetit. Tätä puutetta korjaamaan on ryhtynyt Qt Components –projekti, joka pyrkii tuomaan ohjelmoijille käytettäväksi alustariippumattomia komponenttiratkaisuja, kuten esimerkiksi pudotusvalikon tai latausruudun (Nokia Qt Labs, QML Components for Desktop?). Tämän lisäksi yksi merkittävimmistä puutteista on laitekiihdytystuen puuttuminen QML-graafiikkasuorituksessa. Tätä ongelmaa puolestaan pyrkii korjaamaan QSceneGraph-projekti, joka pyrkii tuomaan OpenGL-teknologian

paremmin QML:ään mukaan ja hyödyntämään laitekiihdytystä grafiikoiden esittämisessä (Nokia Qt Labs, Qt Scene Graph – Round 2). QML on kehittynyt erittäin nopeasti ja sen kasvua käyttöliittymien suunnittelun ja toteutuksen helpottamiseen on ilo seurata.

4.2.1 QML:n lyhyt historia

QML on tämän työn kirjoitushetkellä niin uusi teknologia, että oikeata alan kirjallisuutta ei siitä vielä ole. Tällä hetkellä parhain lähde tutkia QML:n kehityksessä tapahtuneita vaiheita on, Qt-yhteisön mukaan, Qt Labs –kehitysyhteisön sivustoilta ja Nokian Qt-sivuistoilta löytyvät julkiset blogikirjoitukset.

Adam Walhout, Qt:n markkinointipäällikkö, julkaisi helmikuussa 2010 Nokian Qt-blogisivustolla ilmoituksen Qt Quick –teknologiasta otsikolla ”Meet Qt Quick at Mobile World Congress”. Kirjoituksessa Walhout selventää, mitä Qt Quick on ja esittelee lyhyesti, mitä sillä voi tehdä sekä ilmoittaa Qt-yhteisölle Qt Quickin tulevan, Qt:n version 4.7 julkaisun mukana, kehittäjien saataville. Qt Quick oli tätä ennen Qt Labs –projekti ja sen kehitys tapahtui julkisesti beta-versiona. Qt Labsin Qt Quick –projektin tarkoituksena oli luoda helppokäyttöinen, deklaratiivinen, skriptauskielten kaltainen ohjelmointikieli käyttöliittymien kehittämiseen. Qt Quickin oli ja on tarkoitus kuroa umpeen kuilua käyttöliittymäkehittäjien ja ohjelmoijien välillä tarjoamalla samat helpot työkalut molemmille yhteiseen sovelluskehitykseen (Nokia Qt Blogs, Meet Qt Quick at Mobile World Congress).

Daniel Kihlberg Qt:n myynti-, markkinointi- ja palveluvastaava julkaisi syyskuussa 2010 blogikirjoituksen otsikolla ”Qt 4.7 is here, so is Qt Developer Days”. Blogikirjoituksessaan hän kirjoittaa Qt 4.7 julkaisusta ja sen myötä myös Qt Quick –julkaisusta. Hänen mukaansa Qt 4.7 tuo tullessaan ensimmäisen kerran teknologiaa, joka mahdollistaa Javascript-koodin hyödyntämisen Qt:ssä. (Nokia Qt Blogs, Qt 4.7 is here, so is Qt Developer Days). Qt 4.7 version julkaisun jälkeen QML-teknologia tuli laajemmin yleiseen käyttöön ja Qt-yhteisössä keskusteltiin paljon teknologiasta, korjattiin virheitä ja ideoitiin uusia ominaisuuksia.

Virallinen Qt Quick julkaistiin vasta Qt Nokia SDK 1.1 beta –kehitysympäristön myötä. Juha Latvala, Qt-kehityksen johtaja, julkaisi marraskuussa 2011 blogikirjoituksen otsikolla ”Qt SDK 1.1 beta has been released”. Blogikirjoituksessaan Latvala kertoo, että vaikka itse SDK on vielä beta-vaiheessa, sen sisältämät versiot Qt:sta sekä Qt Creator –työkalusta on julkaistu final-nimikkeen alla, eivätkä beta- tai preview-nimikkeen. Tämä tarkoittaa sitä, että tämän SDK:n myötä saavutettiin ensimmäinen virallinen Qt Quick ja QML-teknologian julkaisu. (Nokia Qt Blog, Qt SDK 1.1 beta has been released).

4.2.2 Qt QML:n taustalla

Qt tarjoaa kokoelman C++ rajapintoja, joilla yhdistää QML:n tarjoama helppokäyttöinen käyttöliittymäsuunnittelu kattavaan Qt-kirjastoon. Vaikka QML-ohjelmointi ei itsessään vaadi Qt-taitoja ohjelmoijalta ovat Qt-taidot erittäin olennaisia myös QML-ohjelmoinnissa. Valmis käynnistettäväksi binääriksi asti toteutettu QML-sovellus käyttää QML:ää käyttöliittymän luomiseen ja hyödyntää Qt:n C++-koodia kaikkeen syvemmällä olevaan käyttöliittymää tukevaan taustalogiikkaan (Qt Reference Documentation, QML for Qt Programmers).

Qt-ohjelmien, jotka käyttävät QML:ää käyttöliittymän esittämiseen, on hyödynnettävä Qt:n luokkaa QDeclarativeView tai QDeclarativeEngine taustatoiminnallisuuden luomiseen. Nämä luokat tarjoavat kaksi erilaista tapaa tuoda QML-koodi sovellukseen näkyville (Qt Reference Documentation, Qt Declarative UI Runtime). QDeclarativeView toimii widgetin tavoin tarjoten käyttöliittymäkomponentin, jonka sisällä voidaan näyttää kokonaisia Qt Quick –teknologialla luotuja käyttöliittymiä. QDeclarativeView-luokka niin sanotusti ”enkapsuloi” eli pitää sisällään kokonaan QML-kielellä luodun toteutuksen ja vie sen graafisista elementeistä huolehtivan taustaprosessin, QGraphicsViewin, näytettäväksi. (Qt Reference Documentation, QDeclarativeView Class Reference). QDeclarativeEngine taas toimii suoremmin Qt-kielen kanssa ja sen hyödyt tulevat paremmin esille silloin, kun halutaan integroida QML-käyttöliittymiä jo olemassa olevaan Qt-pohjaiseen käyttöliittymään (Qt Reference Documentation, Using QML in C++ Applications).

QML on suunniteltu osaksi helposti laajennettavaa sovelluskokonaisuutta Qt:n C++-koodia apuna käyttäen ja Qt:n Declarative-moduuli mahdollistaa QML-koodin käytön ja muokkaamisen taustalle rakennetusta C++:sta. Lisäksi Qt:n Meta-Object-toiminnallisuuden avulla QML- ja Qt-oliot voivat helposti keskustella keskenään ja näin esimerkiksi vanhempien pelkkään Qt-koodiin perustuvien ohjelmien integrointi uuteen QML-koodiin on pyritty tekemään helpoksi (Qt Reference Documentation, Using QML in C++ Applications).

5 TOTEUTUSTAVAT

Qt on hyvin joustava kohdealustoihinsa nähden. Kuten aiemmin mainittiin, Qt:llä voi kehittää sovelluksia monelle erilaiselle alustalle. Qt:n avoimen lähdekoodin projekteista projekti nimeltä Lighthouse on erityisen mielenkiintoinen kohdealustojen näkökulmasta. Lighthouse-projektin tarkoituksena on mahdollistaa Qt-kehitys sellaisille alustoille, joita Qt ei vielä tue, kuten esimerkiksi Android- ja iPhone-ympäristöille (Nokia Qt Labs, Lighthouse is integrated!). Qt:n ollessa sopiva kehityskieli monelle alustalle eli niin sanottu cross-platform-kieli, on hyvä käydä läpi pintapuolisesti, mille alustoille Qt:llä voi ohjelmistoja kehittää.

5.1 Mahdolliset kohdealustat

Qt:n tukemat alustat jaetaan Nokian toimesta kolmeen tasoon, joista ensimmäinen on tärkein. Kolmen tason luokittelu on tarkoitettu testauksen jäsentelyyn niin, että ensimmäisen tason alustat testataan kaikista ankarimmin, toisen tason hieman löysemmin ja kolmannen tason alustoja ei ollenkaan. Millä tasolla mikäkin alusta on, riippuu siitä, kuinka paljon Qt-kehittäjät kehittävät ohjelmistoja eri alustoille, eli taso määräytyy niin sanotusti suosion mukaan (Qt Reference Documentation, Supported Platforms).

Hyvin karkeasti jaettuna Qt:n tukemia alustoja ovat Windows, Mac OS X, Embedded Linux, Win CE/Mobile, Maemo/Meego, Symbian sekä Linux/X11. Näistä ensimmäisen tason alustoja ovat Linux/X11, Windows, Mac OS X, Embedded Linux sekä Symbian, joka tarkoittaa sitä, että niiden testaus ja tuki ovat parasta laatua.

Toisella tasolla on myös muutamia alustoja samalla nimellä, joten kehitysympäristöä valitessa tulee olla tarkkana. Syy, miksi samannimisiä alustoja on asetettu useammalle tasolle, johtuu eri alustoille olevista useista kääntäjistä. Esimerkiksi Linux-alustalle käännettäessä gcc-kääntäjän 4.2-versiolla on testaus ja tuki asetettu ensimmäiselle tasolle, kun taas samalle alustalle käännettäessä Itel Compiler –kääntäjällä, tuki on toisella tasolla (Qt Reference Documentation, Supported Platforms).

Virallisesti kaikkia toisen tason alustoja ei tueta, eikä mitään kolmannen tason alustaa. Kolmannelle tasolle sijoittuvat lähinnä itsenäiset avoimen lähdekoodin projektit, joiden toimivuutta isot Qt:ssä mukana olevat yritykset, kuten Nokia ja Intel, eivät takaa tämän opinnäytetyön kirjoitushetkellä millään tavalla (Qt Reference Documentation, Supported Platforms). Todennäköisesti Qt:n tukemat alustat tulevat ajan kanssa vaihtumaan, niiden määrä kasvamaan ja tuen laatu kehittymään. Kehitysympäristöä pystyttäessä onkin syytä tarkastaa, mille mahdollisille alustoille haluaa kohdentaa sovelluksensa ja valita tämän huomioon ottaen, jos mahdollista, parhaiten sopiva käyttöjärjestelmä ja kääntäjä kehitystä varten. Esimerksi gcc-kääntäjää tuetaan parhaiten Linux-ympäristössä, kun taas kirjoitushetkellä Symbian-alustalle käännettäessä helpoin asennusprosessi on Windowsille tarkoitettu SDK, joka pohjaa toimintansa Windowsin MinGW C++ kääntään.

5.2 Ojelmointiympäristö

Tämän opinnäytetyön tueksi tuotetun ohjelmiston tekemisessä keskitytään Windows-Linux- ja Symbian-alustoihin. Linuxiin siksi, että alustan osaaminen tuo arvokasta työelämän kompetenssia tekijälle ja Symbianiin siksi, että Symbian on alustana Nokialle vieläkin hyvin tärkeä ja testauslaitteena Symbian alustalla toimiva Nokian N8 –puhelin on tekijälle helposti saatavilla. Symbian puhelimelle kehitettäessä ehdottomasti vaivattomin kehitysalusta on edelleen Windows ja siksi opinnäytetyön aikana mainitaan nämä kolmea alustaa.

Linux-työpöytäympäristön ja Symbian-mobiililaiteympäristön pystyttäminen ovat keskenään suhteellisen samanlaisia prosesseja. Erilaisuutta kehittämiseen tuo itse käyttöjärjestelmä, jonka päällä kehitystyötä tulee tehdä. Tätä kirjoittaessa Linux-ympäristössä sovelluksia kehitettäessä, helpointa on asentaa Nokia Qt SDK, josta uusin versio on tällä hetkellä 1.1 beta. Vastaavan SDK:n saa Nokian sivuilta myös räätälöitynä Windowsille ja kyseinen versio sisältää kehitystyökalut muun muassa Symbianille. SDK:n 1.1 beta-paketti ei ole vielä täysin hiottu versio kehitystyökalusta, mutta se on tarpeellinen QML kehitystä varten, sillä aiempi vakaa Nokia Qt SDK 1.0 paketti ei sisällä kunnollista tukea Qt:n 4.7 versiolle, jonka myötä QML-kehitys on mahdollista. Nokia Qt SDK 1.1 beta –paketin sisältä löytyy myös Qt Mobility 1.1.1 –

kirjasto, joka on tarkoitettu mobiililaitteille suunnattuun kehittämiseen (Nokia Qt Blog, Qt SDK 1.1 beta has been released).

Linux-ympäristössä sovelluksia kehittäessä ongelmia voi syntyä, jos käyttöjärjestelmän käyttö yleisesti ei ole tuttua. Tämän opinnäytetyön ohessa kehitettävää sovellusta varten käytössä on Ubuntu Linux –jakeluversio sillä Ubuntu on kirjoittamishetkellä ainoa Nokia Qt SDK:n tukema Linux-ympäristö. Ubuntulle SDK:n asentaminen onnistuu helposti esimerkiksi lataamalla Nokia Qt SDK:ta varten tarkoitettu Nokian Qt sivuilla tarjottu .run-päätteinen tiedosto, antamalla sille käynnistysoikeudet ja käynnistämällä se tavallisena käyttäjänä komentoriviltä. Windowsille Nokia Qt SDK:n asentaminen sujuu yhtä helposti kuin minkä tahansa muunkin sovelluksen. Kehittäjän on hyvä ottaa huomioon, että Windows-paketti on 1,6 gigatavun kokoinen ja Linux-versiokin kooltaan vai hieman alle puolet siitä. Internet-liittymän nopeudesta riippuen paketin lataaminen voi kestää hyvinkin kauan ja asennusprosessi tehokkaallakin tietokoneella useita kymmenia minuutteja (Nokia Qt Blog, Qt SDK 1.1 beta has been released).

5.3 Mobiililaiteen huomiointi

Nykyisin mobiililaite löytyy teollisesti kehittyneistä maista lähes jokaiselta. On myös hyvin mahdollista, että kotitaloudessa on useampi kuin yksi mobiililaite asukasta kohden. CIA:n vuodesta 1996 asti tehdyn tutkimuksen mukaan vuonna 2008 kartoitetuista 212 maasta yli neljäsosalla oli yksi tai enemmän mobiililaitetta per henkilö (Nation Master, 2008). Mobiililaitteella voi tehdä nykyään niin monta erilaista toimintoa, että sen hyödyllisyyttä on vaikea olla huomioimatta. Tommi Mikkosen (2004, 1) mukaan suuri osa laitteiden houtuksesta tuleekin niiden sisältämistä ja niille saatavista ohjelmistoista, jotka tuovat olennaisen lisäarvon laitteen perustoiminnoiden rinnalle.

Raskaamman kotitietokoneen suorituskyky on lähes poikkeuksetta parempi kuin mobiililaitteen ja näin ollen työpöydälle suunniteltu ohjelmisto saattaa toimia erittäin hitaasti, jos ollenkaan, mobiililaitteella. Mobiililaitteessa saatavilla olevan muistin määrä on rajoitettu ja sen käyttö tulee suunnitella tarkoin. Jokainen mobiililaite on erilainen resurssien ja saatavilla olevien käyttöominaisuuksien suhteen. Ohjelmistosuunnitelijan tulee ottaa huomioon kaikki nämä eroavaisuudet kaikissa

kohdelaitteissa (Qt Reference Documentation, Optimizing Applications for Mobile Devices).

Mobiililaite on jo nimensäkin puolesta kannettava ja johdoton laite, joka ei ole jatkuvasti kiinni virtalähteessä. Akun kesto muodostuu näin olennaiseksi osaksi mobiililaitteen käyttöä. Ohjelmistosuunnittelijan tulee pitää ohjelmistonsa kokonaisvirrankulutus mahdollisimman alhaalla optimoidakseen sovellus niin, että akku kestää mahdollisimman kauan (Qt Reference Documentation, Optimizing Applications for Mobile Devices).

Mobiililaitteen ohjelmistoissa siedetään huomattavasti vähemmän virheitä ja tahmaisuutta kuin työpöytäkoneissa (Tommi Mikkonen, 2004, 6). Tämä ja yllämainitut suorityskykyyn ja ominaisuuksiin liittyvät seikat luovat ohjelmistosuunnittelijalle tarpeen tutkia ja vertailla erilaisten toteutustapojen energiankulutusta ja suorituskyvyn rajoitteita sekä syyn pyrkiä optimoimaan ohjelmiston suoritus mobiililaitteessa, kun taas työpöytäsovelluksessa nämä asiat eivät ole niin olennaista. Suunniteltaessa ohjelmistoa molempien kaltaisille laitteille tulevat mobiililaitteen vaatimukset karsimaan suuremman suorituskyvyn mukana tulevia mahdollisuuksia. Hyödynnettäessä Qt:n monialustaisuuden mukana tulevaa mahdollisuutta kääntää ja asentaa sama ohjelmisto sekä työpöytä- että mobiililaitteelle, lisääntyy ohjelmiston toteuttajan vastuu sovelluksen suunnittelusta ja testauksesta. Tällaisissa tapauksissa on järkevää pohtia tapauskohtaisesti, tulisiko ohjelmistosta tehdä työpöytä- ja mobiiliversiot erikseen vai onko mahdollista hyödyntää samaa toteutusta molemmille.

Kohdeyleisön huomiointi on hyvä tapa suunnitella ohjelmistoa. Miettimällä, ketkä ovat ohjelmiston pääosaisia käyttäjiä ja mihin nämä käyttäjät tarvitsevat sitä, auttaa muodostamaan kokonaiskuvan sovelluksesta (Qt Reference Documentation, Optimizing Applications for Mobile Devices). Kartoitusta voi tehdä esimerkiksi avoimella kyselyllä tai tutkimalla mahdollisesti kilpailevia jo valmiita samaan käyttötarkoitukseen tuotettuja ohjelmistoja.

Tavallisesti mobiililaitteissa on pienempi näyttö kuin työpöytälaitteissa ja se tuo omat ongelmansa ohjelmistosuunnitteluun. On hyvä suunnitella mitkä asiat ovat ohjelmiston käyttäjän kannalta olennaisimpia ja tuoda ne esiin selkeästi käyttöliittymässä. Ei välttämättä ole kovin järkevää yrittää sovittaa mahdollisimman paljon sisältöä kerralla

näkyville, vaan osittaa sitä valikoiden muun interaktiivisten toimintojen taakse (Qt Reference Documentation, Optimizing Applications for Mobile Devices).

Verrattessa mobiililaitteita työpöytälaitteisiin näytön koko on pienempi, mutta myös mobiililaitteiden keskinäisessä vertailussa näytön koko saattaa vaihdella huomattavasti. Tämän takia ohjelmisto tulisikin suunnitella skaalautuvaksi näytön koosta riippuen ja varmistaa, että sama määrä informaatiota ja toimintoja ovat saatavilla kaikilla näytön koilla (Qt Reference Documentation, Optimizing Applications for Mobile Devices).

Mobiililaitteen käytössä yksi täysin mobiiliselle laitteelle olennainen ominaisuus on näytön orientaation muuttuminen. Näissä laitteissa sovelluksen on pystyttävä näkymään toimivasti sekä vaaka- että pystytasossa ja ottaa huomioon näiden näyttötilojen mahdollinen vaihtuminen kesken ohjelmiston käytön laitetta kääntäessä. Ohjelmiston voi esimerkiksi suunnitella joko skaalautumaan näytön orientaation mukaisesti tai pakottaa se näkymään aina tietynlaisena laitteen kääntämisestä huolimatta (Qt Reference Documentation, Optimizing Applications for Mobile Devices).

Ohjelmistossa navigointi on huomattavan erilaista työpöytälaitteeseen nähden. Lähes poikkeuksetta mobiililaitteissa ei ole käytössä hiirtä ja suurimmassa osassa ei myöskään ole näppäimistöä. Uusimmista älypuhelimista ja tableteista löytyy usein kosketusnäyttö, mutta vanhemmat puhelimet tarjoavat ohjelmiston kanssa vuorovaikutukseen vain numeronäppäimistön. On myös hyvä ottaa huomioon, että käyttäjä saattaa käyttää mobiililaitetta yhdellä kädellä ja suunnitella ohjelmiston toiminnallisuus sen mukaisesti (Qt Reference Documentation, Optimizing Applications for Mobile Devices).

Nykypäivänä mobiililaitteella on myös tavallista hyödyntää internet-yhteyttä. Jos käyttäjälle ei ole kiinteää tiedonsiirtosopimusta tai mahdollisuutta käyttää WLANia, tulee langattoman yhteyden käyttö nopeasti hyvin kalliiksi. Myös kuuluvuus on otettava huomioon suunniteltaessa ohjelmistoa, joka höydyntää internet-yhteyttä. Liikuttaessa mobiililaitteen kanssa paikasta toiseen, voivat yhteyksien katvealueet ja tarjolla olevien yhteyksien nopeudet muuttua jatkuvasti ja näin aiheuttaa ongelmia käyttäjälle (Qt Reference Documentation, Optimizing Applications for Mobile Devices).

6 OHJELMOINTIPROSESSI

6.1 Työstettävä sovellus

Opinnäytetyön yksi tarkoituksista oli tuottaa sovellus QML-teknologialla, jota tilaaja voi hyödyntää IT-alan demomessuilla teknologian esittelemiseen. Lyhyen mietinnän jälkeen sovelluksen aiheeksi muodostui RSS-lukija mobiililaitteille. RSS-lukijasovelluksia on olemassa jo useita ja useille eri alustoille, mutta työhön ei haluttu valita mitään monimutkaista tai innovatiivista sovellusaihetta, jotta se mahdollistaisi ja helpottaisi QML-perusteiden tutkimista ja esittelyä. Tämän lisäksi kirjoittajalla oli pitkään ollut tarvetta tietynlaiselle RSS-lukijalle ja opinnäytetyö tarjosi loistavan tilaisuuden rakentaa myöhemmin laajennettava pohja tällaiselle sovellukselle. Sovelluksen toiminnan tarkoituksena on mahdollistaa internetissä olevien RSS-syötteiden lukeminen mobiililaitteesta käsin. Sovelluksen käyttöliittymä tehtiin QML-teknologiaa hyödyntäen ja taustalle rakennettiin Qt-koodia, joka mahdollistaa sovelluksen binääriksi kääntämisen.

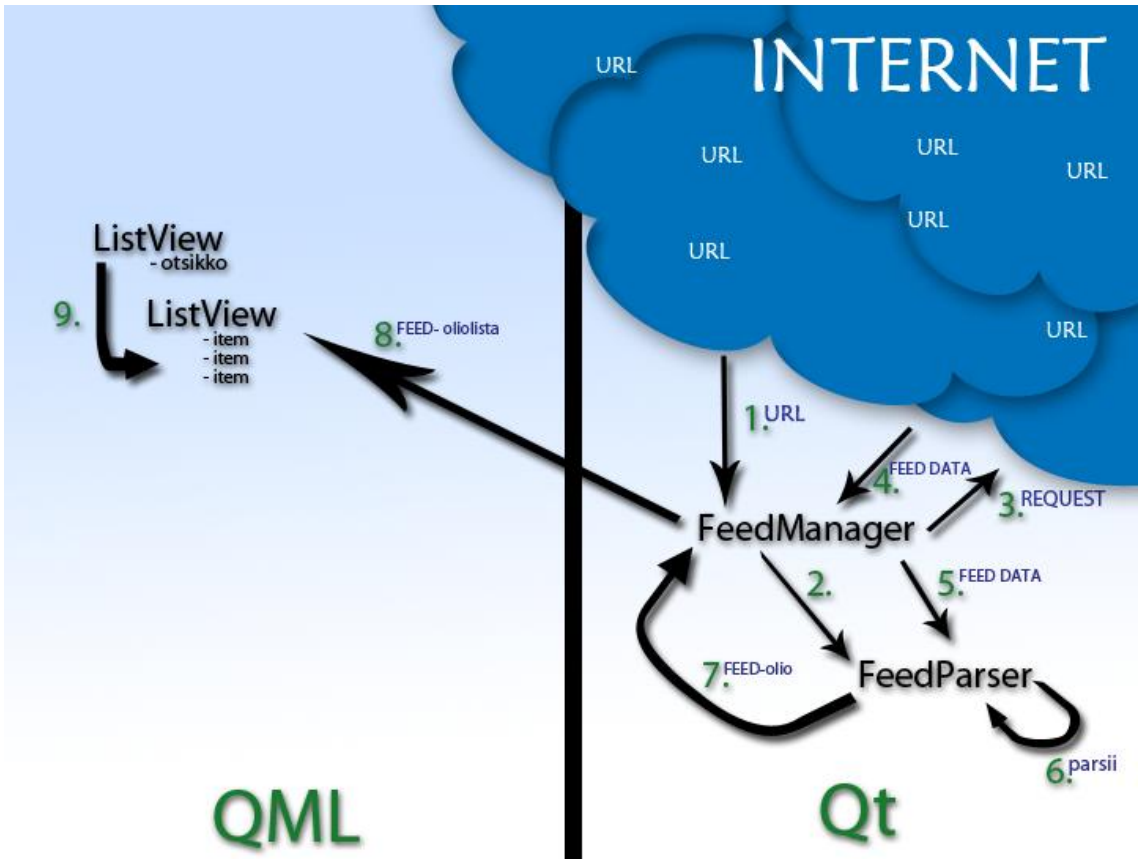
6.1.1 Sovelluksen teko

Sovellus työstettiin yhdessä opinnäytetyöprosessin kirjoitustyön kanssa. Tarkoituksena oli suunnitella mahdollisimman modulaarinen sovelluskokonaisuus, jota on helppo laajentaa myöhemmin. Työstäminen alkoi ohjelmistoalalta tutulla SCRUM-tyylisellä käyttäjätarina mallilla (user story), jossa kirjoitettiin mahdollisia tapoja käyttää sovellusta ja lähdettiin tuottamaan toiminnallisuutta sen pohjalta (Scrum Alliance, New To User Stories). Kyseisessä sovelluksessa esimerkki käyttäjätarinasta voisi olla esimerkiksi muodossa ”Käyttäjänä haluan lukea useita RSS-syötteitä saman sovelluksen avulla” tai ”Käyttäjänä haluan lisätä uuden RSS-syötteen sovellukseen”. Tämä tuo käyttäjäläheisen asetelman ohjelmistojen suunnitteluun sekä toteutukseen ja auttaa varmistamaan, että käyttäjien kaipaamia ominaisuuksia ei jää puuttumaan.

Käyttäjätarinoiden avulla saatiin sovelluksen rakenteen alustava suunnitelma kokoon, jonka jälkeen tarinat jaettiin sopiviin pienempiin työstettäviin osiin (subtask) ja aloitettiin itse toteutus. Tällaisessa työntekomuodossa on tavallista, että työnteon aikana

pienempiä alitehtäviä tulee lisää. Saatetaan myös huomata, että jokin suunniteltu ratkaisu on perustavanlaatuisesti huono tai sen tilalle on tarjolla parempia ratkaisuja. Tällöin luodaan joko uusi tehtävä listalle ja poistetaan vanha tai päivitetään vanhaa uuden tiedon pohjalta. Näin kävi myös RSS-lukijaa työstettäessä ja sovelluksen suunnitelma ja toiminnallisuus muuttui ja parani työskentelyn alusta loppuun asti.

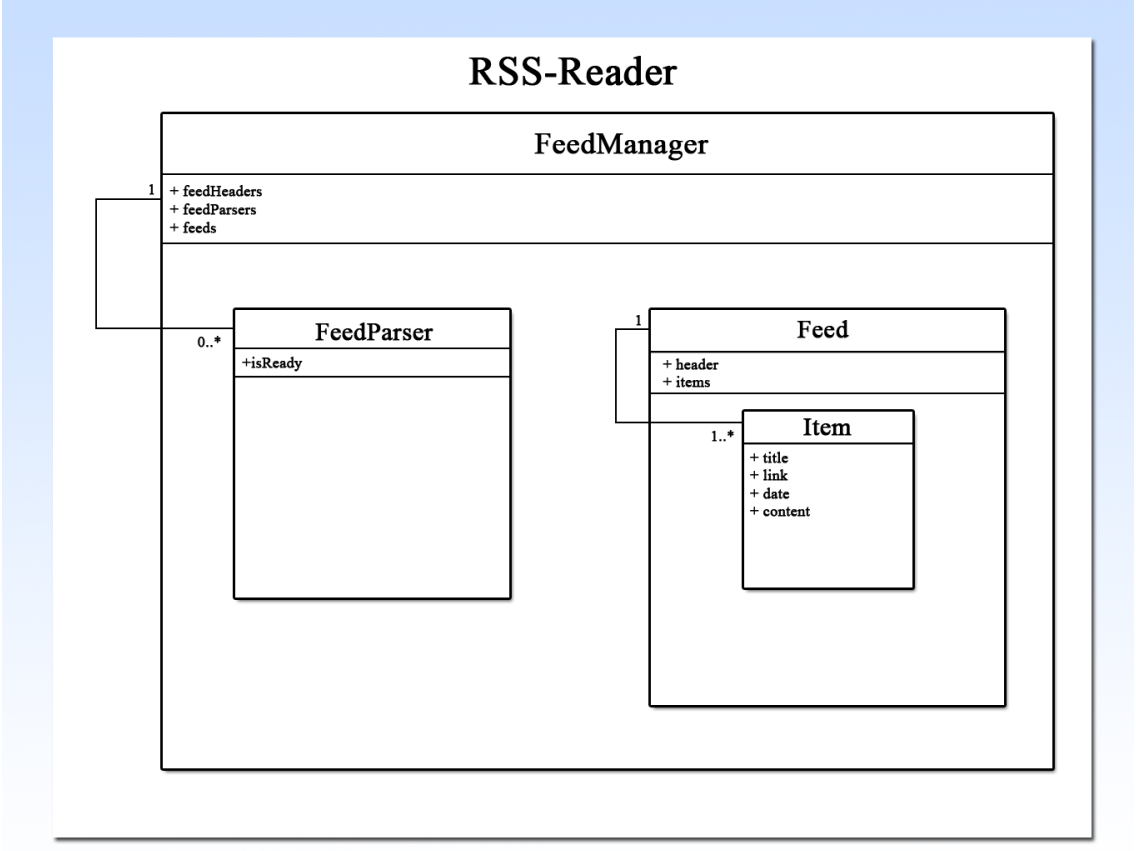
6.1.2 Sovelluksen rakenne



Kuva 1. Sovelluksen tiedonkulku

Kuvassa 1 esitellään tiedonkulkua sovelluksen sisällä. Lähtökohtana on internet, jota sininen "pilvi" kuvastaa ja se on täynnä URL-osoitteita, joiden avulla päästään kiinni RSS-syötedataan. Kuvan kohdassa 1 haetaan tälläinen URL-osoite ja annetaan se FeedManager-luokasta tehdylle oliolle. FeedManager-olio on vastuussa RSS-syötteiden hallinnasta ja on keskeisin tiedonkulun hallinnoija sovelluksessa. Kun FeedManager vastaanottaa uuden URL-osoitteen, se luo tälle osoitteelle FeedParser-luokan olion (Kuvan 1 kohta 2), joka ottaa vastuun URL-osoitteen taustalla sijaitsevan tiedon parsimisesta sovelluksessa hyödynnettävään muotoon. Samalla FeedManager lähettää

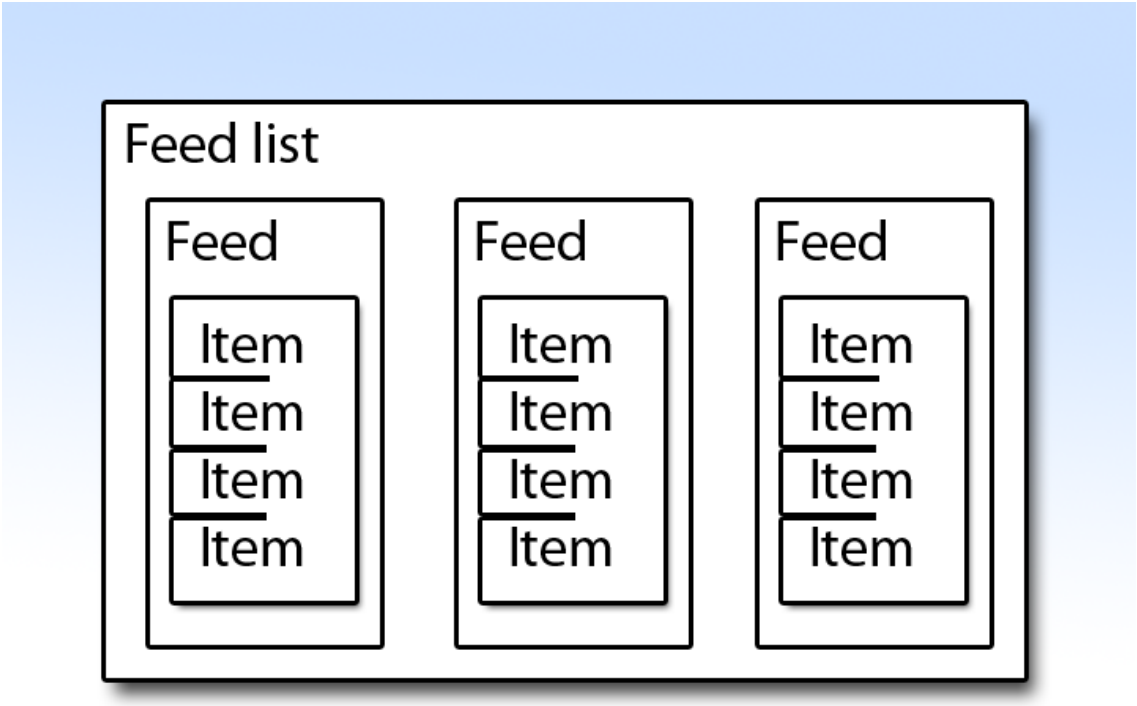
kutsun internetiin ja vastaanottaa sieltä URL-osoitteen taustalla olevan RSS-syötedatan (Kuvan 1 kohdat 3 ja 4). RSS-syötedata on XML-muotoista tietoa ja se lähetetään FeedParser-oliolle parsittavaksi (Kuvan 1 kohta 5). FeedParser-olio parsii datan ja muodostaa siitä Feed-luokkarakenteeseen pohjautuvan olion (Kuvan 1 kohta 6). Tämän jälkeen FeedParser lähettää Feed-olion FeedManagerille, joka vuorostaan siirtää kokoelman kaikista Feed-olioista eteenpäin QML-käyttöliittymään (Kuvan 1 kohdat 7 ja 8). FeedManager lähettää päivitetyn listan Feed-olioista QML-käyttöliittymään aina, kun käyttäjä lisää tai poistaa RSS-syötteen sovelluksesta. QML:ssä Feed-olion tietorakenteen vastaanottaa kaksi QML:ään sisäänrakennettua ListView-tietorakennetta. Ensimmäinen ottaa vastuukseen näyttää käyttäjälle kaikkien RSS-syötteiden otsikot ja jälkimmäinen tuo näyttöruudulle syötekohtaiset alaotsikot.



Kuva 2. Sovelluksen Qt-osuuden rakenne

Kuvassa 2 esitellään sovelluksen taustalla olevan Qt C++ –koodin rakenne. Aikaisemmin mainittiin FeedManager-luokka sisältää FeedParser-, Feed- ja Item-luokat. Kun FeedParser vastaanottaa internetistä tulleen RSS-syötedatan, se muokkaa XML-muodossa olevasta tiedosta Feed-luokan olion, joka puolestaan sisältää lukuisia

Item-luokan olioita. Feed-luokan olio kuvaa yhden internetistä löytyvän RSS-syötteen rakennetta. Se sisältää listan Item-luokasta tehtyjä olioita, jotka kuvaavat yhden RSS-syötteestä löytyvän kirjoituksen tietorakennetta. Tällainen tietorakenne kuvaa sitä muotoa, jossa RSS-syötteet esitetään internetissä. RSS-lukijasovellus pyrkii matkimaan tätä rakennetta (Kuva 3) ja kuljettamaan tietoa sovelluksen sisällä samassa muodossa.



Kuva 3. RSS-syötteiden rakenne sovelluksen sisällä

Oliopohjainen ohjelmistosuunnittelu pyrkii kuvaamaan tietorakenteita tosielämän tavoin ymmärrettävissä kokonaisuuksissa eli olioissa. Qt on pohjimmiltaan oliopohjaista C++:aa, ja näin myös RSS-syötelukija sovellus hyödyntää Qt-osuuden toiminnassaan olioita. Idea QML-puolelle rakennettuun kahden ListView-elementin muodostamaan tietorakenteeseen syntyi keskustelussa Qt-yhteisön jäsenen kanssa ircissä (Internet Relay Chat) freenode verkon #qt-qml-kanavalla. Yhteisön jäsenen tietojen mukaan QML ei tekohetkellä sisältänyt suunniteltua tukea kaksiulotteisille tietorakenteille ja suositteli kahdesta ListView-elementistä muodostuvan tietorakenteen luomista sovelluksen käyttöliittymäosuuteen. Tarkemmin tätä tietorakennetta ja yleisesti QML-ohjelmointia käydään läpi kappaleessa *6.3 QML-ohjelmointi*.

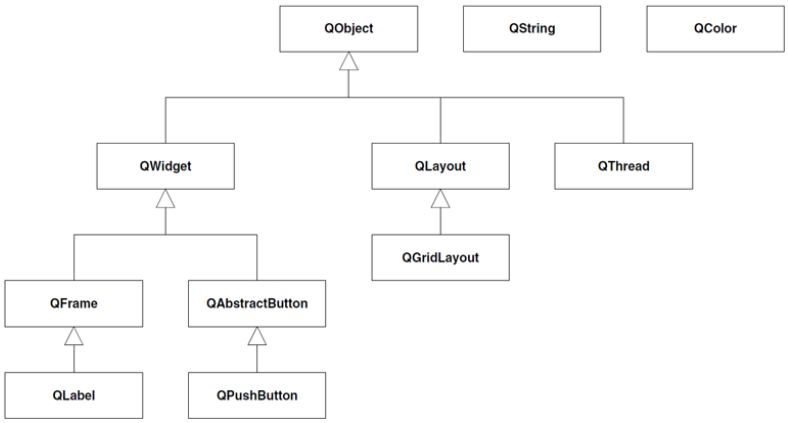
6.2 Qt lyhyesti

Tässä kappaleessa käydään lyhyesti läpi Qt:n olennaisempia seikkoja, jotka auttavat ymmärtämään Qt:n sisäistä rakennetta ja sitä kautta myös helpottavat QML-ohjelmointitaitojen opettelua. Teksti on suunnattu henkilöille, joilla on jo aikaisempaa ohjelmointikokemusta sekä käsitys oliopohjaisesta ohjelmoinnista.

6.2.1 Perintähierarkia

Qt:ssä perintähierarkian ymmärtäminen on hyvin olennaista koko kielen toiminnallisuuden ymmärtämisen kannalta. Hyvin monet Qt:n valmiit luokat, varsinkin widget-tyyliset luokat, perivät jotain kautta QObject-luokan ja perintähierarkia saattaa näin muodostua todella pitkäksikin. QObject-luokka tuo mukanaan tiettyjä Qt:lle olennaisia perustoiminnallisuuksia, kuten esimerkiksi signaalit ja slotit, joista kerrotaan tarkemmin myöhemmin kappaleessa *6.2.3 Qt:n Signaalit ja Slotit*.

Esimerkkinä perintähierarkiasta voisi olla esimerkiksi QPushButton-luokka, jota voidaan käyttää painallusnapin luomiseen. Tämä luokka perii QAbstractButton-luokan, joka puolestaan perii QWidget-luokan ja tämä puolestaan perii QObject-luokan (Kuva 4). Näin QObject-luokan toiminnallisuudet kulkeutuvat aina QPushButton-luokalle asti kolmen perinnän takaa. Qt:ssä on myös luokkia, jotka eivät hyödynnä QObject-luokan mukana perittäviä ominaisuuksia. Qt-sovellusta työstäessä on hyvä tiedostaa käytettyjen valmiiden luokkien perintähierarkia, sillä perintähierarkia paljastaa millaisia toiminnallisuuksia luokka kuljettaa mukanaan (Qt Reference Documentation, Inheritance Hierachy).



Kuva 4. Qt:n perintähierarkia voi jatkua pitkällekin (Molkentin 2007, 41)

QObject-luokan perivät luokat saavat käyttöönsä myös Q_OBJECT-makron. Q_OBJECT makron mainitseminen luokkamäärittelyssä tuo luokalle tiettyjä Qt:n perustoiminnallisuuksia käyttöön. Makro mahdollistaa muun muassa signal-slot-mekanismin sekä sovelluksen kansainvälistämisen useamman kielen implementoinnilla käyttäen tr()-funktiota. Makro tuo käännösvaiheessa Qt:n moc-apusovellukselle (Meta-Object Compiler) tarvittavia toiminnallisuuksia, jotta esimerkiksi edellä mainitut Qt:n ominaisuudet voidaan toteuttaa aidon C++-kielen tavoin (Qt Reference Documentation, The Meta-Object System).

6.2.2 Muistinhallinta

C++-ohjelmoijille muistinhallinasta tutut termit keko- (heap) ja pinomuisti (stack) on Qt:ssä otettu huomioon mielenkiintoisella tavalla. Tavallisesti, esimerkiksi C++-kielessä, ohjelmoijan tulee huolehtia muistinhallinnasta itse kirjoittamalla muistia vapauttavia ”hajoittimia” (destructor). New-operaattorilla luodut oliot varaavat tilaa kekomuistista ja ohjeilmoijan tulisi muistaa kirjoittaa toiminnallisuutta, joka vapauttaa näiden olioiden muistivaraukset. Ilman tällaista muistinvapautusta, kun olio muuttuu sovelluksessa tarpeettomaksi, jäävät oliot varaamaan muistia koko sovelluksen suorituksen ajaksi. Muistivaraukset voivat pahimmillaan täyttää kekomuistin kokonaan aiheuttaen muistin ylivuodon. Qt:ssä perintähierarkia tulee muistinhallinan apuun.

QObject-luokkaan on rakennettu toiminnallisuus, jonka avulla kaikki QObject-luokan perivät oliot huolehtivat automaattisesti lapsiolioidensa vapauttamisesta. QObject-

luokan toiminnallisuuden omaavaa oliota luodessa olion muodostimelle voi määrittää parametrin muodossa kyseisen olion isännän. Isäntä on myös mahdollista jättää määrittämättä, sillä isäntäolion oletusarvo on nolla. Jos isäntäolioksi asettaa jonkin jo olemassa olevan QObject-luokan perivän tai siitä muodostetun olion, huolehtii Qt automaattisesti tämän olion muistin vapauttamisesta silloin, kun sen isännän varaama muisti vapautetaan. Vastaavasti taas sama olio, jolle aikaisemmin asetettiin isäntä, huolehtii omien lapsiensa varaaman muistin vapauttamisesta ja näiden lapset taas omiensa rekursiivisesti. Tätä toiminnallisuutta hyödyntämällä ohjelmoijan tarvitsee huolehtia vain olioiden isäntä-lapsi-suhteista ja Qt hoitaa muistinhallinnan automaattisesti. (Blanchette & Summerfield 2008, 28).

6.2.3 Qt:n signaalit ja slotit

Qt:n signal-slot-mekanismi on yksi Qt:n merkittävimpiä ominaisuuksia. Sen avulla Qt:n oliot voivat lähettää tietoja toisilleen ja näin niin sanotusti sitoa Qt-olioita toisiinsa ilman, että oliot itse ottavat tätä sitomista huomioon toiminnallisuudessaan. Qt-luokkaan rakennetut slotit ovat rakenteeltaan avain kuin tavallisia funktioita. Ne voivat olla virtuaalisia, ne voidaan ylikirjoittaa aliluokassa ja niitä voi kutsua kuten tavallisia funktioita. Olennainen ero tavallisen funktion ja slotin välillä on, että slotin voi yhdistää Qt-olion lähettämään signaaliin (Blanchette & Summerfield 2008, 20).

Slotin yhdistäminen signaaliin voi esimerkiksi luoda toiminnallisuuden, jossa yksi olio ilmoittaa toiselle jonkin tietyn toiminnallisuuden olevan valmis, kuten esimerkiksi XML-tiedoston parsimisen. Parsimisen jälkeen parsijaolio lähettää signaalin, jonka jokin toinen olio havaitsee ja suorittaa oman toiminnallisuutensa. Tämä toiminnallisuus vaatii XML-parsinnan loppuunsuorituksen. Signaali voi olla niin sanotusti ”tyhjä” tai se voi kuljettaa mukanaan yksinkertaisia muuttujia, listoja tai vaikkapa olioita yhdeltä oliolta toiselle. Jos signaali kuljettaa tietoa slotille, on parametrien oltava yhteensopivia (Blanchette & Summerfield 2008, 21).

Signaalin ja slotin yhdistäminen onnistuu QObject-luokan connect-funktiolla (Koodiesimerkki 1). Connect-funktio tarvitsee parametreikseen lähettäjäolion, lähettäjäolion käyttämän signaalin parametreineen, vastaanottajaolion sekä vastaanottajaolion käyttämän slotin parametreineen. Signal-slot-yhdistämisessä on

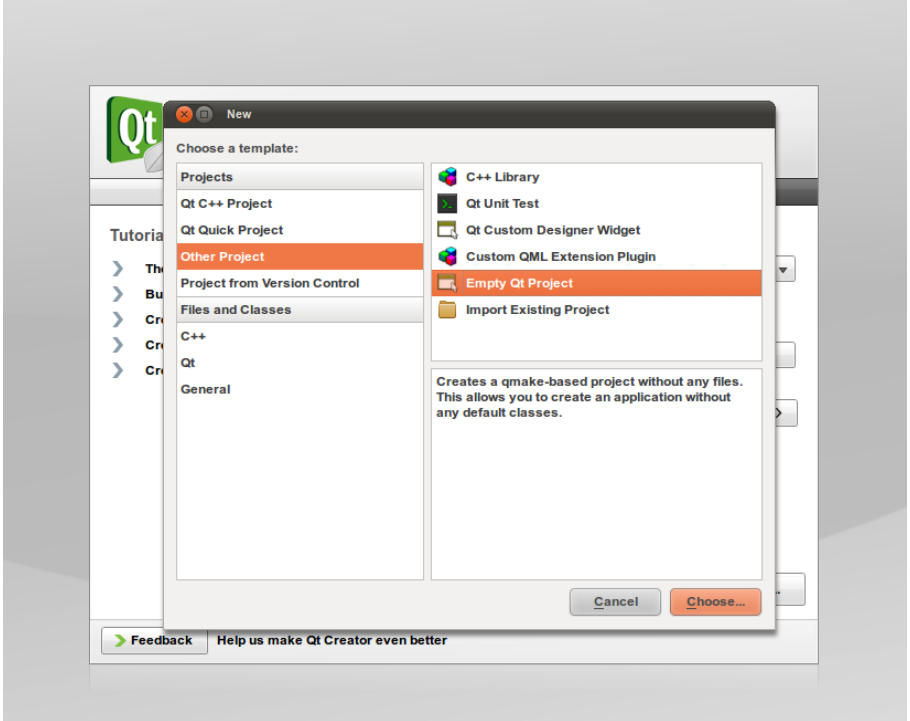
huomioitava, että parametrien nimiä ei tule mainita, pelkkä tietotyyppien kuvaus riittää. Olennaista on myös huomata, että signal-slot-mekanismi sallii useiden signaalien yhdistämisen samaan slottiin ja vastaavasti yhden signaalin yhdistämisen useisiin slotteihin. Signal-slot-mekanismi on käytössä vain niillä olioilla, jotka perivät QObject-luokan ja ohjelmoijan on muistettava mainita Q_OBJECT-makro luokan määrittelyssä (Blanchette & Summerfield 2008, 22).

```
connect(sender,   SIGNAL ( valueChanged( int ) ),
        receiver, SLOT   ( setValue( int )     )
        );
```

Koodiesimerkki 1. Signaalin ja slotin yhdistäminen

6.2.4 Qt-projektin luominen

Qt-projektin luominen QML-sovellusta tukemaan on nykyisillä työkaluilla erittäin helppoa. Nokian Qt SDK:n asennettua käynnistetään Qt Creator ja tehdään työkalulla uusi projekti. Uuden projektin voi tehdä valitsemalla Qt Creatorin File-valikosta ”New File or Project...” –valinnan tai vaihtoehtoisesti painamalla CTRL+N näppäinyhdistelmää. Uuden projektin tekemisessä Qt Creator tarjoaa monia ennalta luotuja pohjia tietynlaisen sovelluksen tekemiseen. Ymmärtääkseen Qt-projektin luomisprosessia parhaiten on suositeltavaa aloittaa mahdollisimman alusta, joten tässä tapauksessa oikea valinta löytyy ”Other Projects” –valikon alta projektipohja ”Empty Qt Project” (Kuva 5).



Kuva 5. Empty Qt Project

Nimensä mukaisesti tämä projektipohja on mahdollisimman tyhjä. ”Empty Qt Project” –pohja luo valmiiksi .pro-tiedoston, johon tullaan määrittelemään kaikki tarpeellinen tieto tulevasta projektista, jotta siitä saadaan luotua oikeanlainen Makefile-tiedosto (Qt Reference Documentation, qmake Project Files). Makefile-tiedoston käyttöä ja tarkoitusta käsitellään myöhemmin kappaleessa *6.6.1 Kääntäminen oikealle alustalle*. Ennen kuin ruvetaan työstämään oikeata toiminnallisuutta sovellukseen on hyvä tarkastella .pro-tiedostoa ja varmistaa, että siinä on kaikki kunnossa.

Koska tarkoituksena on luoda niin sanottu ”Qt C++ –wrapperi” eli kääre, joka mahdollistaa QML-sovelluksen binäärimuotoon kääntämisen, pitää .pro-tiedostoon määritellä mukaan käännettävinä Qt-kirjastoina ainakin core sekä declarative. Tämän lisäksi RSS-lukijasovellus hyödyntää internet-yhteyttä, joten on tarpeellista määritellä network-kirjasto mukaan projektiin (Koodiesimerkki 2).

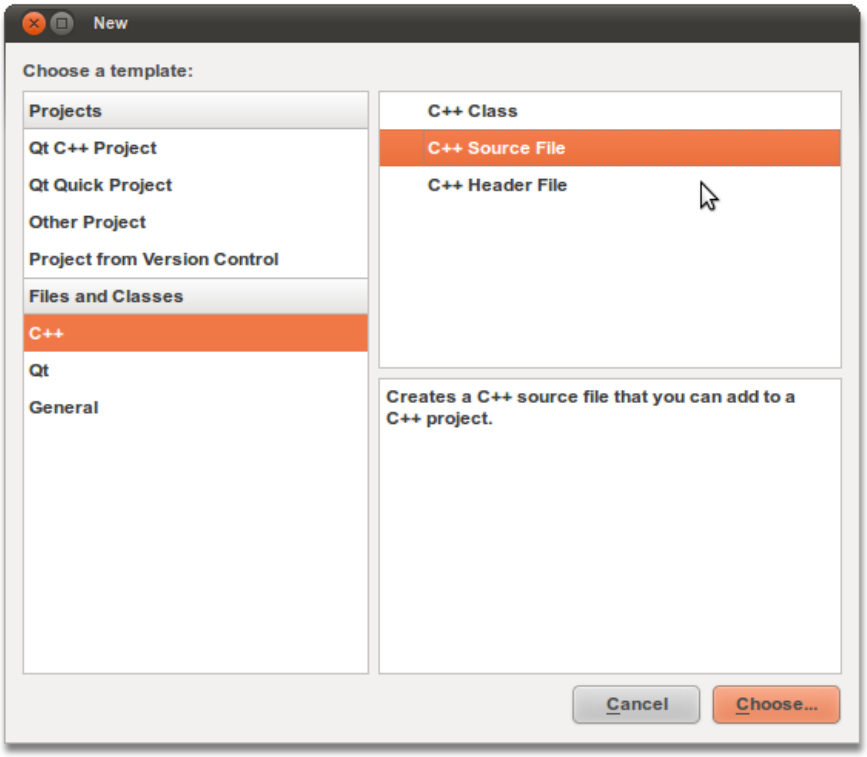
```
// .pro file
QT += core declarative network
```

Koodiesimerkki 2. RSS-lukijan Qt-kirjastot

Core-kirjaston lisääminen antaa ohjelmoijalle käyttöön Qt:n perustoiminnallisuuden ja declarative-kirjasto on lisättävä, jotta käyttöön saadaan QML-pohjaisen graafisen

käyttöliittymän näyttäminen Qt:sta käsin toisin sanoen QDeclarativeView-luokka. Qt Creator –sovellus huomauttaa mahdollisista ohjelmointivirheistä sovellusta työstettäessä ja ilman edellä mainittujen kirjastojen lisäämistä Qt Creator ei suostu kääntämään työstettävää sovellusta.

Seuraavaksi on lisättävä C++:sta tuttu main.cpp-tiedosto, johon toteutetaan sovelluksen käynnistyslogiikka. Tarvittava main.cpp-tiedosto luodaan valitsemalla Qt Creatorin File-valikosta jälleen ”New File or Project...” –valinta. Valmiiden projektipohjien alapuolella löytyy toinen valikko, jonka otsikko ”Files and Classes” kertoo kaiken tarvittavan. Näistä main.cpp-tiedostoa varten oikea valinta on ”C++ Source File” (Kuva 6), sillä main.cpp ei ole header-tiedosto eikä tarvitse header-tiedostoa.



Kuva 6. C++ Source File

Qt Creator pyytää ohjelmoijaa laittamaan manuaalisesti main.cpp-tiedoston nimen, jonka jälkeen Next- ja Finish-painikkeita painamalla tiedosto luodaan projektiin. Main.cpp-tiedosto ilmestyy nyt projektin Sources-hakemiston alle ja on aluksi tyhjä, mutta jos nyt vilkaistaan aiemmin tehtyä .pro-tiedostoa, on Qt Creator lisännyt sinne automaattisesti uuden projektitiedostomerkinnän SOURCES-muuttujaan (Koodiesimerkki 3).

```
// .pro file
QT += core declarative network

SOURCES += \
    main.cpp
```

Koodiesimerkki 3. Uusi tiedosto projektissa

Nämä vaiheet läpi käytyään ohjelmoija saa aikaan Qt-projektipohjan, jonka pohjalta hän voi ryhtyä työstämään projektia eteenpäin. Kappaleessa *6.4 Qt:n lisääminen QML:n taustalle* käsitellään tästä vaiheesta seuraavia toimia, jotta ohjelmoija saa aikaiseksi valmiin QML-ohjelman.

6.3 QML-ohjelmointi

Ohjelmistojen tekemiseen orientoituneelta ihmiseltä ei mene kauan oppia QML:n alkeet. Kieli tavallaan selittää itse itseään, koska se on deklaratiivinen (deklaratiivisuudesta tarkemmin kappaleessa *4.2 QML*) ja näin tavallisimpien asioiden tekeminen on erittäin helppoa. Yksinkertainen esimerkki QML-sovelluksesta voisi olla vaikkapa RSS-syötelukijaa varten tehty sulkemisnappi (Koodiesimerkki 2). Syntaksi muistuttaa hyvin paljon Cascading Style Sheets tyyliohjekieltä (w3Schools, CSS Tutorial). Tämän takia voidaankin olettaa että web-sivujen tekijöille QML:n omaksuminen on helppoa CSS kielen tuttuuden takia.

Koodiesimerkissä 4 luodaan punainen laatikko. Sen id:ksi asetetaan closeButton, jotta elementtiä voidaan kutsua muualta sovelluksesta ja sen kooksi määritetään 100 pikseliä leveyttä ja 40 pikseliä korkeutta. Taustaväriksi asetetaan punainen ja elementti pakotetaan ankkureilla oman isäntäelementtinsä oikeaan yläkulmaan (Ankkureista tarkemmin kappaleessa *6.3.1 Ankkurit*). Laatikon sisälle luodaan QML:stä valmiina löytyvä MouseArea-elementti, joka rekisteröi napin painalluksia ja elementistä valmiina löytyvän onClicked-slotin toiminnaksi määritellään Qt-sovelluksen sulkeminen (Qt:n sloteja käsitellään tarkemmin kappaleessa *6.2.3 Qt:n signaalit ja sloti* sekä *6.3.4 QML:n signaalit ja slotit*). Myöhemmässä sovelluksen tekovaiheessa esimerkissä luotu sulkemistoiminnallisuus voidaan helposti korvata graafisesti näyttävämmällä kuvalla, poistamalla punaisen taustavärin määrittely ja lisäämällä Image-elementti laatikon sisään.

```
Rectangle {
    id: closeButton

    width  : 100
    height : 40
    color  : "red"

    anchors.top   : parent.top
    anchors.right : parent.right

    MouseArea {
        anchors.fill: parent

        onClicked: {
            Qt.quit()
        }
    }
}
```

Koodiesimerkki 4. RSS-lukijan sulkunappi

QML-kielen sekaan voi myös laittaa JavaScript-koodia, joka antaa deklaratiivisen muodon lisäksi monimutkaisempia toiminnallisia ominaisuuksia QML-ohjelmoinnille. Kuten aiemmin mainitun CSS:n, myös JavaScriptin voi olettaa olevan tuttu web-sivujen tekijöille. Koodin asettaminen QML:n sekaan toimii tiettyjen sääntöjen alaisesti, joten sitä ei voi laittaa aivan mihin tahansa. QML:ssä voi luoda JavaScript-funktioita koodin sekaan tai tuoda ulkoisia JavaScript tiedostoja ja käyttää niiden sisällä olevia funktioita. Myös tietyt QML:n sisäänrakennetut toiminnot hyväksyvät JavaScript-koodia normaalin QML-syntaksin sijasta. Esimerkiksi MouseArea-elementin sisäänrakennettu onClicked -toiminto voi ottaa sisälleen JavaScript-koodia (Koodiesimerkki 5).

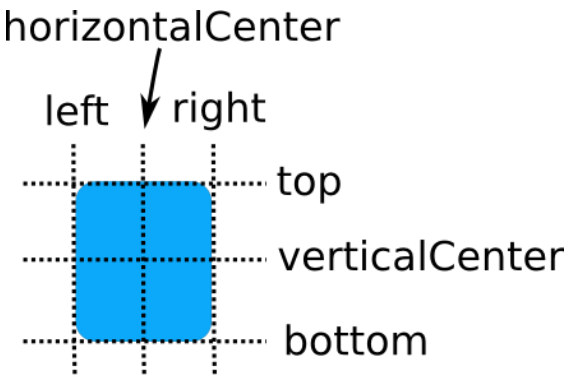
```
Rectangle {
    id: closeButton

    width  : 100
    height : 40
    color  : "red"

    anchors.top   : parent.top
    anchors.right :  parent.right

    MouseArea {
        anchors.fill: parent

        onClicked: {
            if(closeButton.color == "red") {
                console.log("red button was pressed");
                Qt.quit();
            }
        }
    }
}
```

Koodiesimerkki 5. JavaScript koodia QML:n seassa

Kaikki elementit QML:ssä ovat suorakulmion muotoisia. Visuaaliset elementit QML:ssä periytyvät Item-elementistä, joka määrittelee tiettyjä perusarvoja, kuten x- ja y-pisteiden sijainnit sekä leveyden ja korkeuden kaikille sen periville elementeille. Kirjoitushetkellä QML-elementit ovat siis pohjimmiltaan nelikulmaisia, mutta myöhemmissä versioissa asiaan saattaa tulla muutos. Pyöreitä muotoja saadaan aikaan näitä neliskulmaisia elementtejä pyöristämällä. Pyöristämiseen on olemassa oma parametrinsä, joka aiheuttaa reunojen pyöristymistä esimerkiksi Rectangle-elementissä. Parametrin nimi on radius.

6.3.1 Ankkurit

Suorakulmaisten elementtien lisäksi QML sisältää näistä peruspalikoista rakennettuja monimutkaisempia kokonaisuuksia kuten taulukoita sekä rivi- ja sarake-elementtejä. Nämä elementit helpottavat tietynlaisten toisiinsa suhteessa olevien elementtikokonaisuuksien luomista, kuten esimerkiksi kuvien riviin asettamista. Näiden taulukko-, rivi-, ja sarake-elementtien lisäksi QML tarjoaa keinon ohjelmoijalle itselleen asemoida visuaaliset elementit haluamaansa järjestykseen. Tämä onnistuu koodiesimerkissä 4 mainittujen ankkureiden avulla. Ankkurit tarjoavat kiintopisteen elementin neljään sivuun sekä horisontaaliseen ja vertikaaliseen keskustaan (Kuva 7).



Kuva 7. QML-ankkurit (Qt Reference Documentation, Anchor-based Layout in QML)

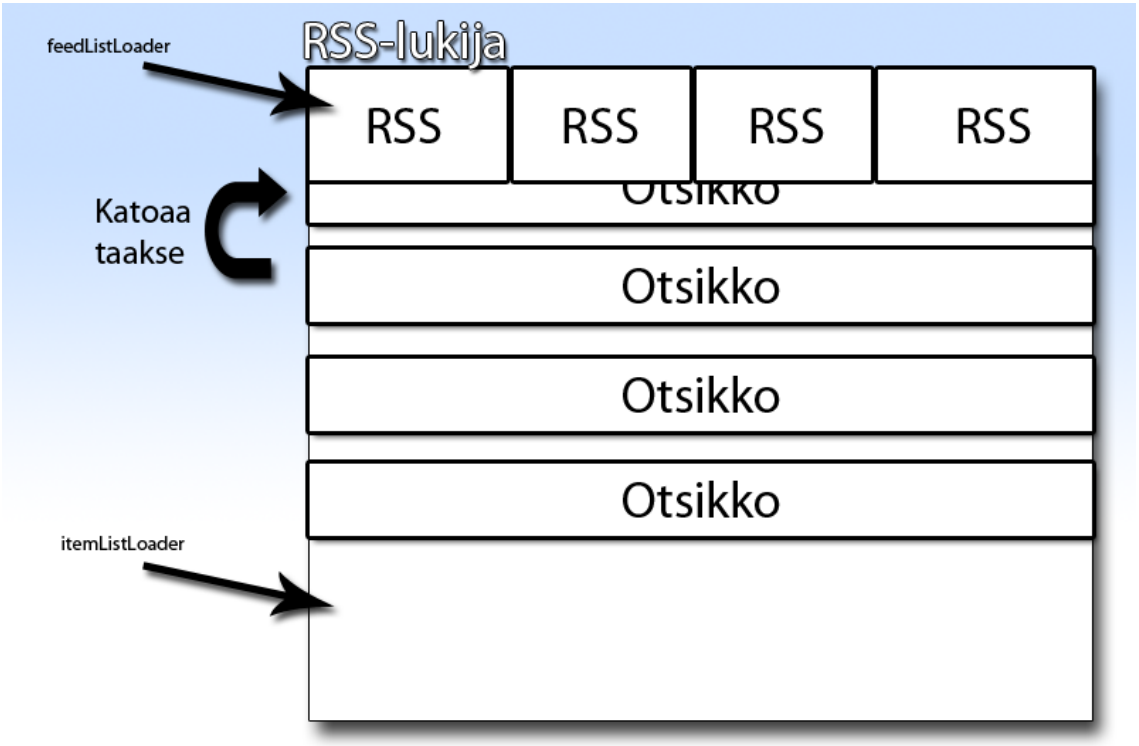
Ankkureita käyttämällä yhden elementin sivun voi kiinnittää toisen elementin sivuun. Suhde sivujen välillä pysyy, vaikka vain toiselle elementille tehtäisiin muutoksia. Kiinnittämällä esimerkiksi virheän neliön oikea sivu sinisen neliön vasempaan sivuun ja muuttamalla sinisen neliön x-koordinaattia, myös vihreä neliö liikkuu samassa suhteessa pitäen neliöiden sivut kiinni toisissaan. Ankkureiden käytössä tarkoituksena on luoda

toisistaan riippuvia asetelmia ja suunnitella toiminnallisuus niin, että tiettyä elementtiä siirtelemällä muu ympärillä oleva käyttöliittymä muokkautuu sen mukaisesti. Ankkureiden käytössä rajoituksena on kuitenkin se, että yhden elementin voi kiinnittää ankkureilla vain sen omaan isäntä- tai sisarelementtiin. Isäntäelementit ovat niitä, joiden sisällä ankkuroitava elementti on ja isännän sisällä olevat elementit ovat keskenään sisarelementtejä (Qt Reference Documentation, Anchor-based Layout in QML).

RSS-lukijasovelluksessa ankkureita on käytetty muun muassa kakkien RSS-syötteiden nimet ja yhden nimen alle kuuluvat otsikot sisältävien elementtien kiinnittämiseen toisiinsa. Koodiesimerkissä 6 on pätkä RSS-lukijasovelluksen koodia, jossa Loader-elementteihin sijoitetut komponentit kiinnitetään toisiinsa ankkureilla. Ensimmäinen Loader-elementti eli itemListLoader sisältää yhden RSS-syötteen kaikki otsikot. Jälkimmäinen feedListLoader-elementti sisältää kaikkien sovelluksessa olevien RSS-syötteiden nimet. Esimerkissä huomioitava seikka on, että koodissa alempana kirjoitettu feedListLoader-elementti luodaan syvyyshierarkisesti eli z-indeksiltään ylempänä kirjoitetun päälle, joten käyttäjän kelatessa ylöspäin kaikkia itemListLoader-elementin sisältämiä RSS-syötteiden otsikkoja ylöspäin liikkuvat otsikot katoavat myöhemmin luodun feedListLoader-elementin alle. Koodissa ensin luodun itemListLoader-elementin yläosa on kiinnitetty ankkureilla jälkimmäisen feedListLoaderin alaosaan. Elementit näkyvät käyttöliittymässä niin, että feedListLoader on ylempänä ja itemListLoader alempana. Näin saadaan aikaiseksi se, että kaikkien RSS-syötteiden nimet ovat sovelluksen yläosassa ja yhden nimen sisältämät otsikot ovat otsikoiden alapuolella niin, että ne katoavat kelatessa nimien alle (Kuva 8).

```
Loader {
    id: itemListLoader

    width  : parent.width
    height : parent.height - feedListLoader.height

    anchors.top:  feedListLoader.bottom
}
Loader {
    id: feedListLoader

    height : 80
    width  : parent.width
}
```

Koodiesimerkki 6. Ankkurien käyttö RSS-lukijan QML-koodissa



Kuva 8. Ankkureiden käyttö RSS-lukijan käyttöliittymässä

6.3.2 Omat elementit

QML sisältää monia valmiita elementtejä, jotka ovat jaettu loogisiin osioihin. (Qt Reference Documentation, List of QML Elements). Näistä elementeistä alkajalle olennaisimpina mainittakoon Rectangle, Text, MouseArea, Column, Row, Grid, Image, Timer sekä Gradient elementit. Näitä elementtejä ohjelmoija tulee todennäköisesti käyttämään hyvin useassa projektissa, sillä ne tuovat kieleen todella perustavanlaatuisia toiminnallisuuksia. Dynaamisuutta ja eloisuutta QML-ohjelmiin saa State, Transition ja Animation toiminnallisuuksilla, joita esitellään myöhemmin.

QML:ään on mahdollista rakentaa myös itse omia elementtejä. Elementtien rakentaminen onnistuu sekä taustalla olevan Qt C++ –kirjastojen avulla taikka sitten suoraan QML:ssä ilman C++-kielen käyttöä. Tässä opinnäytetyössä keskitytään jälkimmäiseen tapaan. Mikä tahansa QML-elementteihin perustuva toiminnallisuus voi olla itse tehty uusi QML-elementti, jos sen laittaa tiedostoon ”<Nimi>.qml”, jossa <Nimi> on uuden elementin nimi. Näin tehdyt QML-elementit tulevat automaattisesti käyttöön ja kutsuttavaksi muille QML-tiedostoille, jotka sijaitsevat samassa hakemistossa. Projektin laajuiseen käyttöön uudet elementit saa parhaiten luomalla

projektiin qmldir-tiedoston. Tätä tapaa käydään läpi myöhemmin kappaleessa *6.4 Qt:n lisääminen QML:n taustalle*.

Tavallisin ja helpoin esimerkki uudesta QML-elementistä on nappielementin luominen, sillä QML ei sellaista valmiina vielä sisällä ja sellaisen tekeminen on hyvin monessa sovelluksessa tarpeellista. Nappielementin, eli englanniksi Button, saa käyttöön tekemällä Button.qml tiedoston samaan hakemistoon kuin projektin muut QML-tiedostot. Hyvin yksinkertaisen nappielementin luominen QML:ään on erittäin helppoa (Koodiesimerkki 7).

```
// Button.qml

Rectangle {
    property string name: ""

    width  : 100
    height : 50
    color  : "blue"

    MouseArea {
        anchors.fill: parent
        onClicked: console.log("Click")
    }
}
```

Koodiesimerkki 7. Button-elementti

Yllä olevassa koodissa luodaan neliön muotoinen Rectangle-elementti, jonka leveys on 100 pikseliä, korkeus 50 pikseliä ja taustaväri on sininen. Koko elementti täytetään MouseArea-elementillä määrittämällä sille fill-ankkuri. Ankkuri täyttää MouseArean koko parent- eli isäntäelementin, joka tässä tapauksessa on edellä mainittu Rectangle-elementti. MouseArea-elementille asetetaan vielä hiiren painalluksen aistiva onClicked-toiminnallisuus, joka kirjoittaa konsoliin ”Click”-tekstin. Konsoliin kirjoittaminen on tässä tapauksessa vain testausta varten tehty toiminnallisuus, jotta nähdään onko napin painallus oikeasti rekisteröity. Todellisessa sovelluksessa tällainen tekstin kirjoittaminen konsoliin on tarpeetonta.

Tarkkasilmäisimmät saattavat Koodiesimerkki 7:ssä huomata, että onClicked-toiminnallisuuden perästä on jätetty aaltosulut pois, vaikka ne vielä Koodiesimerkki 3:ssa olivat. Aaltosulut toimivat QML:ssä JavaScript-toiminnallisuuden rajaamisessa yhdeksi lohkoksi, kun JavaScript-koodi on useamman rivin pituinen. Yhden rivin pituisen koodinpätkän tapauksessa kuitenkin aaltosulut ovat tarpeettomat.

Itse määriteltyihin elementteihin on myös mahdollista luoda uusia ominaisuuksia property-toiminnallisuudella. Property toimii niin sanottuna luokkamuuttujana ja sitä voi käyttää elementin ulkopuolelta pistenotaatiolla. Koodiesimerkissä 7 annetaan Button-elementille property name, jonka tietotyyppi on string. Muualla käytettäessä tätä Button-elementtiä voidaan sille asettaa uusi leveys ja samoin erilainen nimi pistenotaatiota käyttäen. Koodiesimerkki 8:ssä luodaan aikaisemmin määritellystä Button.qml-tiedostosta Button-elementti ja sille asetetaan uusi leveys sekä nimi. Korkeus on Button.qml-tiedostossa määritelty 50 pikseliksi, väri pysyy edelleen sinisenä ja MouseArea-toiminnallisuus kulkee elementin mukana. Tässä esimerkissä on tärkeää huomata, että tavallisella Rectangle-elementillä ei ole name-muuttujaa ja sellaisen kutsuminen aiheuttaisi virheen. Button-elementissä on kuitenkin luotu name-muuttujan käytettäväksi.

```
Button {
    width : 200
    name  : ”LongButton”
}
```

Koodiesimerkki 8. Oman elementin käyttö

6.3.3 Tilat eli statet ja siirtymät eli transitiot

Elämää ja dynaamisuutta QML:ään saadaan luomalla elementteihin tiloja State-toiminnallisuudella ja määrittelemällä siirtymiä eli transitioita tilojen muutosten välille. Eri tilojen eli Statejen välillä voidaan esimerkiksi piilottaa joitain elementtejä ja asettaa näkyviin toisia, käynnistää tai pysäyttää jollekin elementille asetettu animaatio, kutsua esimerkiksi JavaScript-funktioita tai muita sovellukseen rakennettuja toiminnallisia lohkoja tai vaikkapa muutta jonkun elementin sisäisiä arvoja. Koodiesimerkissä 9 on esitelty koodinpätkä, jolla saadaan aikaiseksi vihreä neliö isomman sinisen neliön sisälle ja vihreää neliötä painamalla muutetaan se punaiseksi ja päinvastoin. Muutos väriin on saatu aikaiseksi hyödyntämällä State-toiminnallisuutta (Qt Reference Documentation, QML States).

```
Rectangle {
    width  : 360
    height : 360
    color  : "blue"

    Rectangle {
        id: smallRect

        width  : 100
        height : 50
        color  : "green"

        anchors.centerIn: parent

        states: [
            State {
                name: "red"
                PropertyChanges { target: smallRect; color: "red" }
            }
        ]

        MouseArea {
            anchors.fill: parent

            onClicked: {
                console.log("clicked")
                if(smallRect.state == "")
                    smallRect.state = "red"
                else
                    smallRect.state = ""
            }
        }
    }
}
```

Koodiesimerkki 9. State-elementin käyttö

Koodiesimerkki 9:ssä on hyvä huomioida muutamia kohtia. Ensinnäkin State-toiminnallisuus laitetaan lapsielementiksi sen isäntäelementin sisään, jolle kyseinen State halutaan asettaa. State-elementille annetaan nimi sekä tässä tapauksessa määritellään myös PropertyChanges-elementti, jolla voidaan muokata jotain tiettyä sovelluksessa olevaa elementtiä. PropertyCanges-elementille on asetettava kohde eli target, jonka jälkeen voidaan luetella target-määritelyssä asetetun kohteen muuttujia ja niiden uusia arvoja. Koodiesimerkissä annetaan PropetyChanges-elementille kohteeksi sen isäntäelementti id:n perusteella, joka on ylempänä määritelty ”smallRect”. Tämän jälkeen käsketään target-määrittelyssä annetun elementin väri muuttumaan punaiseksi, kun kyseisen Staten omaava elementti on ”red”-tilassa. Koska State-elementti on smallRect-elementin lapsielementti, voidaan nyt hyödyntää smallRect-elementin tiloja asettamalla sen tilaksi ”red”. Tähän tilaan pääsee käsiksi muokkaamalla smallRect.state-muuttujaa, jota muokataankin MouseArea-elementin onClicked-funktion sisällä. On hyvä huomata, että kun smallRect-elementin väri muutetaan takaisin punaisesta vihreäksi, tehdään onClicked-funktion sisällä tarkistus siitä onko state-muuttuja tyhjä

string-muuttuja. Jokaisen elementin oletustila eli default state on tyhjä string (Qt Reference Documentation, QML States).

Koodiesimerkissä 10 on lisätty edelliseen koodiesimerkkiin animoitu muutos värin vaihtumiseen. Tämä on saatu aikaan Transition-elementillä. Elementin tarkoitus on määritellä animaatiota tilojen vaihtumisen välille. Koodiesimerkissä on luotu transitions-taulukko, jonka sisälle asetetaan kaikki isäntäelementille tarkoitetut Transition-määrittelyt. Taulukko sisältää kaksi Transition-elementtiä sekä isäntäelementin muutokselle "red"-tilaan että siitä pois. Molempien Transitio-elementtien sisään luodaan PropertyAnimation-elementti, joka määrittelee tietyn ominaisuuden muutokseen kiinnitetyn animaation. Tässä tapauksessa ominaisuus on color-muuttuja.

```
Rectangle {
    width  : 360
    height : 360
    color  : "blue"
    Rectangle {
        id: smallRect

        width  : 100
        height : 50
        color  : "green"
        anchors.centerIn: parent

        states: [
            State {
                name: "red"
                PropertyChanges { target: smallRect; color: "red" }
            }
        ]
        transitions: [
            Transition {
                to: "red"
                PropertyAnimation { properties: "color"; duration: 500;
                                                easing.type: Easing.InOutQuad}
            },
            Transition {
                to: ""
                PropertyAnimation { properties: "color"; duration: 500;
                                                easing.type: Easing.InOutQuad}
            }
        ]
        MouseArea {
            anchors.fill: parent
            onClicked: {
                console.log("clicked")
                if(smallRect.state == "")
                    smallRect.state = "red"
                else
                    smallRect.state = ""
            }
        }
    }
}
```

Koodiesimerkki 10. QML projektin elävoittäminen animaatioilla

Koodiesimerkin 10 animaatio on tehty vaikeimmalla mahdollisella tavalla. Molempiin tilamuutoksiin on luotu oma Transition-elementti hallinnoimaan muutosta sekä "red"-tilan että oletustilan välillä. Vaikka tällainen lähestymistapa antaakin ohjelmoijalle enemmän hallintaa tilojen käsittelyn välillä, tarjoaa QML myös helpompia ratkaisuja. Koodiesimerkissä 11 on korvattu edellinen tapa animoida yhdellä Transition-elementillä. Tässä tapauksessa ei aseteta to-määrittelyä Transition-elementille, jolloin QML luo automaattisesti kyseisen animaation aina kun isäntäelementin color-ominaisuus muuttuu. Sama animaatiotoiminnallisuus voidaan saada aikaan myös QML:n Behavior-elementillä (Koodiesimerkki 12), mutta sen käyttö on rajatumpaa kuin Transition-elementin. Behavior-elementtiä ei voi sitoa isäntäelementin tilamuutoksiin ja jos isäntäelementti sisältää Transition-elementin, joka on sidottu samaan ominaisuuteen kuin Behavior-elementti, ylikirjoittaa Transition-elementin toiminnallisuus aina Behavior-elementin.

```
transitions:
            Transition {
                PropertyAnimation {
                    properties: "color";
                    duration: 500;
                    easing.type: Easing.InOutQuad
                }
            }
```

Koodiesimerkki 11. Helpompi tapa animoida

```
Behavior on color {
        PropertyAnimation {
            duration: 500;
            easing.type: Easing.InOutQuad
        }
}
```

Koodiesimerkki 12. Behavior-elementti

RSS-lukijasovelluksessa tiloja ja animaatioita on käytetty yhden RSS-syötteen otsikon sisältämän tekstin esiintuomiseen. Koodiesimerkki 13 sisältää FeedItem-elementin tilat ja transitiot. FeedItem-elementti on itse luotu QML-elementti, jota hyödynnetään ListView-elementin delegaattina. Delegaatti ListView-elementissä toistaa itseään ListView-elementin sisältämien elementtien määrän verran. Toisin sanoen FeedItem-elementtiä käytetään määrittelemään ulkoasupohja jokaiselle RSS-syötteen sisältämälle otsikolle ja otsikkoon liittyvälle tekstille. FeedItem-elementissä on luotu MouseArea-elementti, jota painamalla FeedItem-elementin tila muuttuu "Details"-tilaan ja siitä pois.

Koodiesimerkissä 13 nähdään, kuinka "Details"-tilaan asetetut PropertyChanges-elementit muuttavat FeedItem-elementin ominaisuuksia. Alempana määritellään animaatioita tilamuutoksen muokkaamille ominaisuuksille, jotta saadaan sovellukseen elävyyttä. Oheinen koodi luo toiminnallisuuden, jossa Kuvassa 8 esitellyn itemListLoaderin sisältämää otsikkoa painamalla otsikko laajenee täyttämään koko itemListLoader-elementin ja tuo esiin otsikkoon liitetyn tekstin.

```
states: State {
        name: "Details"
        PropertyChanges { target: background; color: "white" }

        // Make details visible
        PropertyChanges { target: entry; detailsOpacity: 1; x: 0 }

        // Fill the entire list area with the detailed view
        PropertyChanges { target: entry; height: entry.ListView.view.height }

        // Move the list so that this item is at the top.
        PropertyChanges { target: entry.ListView.view; explicit: true;
                          contentY: entry.y }

        // Disallow flicking while we're in detailed view
       PropertyChanges { target: entry.ListView.view; interactive: false }
}

transitions: Transition {
    // Make the state changes smooth
    ParallelAnimation {
        ColorAnimation  { property: "color"; duration: 500 }

        NumberAnimation { duration: 300;
                          properties: "detailsOpacity,
                                       x,
                                       contentY,
                                       height,
                                       width"
                        }
    }
}
```

Koodiesimerkki 13. FeedItem-elementin tilat ja animaatiot

6.3.4 QML:n signaalit ja slotit

Signaalit ja slotit, jotka ovat Qt C++:sta tuttu tapa lähettää tietoa kahden komponentin välillä, on käytettävissä myös QML:n elementeissä. Aikaisemmissa esimerkeissä on käytetty MouseArea-elementin onClicked-funktiota, joka itse asiassa on slotti, joka vastaanottaa MouseArea-elementin lähettämän clicked-signaalin. Tällaisia signal-slot-yhdistelmiä voi tehdä myös ohjelmoijan itse luomiin QML-elementteihin (Koodiesimerkki 14).

```
Rectangle {
    width  : 500
    height : 400

    Rectangle {
        id: square

        width  : 100
        height : 50
        color  : "#0000ff"

        signal green()

        MouseArea {
            anchors.fill: parent
            onClicked:  {

                if(square.color == "#0000ff") {
                    square.color = "#008000";
                    square.green();
                }
                else
                    square.color = "#0000ff";
            }
        }

        onGreen: console.log("green")
    }
}
```

Koodiesimerkki 14. Signaalit ja slotit QML:ssä

Koodiesimerkissä 14 on luotu Rectangle-elementti, joka lähettää ja ottaa vastaan oman signaalinsa. Signaali on tehty elementin määrittelyssä signal-komennolla ja signaalin vastaanottava slot määritellään myös samaan elementtiin. Kaikkien signaalien slotit ovat aina muotoa ”on<Signaalin nimi>”, esimerkiksi ”onClicked”. On-liite kuuluu asettaa jokaiseen slot määrittelyyn ja <Signaalin nimi>-kohta korvataan halutun signaalin nimellä. Tässä tapauksessa onGreen-slot ottaa vastaan green-signaalin. Green-signaali lähetetään yksinkertaisesti kutsumalla signaalia kuten mitä tahansa muuta funktiota. Tämä tehdään onClicked-funktion sisällä sen jälkeen, kun square-elementin väri on muutettu vihreäksi. Esimerkissä on hyvä muistaa, että onClicked-funktio on slot, jonka signaalin lähetys on ohjelmoijalta piilossa. Signaaleja on mahdollista ottaa vastaan myös käyttämällä QML:stä löytyvää Connections-elementtiä. Koodiesimerkissä 15 on muokattu edellistä koodiesimerkkiä niin, että lähetetty signaali otetaan vastaan square-elementin ulkopuolella toisessa elementissä.

```
Rectangle {
    width  : 500
    height : 400

    Rectangle {
        id: square

        width  : 100
        height : 50
        color  : "#0000ff"

        signal green()

        MouseArea {
            anchors.fill: parent
            onClicked:  {

                if(square.color == "#0000ff") {
                    square.color = "#008000";
                    square.green();
                }
                else
                    square.color = "#0000ff";
            }
        }
    }

    Rectangle {
        Connections {
            target  : square
            onGreen : console.log("green")
        }
    }
}
```

Koodiesimerkki 15. Signaalin vastaanottaminen Connections-elementin avulla

Koodiesimerkissä 16 on pätkä RSS-lukijan koodia, jossa käytetään Connections-elementtiä. Signaalin nappaamisen tarkoituksena on varmistaa, että feedListLoader-elementin lataamaa listaa RSS-syötteiden nimistä ei luoda ennen kuin ainakin yksi RSS-syöte on valmis näytettäväksi. Sovelluksen taustaprosessi parsii RSS-syötteen QML-käyttöliittymässä näytettävään muotoon (Taustaprosessia käydään tarkemmin läpi kappaleessa *6.1.2 Sovelluksen rakenne*). Kun ensimmäinen RSS-syöte on valmis, lähettää QML:ään asetettu control-luokka signaalin feedsReady. Connections-elementti toimii slotina tälle signaalille ja vastaanottaa sen onFeedsReady-toiminnallisuudella. Slotin sisällä asetetaan feedListLoader-elementin lähdekomponentiksi RSS-syötteiden otsikot sisältävä lista. Loader-elementti lataa listan vasta, kun sille asetetaan lähdekomponentti, joten sovellus ei luo listaa RSS-syötteiden otsikoista ennen kuin onFeedsReady-slotin toiminnallisuus suoritetaan.

```
Connections {
    target: control
    onFeedsReady: {
            feedListLoader.sourceComponent = feedListComponent;
    }
}

Loader {
    id: feedListLoader

    height : 80
    width  : parent.width
}
```

Koodiesimerkki 16. Connections-elementin käyttä RSS-lukijassa

6.4 Qt:n lisääminen QML:n taustalle

QML:ää varten on tehty oma työkalunsa, jolla QML-ohjelmia voi nopeasti ja helposti testata. Sovelluksen nimi on QML Viewer ja sen voi erikseen asentaa kehittäjän tietokoneelle. QML-koodin testaus täytyi QML kielen alkuvaiheissa suorittaa erikseen QML Viewer –sovelluksella, mutta kirjoitushetkellä esimerkiksi Nokian Qt SDK:n mukana tuleva uusin versio Qt Creator IDE:stä mahdollistaa puhtaan QML-sovelluksen käynnistämisen suoraan Qt Creatorilla. Puhdas QML-sovellus tarkoittaa sitä, että sovellus koostuu vain QML-koodista ja –tiedostoista, eikä sisällä Qt:n C++-koodia ollenkaan taustalla. Useimmiten on kuitenkin järkevää lisätä QML-sovelluksen taustalle Qt:tä, jotta sovellusen saa käännettyä binäärimuotoon ja rakennettua paketoidun asennustiedoston oikeaa kohdealustaa varten.

Kappaleessa *6.2.4 Qt-projektin luominen* esiteltiin kuinka Qt-projektipohja luodaan. Seuraava vaihe projektin teossa on luoda toiminnallisuutta main.cpp-tiedostoon, jolla QML-tiedosto saadaan näkymään sovelluksessa. Tällainen pätkä koodia löytyy valmiiksi Qt:n dokumentaatiosta ja sen voikin kopioida ja liittää suoraan Koodiesimerkki 17 mukaisesti.

```
// main.cpp

#include <QApplication>
#include <QDeclarativeView>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QDeclarativeView view;
    view.setSource(QUrl::fromLocalFile("window.qml"));
    view.show();

    return app.exec();
}
```

Koodiesimerkki 17. QDeclarativeView (Qt Reference Documentation, Qt Declarative UI Runtime)

Koodiesimerkin 17 koodi pohjautuu Qt Reference Documentation esimerkkiin sekä RSS-lukijaan. Siinä oletetaan, että projektissa luodaan QML-tiedosto nimeltä window.qml, joka näytetään ensimmäisenä ruudulla, mutta tiedosto voi tietenkin olla minkä tahansa muunkin niminen.

Seuraavaksi luodaan QML-tiedosto, johon määritellään aloitusnäytön ulkoasu. Tämäkin onnistuu tutun File-valikon ”New File or Project...” –valinnan kautta. Qt Creatorin versiossa 2.1.0 tiedosto löytyy QML valikon takaa otsikolla QML File. Valitsemalla tämän ja antamalla tiedostolle nimeksi window.qml Qt Creator luo uuden QML-tiedoston (Koodiesimerkki 18) ja lisää .pro-tiedostoon uuden QML-tiedoston OTHER_FILES-muuttujaan (Koodiesimerkki 19).

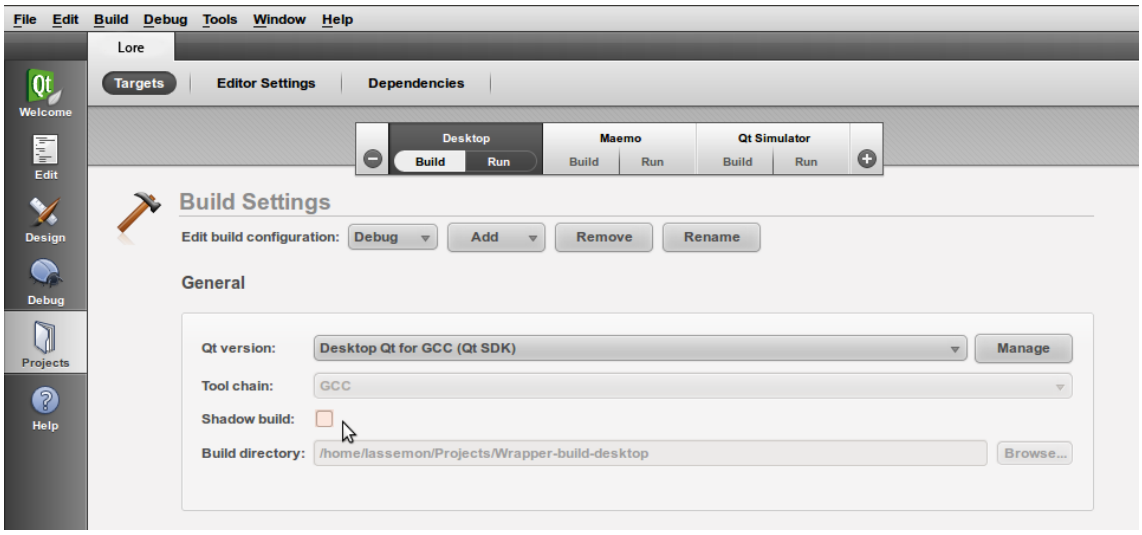
```
// window.qml
import QtQuick 1.0

Rectangle {
    width  : 640
    height : 480
}
```

Koodiesimerkki 18. application.qml

```
// .pro file
QT += core declarative

SOURCES += \
    main.cpp

OTHER_FILES += \
    window.qml
```

Koodiesimerkki 19. OTHER_FILES-muuttuja

Näiden vaiheiden jälkeen on QML-sovelluksen kääntäminen ja ajaminen viimein mahdollista. Kääntämistä käydään tarkemmin läpi kappaleessa *6.6.1 Kääntäminen oikealle alustalle*. Ennen ensimmäistä kääntöä ja ajoa on hyvä tarkistaa, että projektin Project-välilehden takana ruksitettava Shadow build –valinta ei ole ruksitettu (Kuva 9). Shadow build luo erillisen hakemiston kääntötiedostoille ja on varsin kätevä tietyissä tilanteissa, mutta tässä vaiheessa projektin luontia erilliseen hakemistoon kääntäminen johtaa käännösvirheeseen. Tämä johtuu siitä, että QML-tiedostoja ei tällä tavalla etsitä oikeasta paikasta, koska ne sijaitsevan alkuperäisessä projektihakemistossa.



Kuva 9. Shadow build valinta projektissa

```
// application.qml
import QtQuick 1.0

Rectangle {
    width  : 640
    height : 480
    color  : "red"
}
```

Koodiesimerkki 20. Punainen neliö

Lisätään vielä Qt Creatorin luomaan window.qml-tiedostoon valmiiksi luodun koodin lisäksi punainen taustavärimäärittely, jotta varmasti tiedetään, että sovellus kääntyi ja näyttää, miltä sen on tarkoitus näyttää. (Koodiesimerkki 20). Tämän jälkeen sovellus voidaan ajaa painamalla isoa virheätä Run-käskyn suorittavaa kolmiota Qt Creatorin vasemmassa alalaidassa tai vaihtoehtoisesti näppäinyhdistelmää CTRL+R. Jos kaikki on mennyt niin kuin pitäisi, ilmestyy työpöydälle 640 pikseliä leveä ja 480 pikseliä korkea punainen neliö.

Seuraavaksi luodaan projektiin Qmldir-tiedosto, jolla kartoitetaan kaikki projektissa olevat QML-tiedostot. Qmldir-tiedosto on suunniteltu ulkoisten QML-kokonaisuuksien lisäämiseen projektiin. Tiedosto mahdollistaa samalla myös näiden kokonaisuuksien eli moduulien versioinnin ja näin ollen helpottaa ohjelmistojen jatkokehitystä (Qt Reference Documentation, Modules). On hyvä huomata, että tässä mainitut QML-moduulit tarkoittavat samaa kuin aiemmin kappaleessa *6.3 QML Ohjelmointi* mainitut QML-elementit.

Qmldir-tiedoston luonti ei ole suoraan tuettuna tämän hetken uusimmankaan Qt Creatorin avulla ja koska tiedoston ei tule sisältää tiedostopäätettä ja sen tulee sijaita projektin juurihakemistossa nimellä qmldir, on se helpointa luoda manuaalisesti Qt Creatorin ulkopuolella ja sitten lisätä se Qt Creatorista projektiin (Qt Reference Documentation, Modules). Olemassa olevien tiedostojen lisääminen projektiin onnistuu esimerkiksi klikkaamalla oikealla hiiren näppäimellä projektin nimeä ja valitsemalla valikosta ”Add Existing Files..” –valinnan. Qmldir-tiedoston yleinen syntaksi on demonstroitu koodiesimerkissä 21.

```
# <Comment>
 <TypeName> [<InitialVersion>] <File>
 internal <TypeName> <File>
 plugin <Name> [<Path>]
```

Koodiesimerkki 21. Qmldir-tiedoston syntaksi (Qt Reference Documentation, Modules)

Qmldir-tiedoston lisäämisen jälkeen täytyy vielä lisätä olemassa olevat QML-moduulit tiedostoon. Tässä tapauksessa projektiin ei ole vielä luotu oikeita moduuleja, joten demonstraatiotarkoituksissa voidaan kuvitella, että projektiin lisätään aiemmin koodiesimerkissä 7 luotu Button.qml-tiedosto. Tässä esimerkissä luodun qmldir-tiedoston syntaksi näyttäisi siis koodiesimerkki 22:n mukaiselta, kun qmldir-tiedostoon lisätään ainoa projektissa oleva moduuli Button.qml. Button.qml on näin ollen merkitty projektiin omana moduulinaan eli elementtinään ja sitä voisi käyttää QML-tiedostossa Button-nimellä (Qt Reference Documentation, Modules).

```
# qmldir file
Button 1.0 Button.qml
```

Koodiesimerkki 22. Qmldir tiedosto esimerkkiprojektissa

Koodiesimerkissä 23 demonstroidaan, miten Button-moduulia käytetään koodissa sen jälkeen, kun se on lisätty projektiin ja qmldir-tiedostoon.

```
// application.qml

import QtQuick 1.0

Rectangle {
    width  : 640
    height : 480
    color  : "red"

    Button {}
}
```

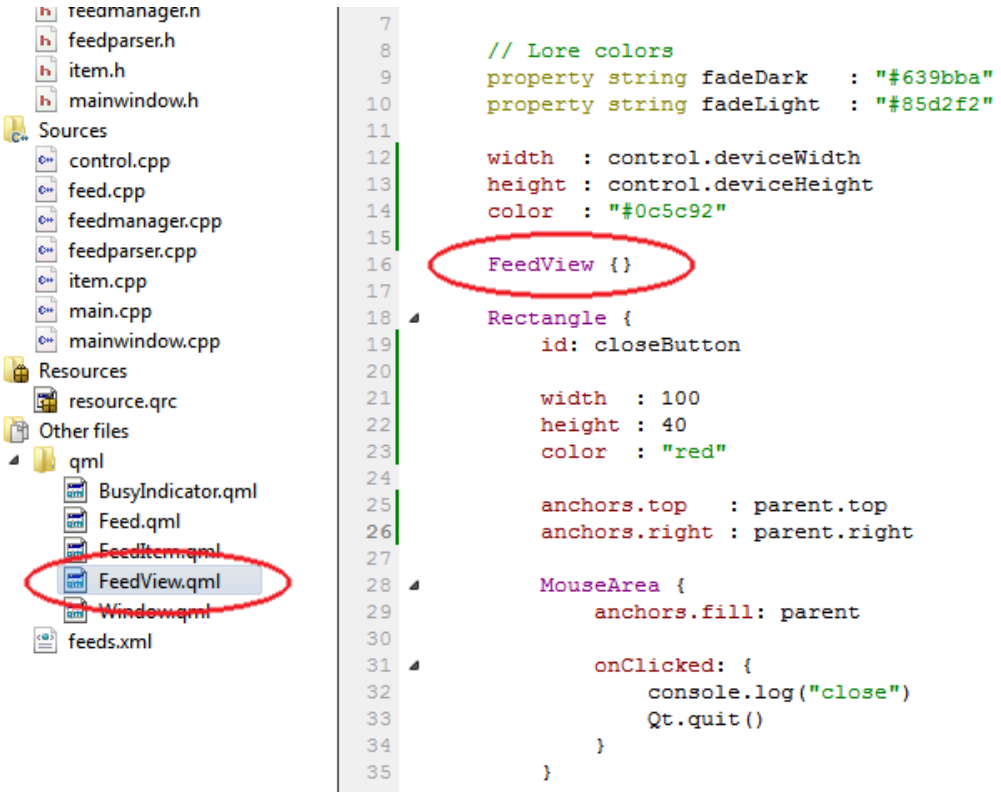
Koodiesimerkki 23. Button-moduulin käyttö

Koodiesimerkkissä 23 luotu sovellus kääntyisi ja toimisi hyvin myös ilman qmldir-tiedostoa, mutta jos Button.qml siirrettäisiin ulos samasta hakemistosta, kuin missä application.qml-tiedosto sijaitsee tai se haettaisiin internetistä URL:n perusteella se tulisi lisätä qmldir-tiedostoon, jotta sovellus kääntyisi ja Button-moduulia voisi käyttää. Tällöin myös qmldir-tiedoston moduulin määrittelevän rivin loppuosaa tulisi muuttaa niin, että se määrittelee oikean tiedostopolun tai URL-osoitteen. Koodiesimerkki 24 demonstroi, miltä qmldir-tiedosto näyttäisi, jos Button.qml-tiedosto siirrettäisiin projektihakemistosta ulos yhden hakemiston ylemmäs hakemistohierarkiassa.

```
# qmldir file
Button 1.0 ../Button.qml
```

Koodiesimerkki 24. Ulkoinen Button.qml-tiedosto qmldir-tiedostossa

RSS-syötelukijassa tällaista moduulien rekisteröintiä on käytetty koodin selkeyttämistarkoituksissa. Ensimmäisenä avattava window.qml tiedosto luo uuden toisessa tiedostossa määritellyn FeedView-elementin. FeedView-elementin tarkoitus on ottaa vastuu itse näkymän esittämisestä, kun taas window.qml-tiedosto sisältää yleisiä projektin QML-käyttöliittymään liittyviä määrityksiä. Näitä määrityksiä ovat esimerkiksi laitekohtaisen leveyden ja korkeuden asettamisen sekä tiedon siitä, mikä RSS-syöte on tällä hetkellä käyttäjän näkyvillä. Kuvassa 10 on esitelty, kuinka FeedView-elementin käyttö näkyy Qt Creator projektissa.



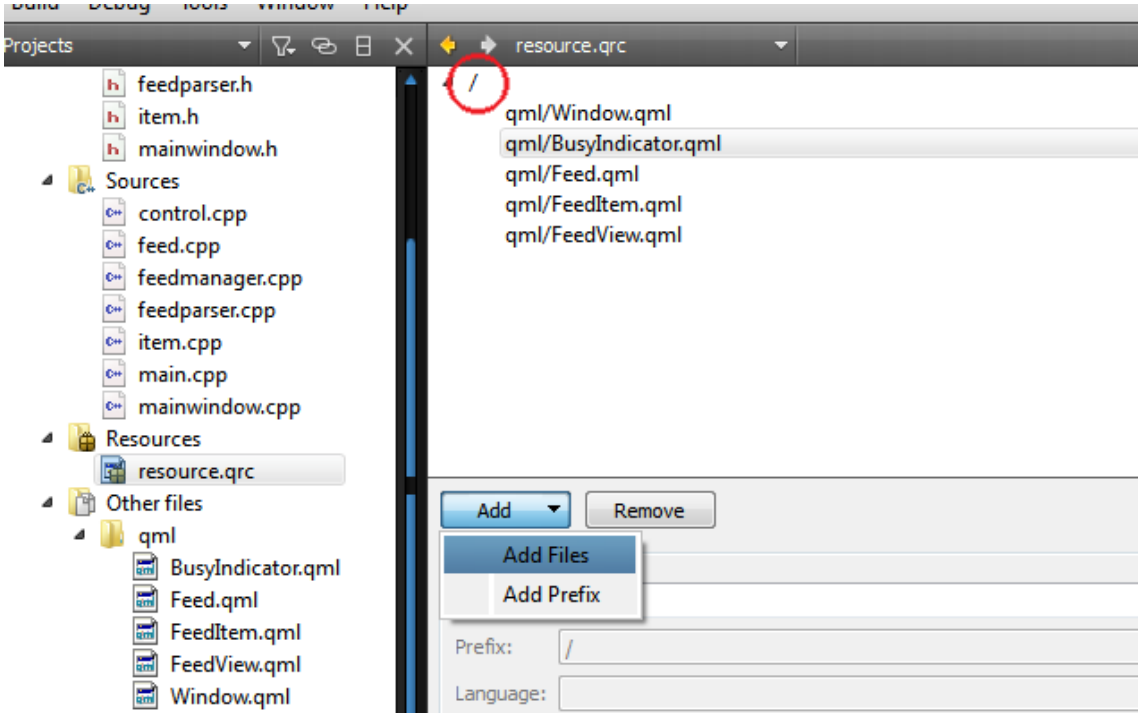
Kuva 10. FeedView-elementin käyttö RSS-lukijassa

Tämän jälkeen on hyvä lisätä projektiin resurssitiedosto. Resurssitiedoston tiedostopääte on .qrc, joka tulee sanoista Qt Resource Collection. Nimensä mukaisesti se kerää yhteen Qt-projektissa olevat resurssit, joita voivat olla esimerkiksi QML-moduulitiedostot, kuvatiedostot tai vaikkapa videotiedostot (Koodiesimerkki 25). Resurssitiedostoon merkityt resurssit tulevat osaksi lopullista alustakohtaisesti käännettyä binääritiedostoa, eikä niitä näin ollen tarvitse tarjota binääristä käynnistettävän sovelluksen lisänä erillisinä tiedostoina. Kaikki QML-tiedostot on hyvä määritellä resurssitiedostoon, sillä muussa tapauksessa tiedostot pitää tarjota erillisinä QML-tiedostoina käännetyn binäärin rinnalla, mikä monimutkaistaa ohjelmointiprosessia huomattavasti. (Qt Reference Documentation, The Qt Resource System).

```
<RCC>
    <qresource prefix="/">
        <file>qml/Window.qml</file>
        <file>qml/BusyIndicator.qml</file>
        <file>qml/Feed.qml</file>
        <file>qml/FeedItem.qml</file>
        <file>qml/FeedView.qml</file>
    </qresource>
</RCC>
```

Koodiesimerkki 25. qrc-tiedoston syntaksi RSS-lukija sovelluksessa

QML:n resurssitiedosto muistuttaa syntaksiltaan XML:ää (Koodiesimerkki 25), mutta käytettäessä Qt Creatoria ohjelmoijan ei tarvitse huolehtia qrc-tiedoston syntaksista. Resurssitiedoston voi luoda Qt Creatorin tutun ”New File or Project...” –valikon kautta valitsemalla Qt-otsikon alta ”Qt Resource file” –valinnan. Qt Creator luo uuden qrc-tiedoston, jota voi graafisesti editoida Qt Creatorin valikoiden avulla. Ensimmäisenä tiedostoon tulee lisätä etuliite eli prefix, joka määrittelee resurssitiedostojen juuren. Qt Creator tarjoaa valmiiksi prefix-vaihtoehdoksi /new/prefix1-tiedostopolkua, joka kannattaa korvata pelkällä yhdellä kauttaviivalla merkitsemään koko projektin juurihakemistoa. Tämän jälkeen resurssitiedostoon voi lisätä projektiin haluttuja resurssitiedostoja (Kuva 11).



Kuva 11. Resurssitiedoston graafinen editointi

Tämän jälkeen resursseiksi lisättyjä tiedostoja voidaan käyttää projektissa niin, että resurssit pakataan lopuksi sovelluksen mukana binääreiksi eikä niitä tarvitse tarjota erillisenä tiedostona sovelluksen mukana.

6.5 Qt:n ja QML:n välinen kommunikointi

Aiemmin on käyty läpi, miten QML-teknologialla rakennettuja näkymiä saadaan näkymään käyttäjälle asti Qt:sta käsin. On myös mahdollista viedä Qt:ssä muodostettuja tietorakenteita QML:ään käyttöliittymän hyödynnettäväksi. QML:n myötä Qt-ohjelmoijat voivat luoda Qt-luokkansa niin, että ne on mahdollista siirtää suoraan QML-koodiin käytettäväksi omina elementteinään. Tämä onnistuu Q_OBJECT- ja Q_PROPERTY-makrojen avulla. Q_OBJECT-makro käydään tarkemmin läpi kappaleessa *6.2.1 Perintähierarkia*.

Q_PROPERTY-makroa käytetään luokan ominaisuuksien määrittelyyn niin, että ne ovat ymmärrettävissä myös QML:stä käsin. Makron käyttöä varten luokan on perittävä QObject-luokka joko suoraan tai jonkin toisen luokan kautta (Qt Reference Documentation, The Property System). Q_PROPERTY-makroa hyödyntämällä on mahdollista luoda oliokokonaisuus, joka voidaan antaa QML-koodiin käytettäväksi DeclarativeContext-luokan setContextProperty-funktiota hyödyntäen. Kyseiseen funktioon pääsee käsiksi esimerkiksi kutsumalla QDeclarativeView-luokan rootContext-funktiota (Koodiesimerkki 26).

```
// FeedManager.cpp

void FeedManager::replyFeedFinished(Feed *feed)
{
    feeds.append(feed);
    m_view->rootContext()->setContextProperty( "feeds",
                                               QVariant::fromValue(feeds) );
}
```

Koodiesimerkki 26. setContextProperty-funktion käyttö

Koodiesimerkki 26 on pätkä opinnäytetyön ohella rakennettavasta RSS-lukijasta. Kyseisessä FeedManager-luokassa on feeds-luokkamuuttuja, joka on lista QObject-muuttujia. Feeds-oliolistan oliot ovat muodoltaan Feed-luokasta rakennettuja oliota. Feed-luokka puolestaan on kirjoitettu kuvamaan yhden RSS-syötteen rakennetta. Funktiosta löytyvä m_view-muuttuja on FeedManager-luokan luokkamuuttuja ja sisältää luokkaan muodostimen kautta tuodun muistiviitteen main.cpp-tiedostossa luotuun QDeclarativeView luokkaan. Koodiesimerkissä 26 kuvattua replyFeedFinished-funktiota kutsutaan aina, kun yhden Qt:n C++-koodiin parsitun RSS-syötteen työstö on valmis. Parsimisen jälkeen Feed-luokasta luotu olio lisätään feeds-listaan ja feeds-lista päivitetään lähettämällä se QML-käyttöliittymälle setContextProperty-funktiolla. SetContextProperty-funktio ottaa parametreikseen nimen, jota voidaan kutsua QML-

koodista käsin sekä itse tietorakenteen, joka tulee muokata QVariant-luokan instanssiksi. QVariant-luokka on tietynlainen perusrakenne, joka auttaa yleisimpien Qt:n tietotyyppien välisessä kommunikoinnissa (Qt Reference Documentation, QVariant Class Reference).

SetContextProperty-funktiokutsun jälkeen feeds-lista on käytettävissä QML-koodista käsin. Koodiesimerkissä 27 luodaan ListView-elementti, jonka malliksi (model) asetetaan feeds-listan sisältämät oliot. ListView-elementti luo listan rakenteita, joita toistetaan niin useasti kuin model-muuttujassa määritelty rakenne sisältää. Delegate-muuttuja kuvaa yksittäisen rakenteen visuaalista muotoa QML-kielellä. Tässä tapauksessa se on ohjelmoitu erilliseen Feed.qml-tiedostoon ja näin ollen sitä voi käyttää elementtinä delegaatin asettamisessa.

```
ListView {
    id: feedList

    anchors.fill: parent

    model    : feeds
    delegate : Feed {}
}
```

Koodiesimerkki 27. Feeds-listan käyttö QML:ssä

Opinnäytetyön kirjoittamishetkellä QML:n rakenne ei sisällä helppokäytöistä toiminnallisuutta moniulotteisten tietorakenteiden hyödyntämiseen Qt C++:sta käsin. Tämä muodostui ongelmaksi opinnäytetyön ohessa kirjoitettavaa sovellusta tehdessä, sillä yhden RSS-syötteen sisällä on listamainen rakenne useita otsikoita ja näiden sisältämiä tekstejä ja sovelluksen tulisi sisältää useita RSS-syötteitä. Tästä muodostuu moniulotteinen tietorakenne, joka tulee siirtää Qt:sta QML:ään. Ongelma on ratkaistu työssä asettamalla Feed-luokan yhdeksi luokkamuuttujaksi QObject-olioita sisältävä lista Q_PROPERTY-makroa käyttäen (Koodiesimerkki 28).

```
// feed.h

#ifndef FEED_H
#define FEED_H

#include <QObject>
#include "item.h"

class Feed  : public QObject
{
    Q_OBJECT
    Q_PROPERTY( QString header READ getHeader
                              WRITE setHeader
                              NOTIFY headerChanged )

    Q_PROPERTY( QList<QObject*> items READ getItems
                                    WRITE setItems
                                    NOTIFY itemsChanged )

    public:
        Feed(QObject *parent = 0);

        QString getHeader() const;
        void setHeader(const QString);

        QList<QObject*> getItems() const;
        void setItems(const QList<QObject*>);


    signals:
        void headerChanged();
        void itemsChanged();

    private:
        QList<QObject*> m_items;
        QString m_header;

};

#endif // FEED_H
```

Koodiesimerkki 28. Feed-luokan header-tiedosto

Koodiesimerkki 28 sisältää RSS-lukijan Feed-luokan header-tiedoston sellaisenaan. Jälkimmäinen Q_PROPERTY-makron määrittelemä luokkamuuttuja sisältää listan Item-luokan olioita. Item-luokka on kirjoitettu kuvaamaan yhden RSS-syötteessä esiintyvän otsikon ja sen mukana kulkevien tietojen rakennetta. Koodiesimerkissä 26 kuvattu feedList id:llä esitelty QML ListView-elementti luo niin monta Feed.qml tiedoston kuvaamaa rakennetta kuin feedList-elementin määrittelemä malli (model) sisältää. Toisin sanoen feedList-elementti luo sarjan RSS-syötteitä näkyville QML-käyttöliittymään. Jokainen näistä RSS-syötteistä sisältää useita otsikoita ja otsikoiden tekstejä. Näin ollen Feed-elementin rinnalle luodaan uusi ListView-elementti, joka tuo käyttöliittymään näkyville jokaisen RSS-syötteen sisältämän otsikon (Koodiesimerkki 29). Tällä kertaa listan malliksi annetaan koodiesimerkissä 28 kuvatun Feed-luokan

sisältämä luokkamuuttuja items. Tähän muuttujaan päästään käsiksi hyödyntämällä modelData-ominaisuutta (Qt Reference Documentation, QML Data Models).

```
ListView {
    id: itemList

    width  : window.width - leftPanel.width
    height : window.height

    visible: window.currentFeed == modelData.header ? true : false;

    anchors.left      : parent.right
    anchors.top       : parent.top
    anchors.topMargin : -40 * index

    model    : modelData.items
    delegate : ListDelegate{}

}
```

Koodiesimerkki 29. Feed-luokan sisältämä ListView

6.6 Sovelluksen käyttöönotto

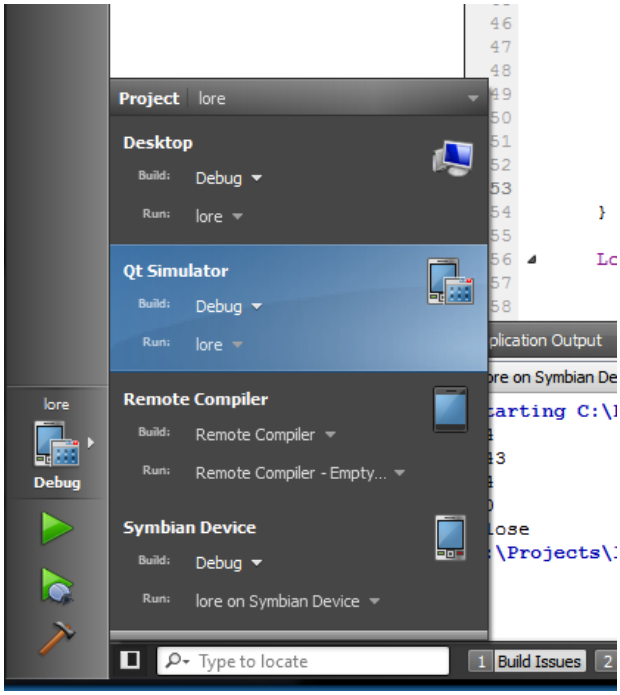
Qt koodia käännettäessä ohjelmoijalla on kolme vaihtoehtoa. Ohjelmoija voi joko käyttää Qt:n qmake-työkalua, valmista kehitysympäristöä esimeriksi asentamalla Nokia Qt SDK paketin tai kääntämällä sovelluksen kolmannen osapuolen työkalulla (Blanchette & Summerfield 2008, 593). Tässä opinnäytetyössä keskitytään kahteen ensimmäiseen vaihtoehtoon.

6.6.1 Kääntäminen oikealle alustalle

Qt:n qmake-työkalu luo projektissa olevan alustariippumattoman .pro-tiedoston pohjalta kohdealustakohtaisen Makefile-tiedoston. Qmaken luoma Makefile-tiedosto sisältää alustakohtaiset käännösohjeet, joita varsinainen kääntäjä voi hyödyntää. Tämän jälkeen sovellus voidaan kääntää käyttämällä make-sovellusta. Make-sovellus lukee Makefile-tiedoston sisällä olevat käännösohjeet ja luo lähdekoodista alustakohtaisen ajettavan binääritiedoston. Käännöksen jälkeen sovellus on valmis käynnistettäväksi (Qt Reference Documentation, qmake Manual).

Qmake-sovellusta voi käyttää joko komentoriviltä tai sitten Qt Creatorin kautta. Jälkimmäinen vaihtoehto on luonnollisesti helpompi, jos projekti on valmiiksi

ohjelmoitu Qt Creatorin avulla. Qt Creatorissa ennen ajamista on valittava oikea alusta Qt Creatorin vasemman alalaidan kohde valikosta (Kuva 12).



Kuva 12. Kohdealustan valinta

Kuvan 12 sisältämässä projektissa on valittu Qt Simulator –kääntäjä ja Debug-kääntövahtoehto. Tämä Qt Simulator –käännösvaihtoehto käynnistää simulaattorin, joka matkii aitoa mobiililaitetta. Käännös voidaan tehdä painamalla vasaran kuvaa Qt Creatorin vasemmassa alalaidassa tai vaihtoehtoisesti näppäinyhdistelmällä CTRL+B. Käännöksen jälkeen Qt Creator on luonut projektihakemistoon projektin nimellä olevan binääritiedoston. RSS-lukijan projektinimi on lore ja sen lähdekoodi sijaitsee lore-hakemistossa. Jos shadow build –valinta on projektin asetuksissa ruksitettuna, luodaan binääritiedosto oletusarvoisesti lore-build-desktop hakemistoon lore-hakemiston rinnalle. Vastaavasti luodun binääritiedoston nimi on "lore".

6.6.2 Paketin tekeminen Symbian-alustalle

Kuten aiemmin on jo todettiin Qt-koodia pystyy kääntämään ja ajamaan usealla alustalla. Tästä syntyy tilanne, jossa ohjelmoijan pitää osata myös paketoida sovellus useammalle alustalle. Tässä opinnäytetyössä olennaisin kohdealusta on Nokian

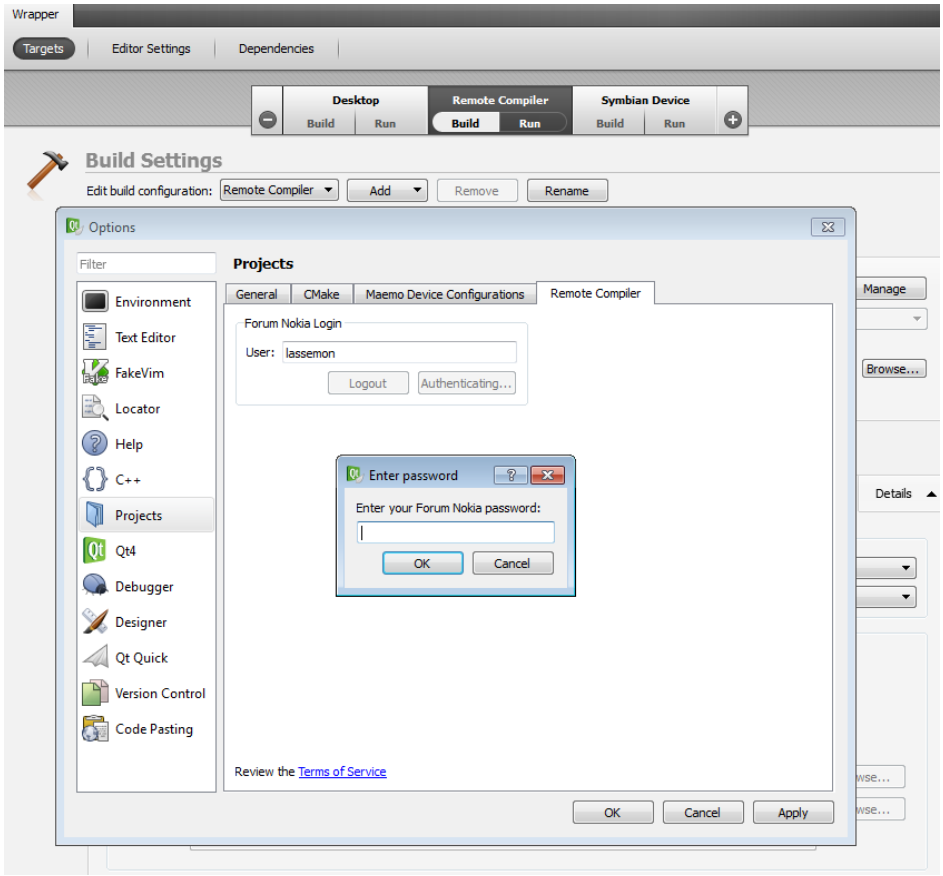
Symbian-käyttöjärjestelmä ja seuraavaksi käydään läpi paketin tekeminen tälle käyttöjärjestelmälle.

Symbian-alustalle suunnitellut sovellukset asennetaan .sis- tai .sisx-pakettien avulla. Sis-paketti on vanhempi Symbian-alustalle tarkoitettu asennuspaketti ja sisx-paketti vastaavasti uudemmille kolmannen sarjan S60 puhelimille. Tällaiset paketit saadaan luotua muun muassa Qt Creatorin Release-käännöksessä tekemän PKG-script-tiedoston avulla, vaikka se ei olekaan helpoin tapa. Tämä tiedosto pitää sisällään ohjeet siitä mitä tiedostoja laittaa mihinkin hakemistoon puhelimella. Symbian-alusta osaa hyödyntää näitä tietoja automaattisesti asennusprosessin aikana. (Nokia Corporation, Nokia Smart Installer for Symbian Introduction). Sis-paketit tulee erikseen allekirjoittaa Nokian luovuttamalla sertifikaatilla, jotta paketin voi asentaa Symbian-laitteeseen tai lähettää Ovi Storeen ladattavaksi. Myös itse allekirjoitettuja .sis-paketteja on mahdollista tehdä, mutta näin luodut sovellukset eivät voi hyödyntää tiettyjä Symbian puhelimien suojattuja ominaisuuksia. (Nokia Corporation, Deploying Applications to Symbian Devices).

Yksi helpoimmista tavoista luoda valmis .sis-paketti Qt-projektista on käyttää Qt Creatorin mukana tulevaa Remote Compiler –käännösmahdollisuutta. Nokian Remote Compiler on suunniteltu Linux- ja Apple-käyttäjiä ajatellen, sillä näillä kehitysalustoilla ei ole ollut olemassa helppoa tapaa luoda Symbianille sopivia binääritiedostoja. Remote Compilerin käyttö onnistuu kuitenkin myös Windows-kehitysympäristössä. Tätä työtä kirjoittaessa Remote Compilerin käyttö vaatii internet-yhteyden välityksellä tapahtuvan autentikoinnin Forum Nokia –tunnuksilla, jotka saa ilmaiseksi rekisteröitymällä Forum Nokian kotisivuilla. (Forum Nokia, Nokia Qt SDK Remote Compiler).

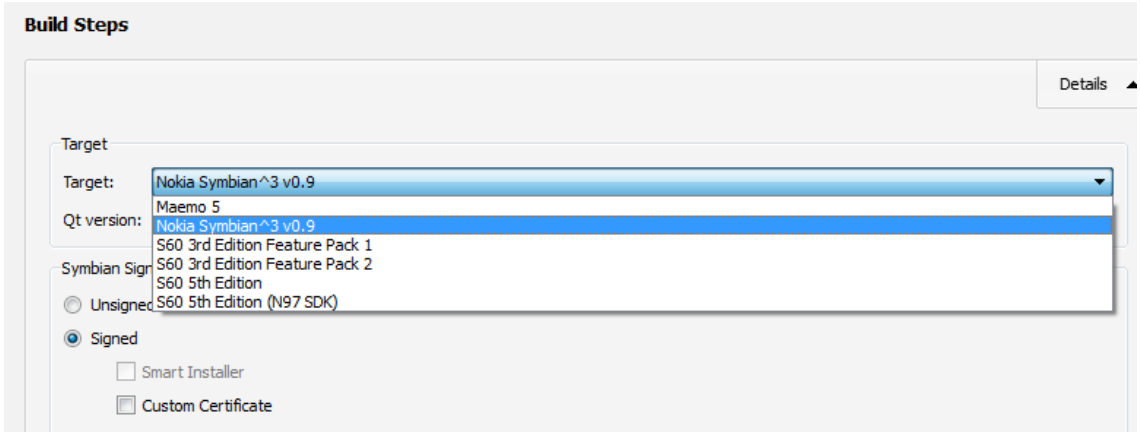
Remote Compilerin käytössä ohjelmoijan luoma lähdekoodi lähetetään internetin välityksellä Nokian palvelimelle, jossa se käännetään halutulle alustalle sopivassa ympäristössä ja siitä tehdään alustakohtainen asennuspaketti. Tämän jälkeen paketti lähetetään takaisin ohjelmoijalle ja se ilmestyy oletusarvoisest <Projektinnimi>-build-remote –hakemistoon alkuperäisen projektihakemiston rinnalle, sillä Shadow Build –valinta on oletuksena ruksitettuna Project-välilehdellä Remote Compilerin asetuksissa (Forum Nokia, Nokia Qt SDK Remote Compiler).

Jotta varsinainen Remote Compiler-käännös ja paketointi voidaan tehdä, Qt Creatorista on ensin autentikoitava itsensä Remote Compileriin (Kuva 13). (Forum Nokia, How to install and use the Nokia Qt SDK Remote Compiler).



Kuva 13. Remote Compiler –autentikointi

Autentikoinnin jälkeen Remote Compilerin asetuksista on vielä valittava mille kohdealustalle sovellus halutaan kääntää. Kuvassa 14 valitaan kohdealustaksi Symbian^3-alusta, joten Remote Compiler tuottaa ohjelmoijalle .sis-paketin.



Kuva 14. Symbian alustan valinta Remote Compilerille

Aikaisemmin Symbian-ohjelmien kehittäminen oli mahdollista vain Window-kehitysympäristössä ja tätä kirjoitettaessa Windows on edelleen kaikista helpoin ympäristö Symbian-kehitykseen, vaikkakin Linux-vaihtoehto on kehitteillä. Jos ohjelmoijalla on Symbian-laite, jolla testata ohjelmistoja, on sovelluksen käyttöönottoon rakennettu mahdollisuus asentaa ja käynnistää sovellus suoraan Qt Creatorista laitteessa. Tällöin laitteeseen yhdistetään liittämällä se tietokoneeseen esimerkiksi USB-portin kautta.

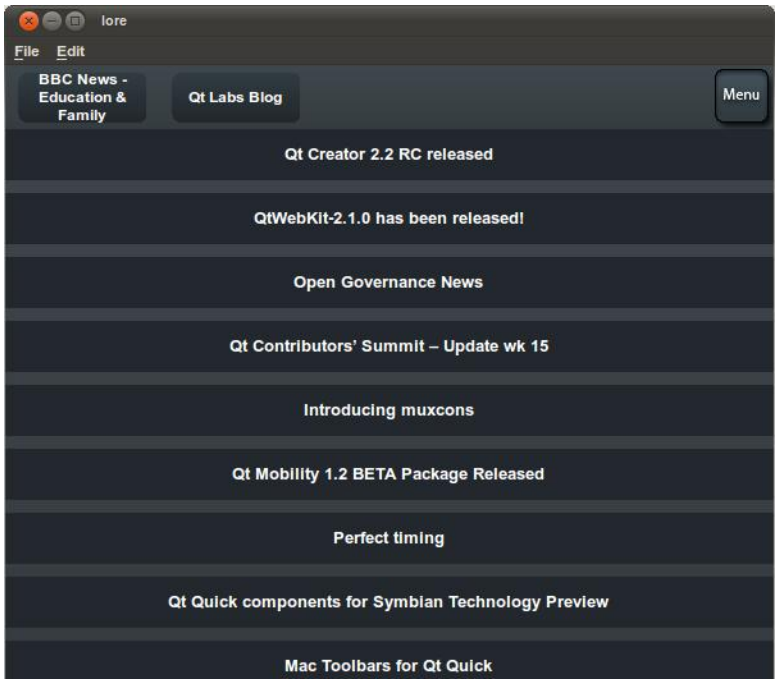
Symbian-laitteeseen tulee asentaa AppTRK-sovellus, jonka avulla Qt Creatorin pitäisi tunnistaa suoraan tietokoneessa kiinni oleva Symbian-laite, kunhan Symbian-kohdealusta on valittu Qt Creatorin vasemmassa alalaidassa olevista vaihtoehdoista (Forum Nokia, Qt Creator and on device debugging with AppTRK). Nämä vaiheet tehtyään ohjelmoija voi painaa Qt Creatorista vihreätä Run-kolmiota ja sovellus käynnistetään suoraan Symbian laitteessa ja ohjelmoija näkee välittömästi, miten sovellus toimii oikeassa laitteessa.

6.6.3 Sovelluksen vieminen Symbian-laitteeseen

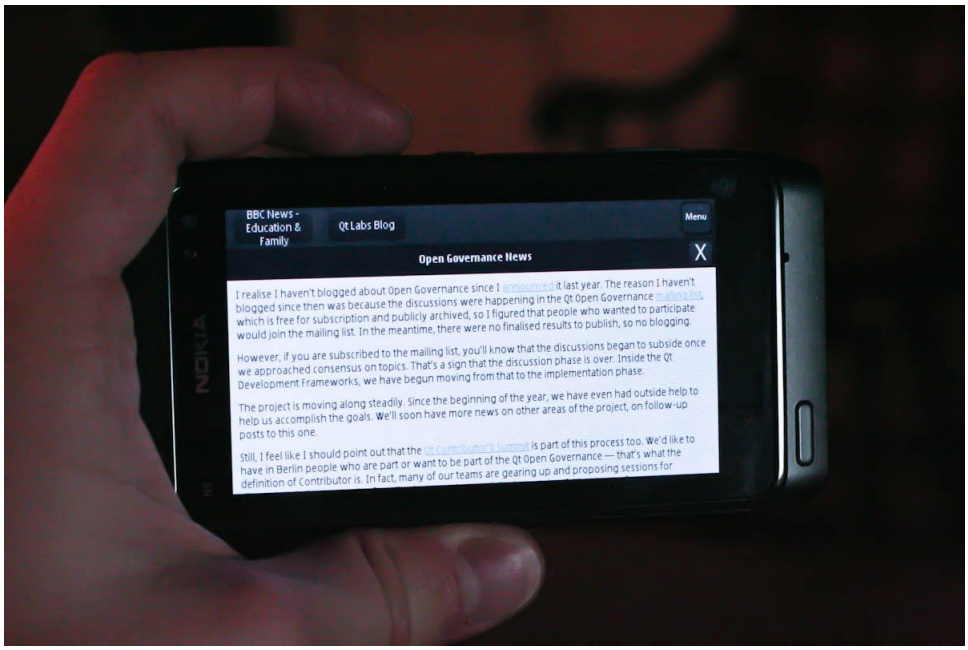
Kun ohjelmoijalla on viimein käytössään .sis- tai .sisx-paketti voidaan se siirtää Symbian laitteeseen asennettavaksi. Aikaisemmassa kappaleessa mainitun AppTRK-sovelluksen avulla kehittäjä saa nopeasti sovelluksensa Symbian-laitteeseen testausta varten, mutta sovelluksen laajempaa levitystä varten tulee siitä tehdä .sis- tai .sisx-paketti. Aikaisemmassa kappaleessa kuvattiin kuinka .sis- tai .sisx-paketti luodaan. Tämän jälkeen paketti siirretään esimerkiksi USB-kaapelin tai langattoman internetin välityksellä laitteeseen.

Windowsilla paketin siirtäminen USB-kaapelilla on erittäin yksinkertaista. Symbian-laite käyttäytyy tavallisen ulkoisen kiintolevyn tai USB-tikun tavoin ja paketin voi kopioida laitteeseen tätä kautta. Linux-ympäristöstä mainittakoon, että Nokian N8 Symbian-puhelimesta löytyvä ”Nokia Ovi Suite” USB-yhteysasetus ei mahdollistanut laitteen tunnistamista Ubuntu-linuxjakelusta käsin. Vasta kun USB-yhteysasetuksen vaihtoi ”Massamuisti”-valintaan kykeni Ubuntu tunnistamaan laitteen ja siirtämään .sis-paketin puhelimeen.

Tämän kirjallisen työn ohella ohjelmoitu RSS-lukijasovellus tuotti omat ongelmansa laitteelle asentaessa. Esimerkiksi juuri edellä mainittu ”Nokia Ovi Suite” USB-yhteysasetus sekä Symbian-paketoinnin monimutkaisuus Linux-ympäristössä. Ohessa kuva 15 on kuvankaappaus Qt Creatorin avulla Linux-ympäristössä käynnistetystä RSS-lukijasovelluksen QML-käyttöliittymästä ja kuvassa 16 esitellään miltä ohjelma näyttää Nokian N8 Symbian-puhelimessa.



Kuva 15. RSS-lukijasovellus Linux-ympäristössä



Kuva 16. RSS-lukijasovellus Nokian N8-puhelimessa.

JOHTOPÄÄTÖKSET JA POHDINTA

Ajatus opinnäytetyön aiheesta syntyi siitä, että olin harjoittelun aikana työskennellyt QML-teknologian parissa ja tuolloin aikaan sen suosio ja yritysmaailman hyödyt näyttivät olevan kasvussa. Toimeksiantajan kanssa käydyt keskustelut johtivat siihen tulokseen, että firmalla olisi tarvetta sisäiseen QML-teknologian esittelyyn, koodiesimerkkien tuottamiseen sekä valmiille sovellukselle, jota voisi käyttää QML-teknologian esittelyyn demomessuilla. Yksikin näistä toteutuessaan olisi tarpeellinen, mutta päätin aluksi ottaa opinnäytetyön tavoitteiksi kaikki kolme.

Työn aikana Qt- ja QML-teknologioiden merkitys toimeksiantajan näkökulmasta laski yllättäen, kun Nokia ilmoitti 11.2.2011 siirtyvänsä Windows Mobile –alustaan puhelimissaan. Tämä aiheutti vakavan, joskin väliaikaisen, pudotuksen motivaatiossa Qt-teknologiaa kohtaan toimeksiantajani näkökulmasta ja sen vuoksi omassa työssänikin. Qt:n avoimen lähdekoodin yhteisö ja teknologiaa muuten hyödyntävät tahot ovat Nokian ilmoituksen myötä käyneet läpi raskaitakin muutosprosesseja ja mitä todennäköisimmin käyvät vielä tämän työn valmistumisen jälkeekin. Onneksi ajan kuluessa ilmapiiri Qt:n ympärillä on muuttunut positiivisemmaksi ja Nokia on vakuutellut Qt:n ja Symbian-alustan olevan edelleen tärkeitä ja muistuttaneet MeeGonkin pysyvän kehitysalustana Nokian strategiassa.

Nokian ilmoituksen myötä lähtökohdat työn valmistumisen tärkeydelle muuttuivat omalla kohdallani anteeksiantavammiksi, sillä en enää kokenut, että työllä olisi niin suurta välitöntä työelämän kompetensseja kasvattavaa arvoa kuin työtä aloittaessa. Mielenkiintoni itse Qt-teknologiaan ei kuitenkaan hiipunut ja mitä todennäköisemmin puuhastelen sen parissa vapaa-ajalla tulevaisuudessakin. Aika näyttää, kuinka monia mahdollisuuksia työelämä tarjoaa Qt:n parissa työskentelyyn. Tällainen tilannemuutos kuitenkin vaikeutti opinnäytetyöprosessia asettamalla kyseenalaiseksi alussa asetetut tavoitteet ja tarkoituksen. Päätin kuitenkin jatkaa työskentelyä muuttamatta työn tavoitteita, sillä olin edennyt työprosessissa jo varsin pitkälle ja kiinnostukseni Qt-teknologiaa kohtaan teki aiheesta kaikesta huolimatta erittäin mielenkiintoisen.

Opinnäytetyöprosessin aikana opin mittavasti Qt:n toiminnasta ja sen varaan rakentuvasta QML-kielestä. Kykenen tuottamaan Qt-sovelluksia graafisilla QML-

käyttöliittymillä ja opiskelemaan itsenäisesti syvällisemmin Qt:n ja QML:n jo olemassa olevia toiminnallisuuksia. Tämän lisäksi olen oppinut ymmärtämään Qt-yhteistön toimintatapoja ja tavoitteita sekä seuraamaan Qt-kielen kehitystä erinäisistä internet-lähteistä. Kiinnostukseni kieltä kohtaan on kasvanut huimasti opinnäytetyöprosessin aikana. Arvioisin tämän opinnäytetyön työstämisen ajalle asettamani henkilökohtaisen kehitystavoitteen onnistuneen erinomaisesti. Tämän lisäksi olen tuottanut esittelymateriaalia tämän kirjallisen työn muodossa, joka mielestäni voi toimia oivana oppaana QML:n perusteiden opiskeluun sekä esitellä kieltä ja siihen liittyviä teknologioita myös henkilöille, joilla ei ole aikaisempaa kokemusta Qt-teknologiasta.

Työn ohessa työstetty sovellus jäi ominaisuuksiltaan alkuperäistä suunnitelmaa vajaammaksi. Työn alussa sovelluksen suunnitelluihin toiminnallisuuksiin kuului RSS-syötteiden mobiililaittesta lukemisen lisäksi mahdollisuus synkronoida luetut syötteet vastaavan työpöytäsovelluksen kanssa pilvessä toimivan palvelimen välityksellä. Työn aikana tavoite kuitenkin osoittautui liian kunnianhimoiseksi pääosin siksi, että työskentely kolme päivää viikossa toimeksiantajalle toisen teknologian parissa opinnäytetyöprosessin ohella ei luonut realistisia mahdollisuuksia niin laajan sovelluksen luomiseen. Sovelluksen perustoiminnallisuus kuitenkin valmistui opinnäytetyöprosessin aikana. Sovelluksella on mahdollista lukea RSS-syötteitä ja sen graafinen käyttöliittymä on toteutettu QML-teknologialla. Sen rakenne on suunniteltu ja toteutettu modulaarisesti, jotta myöhempi kehitys olisi mahdollisimman helppoa. Sovellus kuitenkin vaatii vielä työstöä Symbian-alustan parissa ja rinnalleen muille alustoille, kuten esimerkiksi Linux-työpöydälle, käännetyt ja suunnitellut versiot. Arvioin opinnäytetyön alussa asetetun sovellustavoitteen saavutetuksi, mutta ei erinomaisesti suoritetuksi.

Suurin osa työssä käytetystä lähdemateriaalista on internetistä, sillä työn kirjoitushetkellä QML on niin uusi teknologia, että alan kirjallisuutta siitä ei ole. Työssä käytetyt kirjalähteet, joita työssä on käytetty, ovat lähinnä Qt-teknologiaan liittyviä kirjoja tai sitten ohjelmointiprosessiin ja mobiiliohjelmointiin yleisesti liittyviä kirjoja. Internet-lähteiden sisältö tulee mitä todennäköisimmin muuttumaan Qt:n ja QML:n kehittyessä. Yleisesti ottaen koko Qt-teknologian ja yhteisön kohdalla voidaan sanoa, että muutoksia tapahtuu todella nopeasti. Tämän opinnäytetyöprosessin aikana tapahtui radikaaleja muutoksia teknologian kehityksen suunnassa ja uusia ideoita, ominaisuuksia ja korjauksia julkaistaan erittäin tiheään tahtiin. Siitä huolimatta kaikki näihin

teknologioihin liittyvät lähteet ovat Nokian ja Qt-yhteisön virallisia internet-sivustoja joten pidän niitä hyvin luotettavina.

LÄHTEET

Blanchette, J. & Summerfield, M. 2008. C++ GUI Programming with Qt 4, Second Edition. Westford Massachusetts: Prentice Hall.

Economist. 2008. Pocket World in Figures - Computer Ownership. Luettu 22.01.2011. http://www.economist.com/research.

Fahland, D. Lübke, D. Mendling, J. Reijers, H. Weber, B. Weidlich, M. & Zugal, S. 2009. Declarative versus Imperative Process Modeling Languages: The Issue of Understandability [pdf-tiedosto]. Saatavilla: http://is.tm.tue.nl/staff/hreijers/H.A.%20Reijers%20Bestanden/emmsad09.pdf.

Forum Nokia. 2010. Nokia Qt SDK Remote Compiler. Luettu 12.3.2011. http://wiki.forum.nokia.com/index.php/Nokia_Qt_SDK_Remote_Compiler.

Forum Nokia. 2011. How to install and use the Nokia Qt SDK Remote Compiler. Luettu 12.3.2011. http://wiki.forum.nokia.com/index.php/How_to_install_and_use_the_Nokia_Qt_SDK_Remote_Compiler.

Forum Nokia. 2010. Qt Creator and on device debugging with AppTRK. Luettu 12.3.2011. http://wiki.forum.nokia.com/index.php/Qt_Creator_and_on_device_debugging_with_AppTRK.

Hex-Rays SA. 2008. Debugging Symbian Applications with IDA Pro [pdf-tiedosto] Luettu 9.1.2011. Saatavilla: http://www.hex-rays.com/idapro/debugger/symbian_primer.pdf.

IEEE Xplore. A Practical Course on Mobile-Software Engineering Mobile Solutions Laboratory, [pdf-tiedosto]. Saatavilla: http://ieeexplore.ieee.org.elib.tamk.fi/stamp/stamp.jsp?tp=&arnumber=5298194.

IEEE Xplore. Arching Over the Mobile Chasm Platforms and Runtime, [pdf-tiedosto]. Saatavilla: http://ieeexplore.ieee.org.elib.tamk.fi/stamp/stamp.jsp?tp=&arnumber=5601664.

IEEE Xplore. The Mobile Web Comes Of Age [pdf-tiedosto]. Saatavilla: http://ieeexplore.ieee.org.elib.tamk.fi/stamp/stamp.jsp?tp=&arnumber=4668675.

IEEE Xplore. Visiting Mobile Application Development What, How and Where [pdf-tiedosto]. Saatavilla: http://ieeexplore.ieee.org.elib.tamk.fi/stamp/stamp.jsp?tp=&arnumber=5494784.

Laptop Magazine. 2011. Intel: “We’re Not Blinking on MeeGo”. Luettu 17.02.2011. http://blog.laptopmag.com/intel-were-not-blinking-on-meego.

Maemo. 2010. Evolution of Maemo. Luettu 22.01.2011. http://maemo.org/intro/maemo_history.

Meego Experts. 2011. 10 Qt use cases we didn’t know about. Luettu 16.3.2001.

http://www.meegoexperts.com/2011/03/10-qt-cases-we-didn%E2%80%99t.

Mikkonen, T. 2004. Mobiiliohjelmointi. Valikko-sarja
Jyväskylä: Gummerus Kirjapaino Oy.

Mikkonen, T. 2007. Programming Mobile Devices – An Introduction for Practitioners. West Sussex United Kingdom: John Wiley & Sons Ltd.

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

NationMaster. 2008. Media Statistics Mobile Cellular (per capita). Luettu 20.01.2011. http://www.nationmaster.com.

Nokia Conversations. 2010. Nokia and Intel create MeeGo for new era of mobile computing. Luettu 22.01.2011. http://conversations.nokia.com.

Nokia Corporation. 2010. Application Creation. Luettu 17.01.2011. http://qt.nokia.com/developer/learning/online/training/specialized-elearning/application-creation/application-creation-settings-resources-deployment.

Nokia Corporation. 2011. Tiedotteet. Luettu 17.02.2011. http://www.nokia.fi/nokia/lehdisto/tiedotteet/tiedotteet?newsid=1488003.

NokiaConversations Youtube Channel. 2011. Nokia CTO Rich Green talks about Symbian. Katsottu 17.02.2011. http://www.youtube.com/watch?v=_kOKyKyCeSg.

Nokia Corporation. 2011. Nokia Smart Installer for Symbian Introduction. Luettu 12.3.2011. http://doc.qt.nokia.com/smart-installer-1.1/smartinstaller-introduction.html.

Nokia Corporation. 2010. Whitepaper: Qt Quick for C++ Developers. Luettu 9.1.2011. http://qt.nokia.com/files/pdf/qt-quick-for-c-developers.

Nokia Corporation. 2010. The Nokia acquisition. Luettu 19.2.2011. http://qt.nokia.com/about/the-nokia-acquisition.

Nokia Corporation. 2011. Deploying Applications to Symbian Devices. Luettu 12.3.2011. http://doc.qt.nokia.com/qtcreator-snapshot/creator-deployment-symbian.html.

Nokia Qt Blogs. 2010. Meet Qt Quick at Mobile World Congress. Luettu 13.3.2011. http://blog.qt.nokia.com/2010/02/15/meet-qt-quick.

Nokia Qt Blogs. 2010. Qt 4.7 is here, so is Qt Developer Days. Luettu 13.3.2011. http://blog.qt.nokia.com/2010/09/21/qt-4-7-is-here-so-is-qt-developer-days.

Nokia Qt Blogs. 2011. Nokia new strategic direction. What is the future for Qt? Luettu 16.3.2011. http://blog.qt.nokia.com/2011/02/12/nokia-new-strategic-direction-what-is-the-future-for-qt.

Nokia Qt Blogs. 2011. Qt SDK 1.1 beta has been released. Luettu 10.3.2011. http://blog.qt.nokia.com/2011/03/01/qt-sdk-1-1-beta-has-been-released.

Nokia Qt Blogs. 2011. Qt and Digia, facts and fiction. Luettu 16.3.2011. http://blog.qt.nokia.com/2011/03/14/qt-and-digia-facts-and-fiction.

Nokia Qt Labs. 2011. Bringing Qt applications to Android – a quickstart video. Luettu 13.3.2001. http://labs.qt.nokia.com/2011/02/28/necessitas.

Nokia Qt Labs. 2011. QML Components for Desktop? Luettu 13.3.2011. http://labs.qt.nokia.com/2011/03/10/qml-components-for-desktop.

Nokia Qt Labs. 2011. Qt Screne Graph – Round 2. Luettu 13.3.2011. http://labs.qt.nokia.com/2010/10/08/qt-scene-graph-round-2.

Nokia Qt Labs. 2010. Lighthouse is integrated! Luettu 16.3.2011. http://labs.qt.nokia.com/2010/10/29/lighthouse-is-integrated.

Pingdom. 2010. Mobile web usage highest in Asia and Africa. Luettu 16.3.2011. http://royal.pingdom.com/2010/11/23/mobile-web-usage-highest-in-asia-and-africa.

Qt Developer Network. 2010. Qt Quick Tutorial. Luettu 9.1.2011. http://developer.qt.nokia.com/wiki/Qt_Quick_Tutorial.

Qt Reference Documentation. 2010. Best Practises Guides. Luettu 9.1.2011. http://doc.qt.nokia.com/4.7-snapshot/best-practices.html.

Qt Reference Documentation. 2010. List of QML Elements. Luettu 9.1.2011. http://doc.qt.nokia.com/latest/qdeclarativeelements.html.

Qt Reference Documentation. 2010. Qt Quick. Luettu 9.1.2011. http://doc.qt.nokia.com/4.7/qtquick.html.

Qt Reference Documentation. 2010. QML Examples and Demos. Luettu 9.1.2011. http://doc.qt.nokia.com/4.7/qdeclarativeexamples.html.

Qt Reference Documentation. 2011. Anchor-based Layout in QML. Luettu 3.3.2011 http://doc.qt.nokia.com/4.7/qml-anchor-layout.html.

Qt Reference Documentation. 2011. Writing QML Components. Luettu 3.3.2011. http://doc.qt.nokia.com/4.7/qml-extending-types.html.

Qt Reference Documentation. 2011. QML States. Luettu 3.3.2011. http://doc.qt.nokia.com/4.7/qdeclarativestates.html.

Qt Reference Documentation. 2010. Optimizing Applications for Mobile Devices. Luettu 22.01.2011. http://doc.qt.nokia.com/qtcreator-2.1-snapshot/creator-usability.html.

Qt Reference Documentation. 2011. Supported Platforms. Luettu 17.02.2011. http://doc.trolltech.com/4.7/supported-platforms.html.

Qt Reference Documentation. 2011. Platform Notes. Luettu 17.02.2011. http://doc.trolltech.com/4.7/platform-notes.html.

Qt Reference Documentation. 2011. Using the Meta-Object Compiler (moc). Luettu 20.2.2011. http://doc.qt.nokia.com/latest/moc.html.

Qt Reference Documentation. 2011. qmake Project Files. Luettu 6.3.2011 http://doc.qt.nokia.com/latest/qmake-project-files.html.

Qt Reference Documentation. 2011. Qt Declarative UI Runtime. Luettu 6.3.2011. http://doc.qt.nokia.com/latest/qmlruntime.html.

Qt Reference Documentation. 2011. Using QML in C++ Applications. Luettu 13.3.2001. http://doc.qt.nokia.com/4.7-snapshot/qtbinding.html.

Qt Reference Documentation. 2011. The Qt Resource System. Luettu 10.3.2011. http://doc.qt.nokia.com/latest/resources.html.

Qt Reference Documentation. 2011. The Meta-Object System. Luettu 16.3.2001. http://doc.qt.nokia.com/latest/metaobjects.html.

Qt Reference Documentation. 2011. QML Modules. Luettu 10.3.2011. http://doc.qt.nokia.com/4.7-snapshot/qdeclarativemodules.html.

Qt Reference Documentation. 2011. qmake Manual. Luettu 11.3.2011. http://doc.qt.nokia.com/4.7-snapshot/qmake-manual.html.

Qt Reference Documentation. 2011. QML for Qt Programmers. Luettu 13.3.2011. http://doc.qt.nokia.com/4.7/qtprogrammers.html.

Qt Reference Documentation. 2011. Inheritance Hierachy. Luettu 16.3.2011. http://doc.qt.nokia.com/latest/hierarchy.html.

Qt Reference Documentation. 2011. QVariant Class Reference. Luettu 18.3.2011. http://doc.qt.nokia.com/4.7/qvariant.html.

Qt Reference Documentation. 2011. The Property System. Luettu 18.3.2011. http://doc.qt.nokia.com/4.7/properties.html.

Qt Reference Documentation. 2011. QML Data Models. Luettu 18.3.2011. http://doc.qt.nokia.com/4.7-snapshot/qdeclarativemodels.html.

QtStudios Youtube Channel. 2011. Qt Everywhere on All Platforms. Katsottu 20.2.2011. http://www.youtube.com/watch?v=yh5Rt3cfuoQ.

Scrum Alliance. 2010. New To User Stories? Luettu 25.3.2011. http://www.scrumalliance.org/articles/169-new-to-user-stories.

Wilcox, M. 2009. Porting to the Symbian Platform – Open Mobile Development in C/C++. West Sussex United Kingdom: John Wiley & Sons Ltd.

W3Schools. 2011. CSS Tutorial. Luettu 27.2.2011. http://www.w3schools.com/css.