Tampereen yliopisto

Santeri Kangas

QTQUICK SAILFISH OS -SOVELLUSTEN KEHITYKSESSÄ

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintutkielma
Huhtikuu 2021

TIIVISTELMÄ

Santeri Kangas: QtQuick Sailfish OS -sovellusten kehityksessä
Kandidaatintutkielma
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Huhtikuu 2021

Tässä työssä tarkastellaan yleisellä tasolla Qt-ohjelmistokehykseen kuuluvaa QtQuick-moduulia sekä sen käyttöä Sailfish OS -mobiilikäyttöjärjestelmän sovellusten kehityksessä. Työssä selvitetään ja esitetään, mitkä ovat QtQuickin ja tähän kuuluvan QML-kuvauskielen (engl. Qt modeling language) yleiset ominaisuudet ja edut. Lisäksi perehdytään QtQuickin käyttöön Sailfish-ympäristössä sekä käyttöjärjestelmän omaan Silica-komponenttikirjastoon.

Työ jakaantuu kolmeen pääosaan. Ensimmäisessä osassa käsitellään QtQuick-moduulia ja siihen liittyvää QML-kuvauskieltä. Lisäksi havainnollistetaan JavaScriptin ja C++-ohjelmointikielen liittämistä mukaan QtQuick-sovelluskehitykseen. Näiden ohella esitellään lyhyesti Qt-kehyksen ja QtQuickin ohjelmisto- ja laitteistoriippumattomuuden toteuttamista. Toisessa osassa käsittely painottuu Sailfish-käyttöjärjestelmän ympäristön ja sovelluskehitystyökalujen esittelyyn. Lisäksi esitellään käyttöjärjestelmän omaa Silica-komponenttikirjastoa. Viimeisessä osassa yhdistellään QtQuick- ja Sailfish-käsittelylukujen sisältöä konkreettisen Sailfish-sovelluksen suunnittelussa ja toteutuksessa. Sovelluksen toteuttamisessa nostetaan esiin myös yleinen mobiilisovellusten arkkitehtuuri ja rajapintakyselyt.

Tutkimus osoittaa, että QtQuick kykenee tehokkaaseen sovelluskehitykseen yhdessä JavaScript ja C++-kielen kanssa. QML ja JavaScript mahdollistavat yksinkertaisten dynaamisten sovellusten luomisen. QML itsessään on joustava ja dynaaminen deklaratiivisen luonteensa vuoksi. C++ mahdollistaa laitteistoläheisen ohjelmoinnin sekä useiden matalan tason rajapintojen ja kirjastojen tuomisen mukaan sovelluskehitykseen. C++ mahdollistaa myös QML-komponenttien ja käyttöliittymän laajentamisen periyttämällä. Sailfish OS alustana ja ympäristönä osoittaa, että Qt ja QtQuick ovat kykeneviä toimimaan myös mobiilialustoilla. Sailfish-sovelluksen suunnittelu ja kehittäminen osoittivat myös omalta osaltaan, kuinka sovelluskehityksessä voidaan hyödyntää tehokkaasti JavaScriptin ja QML:n joustavuutta, deklaratiivisuutta ja skaalautuvuutta. Lisäksi ohjelmisto- ja laitteistoriippumattomuus sekä Sailfish Silica -kirjasto havainnollistavat laajennettavuutta, joustavuutta ja kustomoitavuutta.

Avainsanat: QtQuick, QML, Sailfish OS, sovelluskehitys

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

SISÄLLYSLUETTELO

1. JOHDANTO 1

2. QTQUICK 2

2.1 QML ja moduulit 2

2.2 Mallit ja näkymät 3

2.3 JavaScript ja C++ 4

2.4 Alusta- ja laitteistoriippumattomuus 6

3. SAILFISH OS 8

3.1 Sailfish SDK ja emulaattori 8

3.2 Sailfish Silica -komponenttikirjasto 10

4. SAILFISH-SOVELLUKSEN TEKEMINEN 13

4.1 Rajapintakyselyt ja mobiilisovelluksen toiminta 13

4.2 Sovelluksen näkymä ja toiminnallisuus 14

YHTEENVETO 17

LÄHTEET 18

KUVALUETTELO

Kuva 1. *Sailfish-käyttöliittymän yleisilme. 8*

Kuva 2. *Sailfish SDK:n tarjoama QtCreator IDE ja sen yleisnäkymä. 9*

Kuva 3. *SDK tarjoaa Sailfish-ajoympäristöä varten emulaattorin. 9*

Kuva 4. *Ohjelman 9 mukainen sovellus, jossa vasemmalla yleisnäkymä ja oikealla alasvetovalikko. 11*

Kuva 5. *Ohjelman 10 mukainen Cover-näkymä. 12*

Kuva 6. *Sovelluksen päänäkymä, vetovalikko sekä moniajonäkymä. 16*

LYHENTEET JA MERKINNÄT

API	engl. Application programming interface, rajapinta tiedonvaihtamista varten
HTML	engl. Hypertext markup language, verkkosivustojen kuvauskieli
IDE	engl. Integrated development environment, ohjelmointiympäristö
JSON	engl. JavaScript Object Notation, avoimen standardin tiedostomuoto tiedonvälitykseen
MVC	engl. Model-view-controller, malli-näkymä-ohjain arkkitehtuuri
QML	engl. Qt modeling language, QtQuick sovellusten ja komponenttien kuvauskieli
SDK	engl. Software development kit, sovelluskehityksen työkalut
URL	engl. Uniform Resource Locator, verkkosivun osoite
XML	engl. Extensible Markup Language, tiedonvälitykseen tarkoitettu kuvauskieli

1. JOHDANTO

Skaalautuvien ja responsiivisten sovellusten sekä verkkosivustojen kehittämiseen on herätty yhä laajemmin 2010-luvun taitteessa. Erilaisten mobiililaitteiden kirjo ja käytettävyyttä koskevat tekijät ovat pakottaneet kehittämään dynaamisia ratkaisuja myös sovellusten tekemiseen. Verkkosivustojen puolella hallitsevassa asemassa ovat edelleen HTML-dokumentteihin (engl. Hypertext markup language) liittyvät kuvauskielet ja erilaiset JavaScript-pohjaiset ohjelmistokehykset. Myös mobiilisovellusten puolella on kehitetty deklaratiivisia eli kuvaavia kieliä sekä pyritty tuomaan JavaScriptin hyödyt ja tehokkuus mukaan sovelluskehitykseen.

Tässä työssä käsitellään yleisellä tasolla Qt-ohjelmistokehyksen QtQuick-moduulia ja siihen liittyviä ominaisuuksia sekä sen soveltamista Sailfish OS -mobiilikäyttöjärjestelmän sovellusten ohjelmointiin. Tutkimuskysymyksinä ovat, mikä on QtQuick ja mitkä ovat sen edut sekä miten sitä käytetään Sailfish OS -sovelluksissa. Aihealuetta käsitellään työssä myös Sailfish-sovelluksen suunnittelun ja toteutuksen kautta.

Työn alkupuolella luvussa 2 esitellään QtQuickin ominaisuuksia yleisellä tasolla. Luvussa 3 käsitellään Sailfish-käyttöjärjestelmää yleisesti, SDK:n (engl. Software development kit) eli kehitysympäristön pystyttämistä ja Sailfish Silica -komponenttikirjastoa. Luvussa 4 käydään läpi yksinkertaisen Sailfish-sovelluksen suunnittelua ja toteuttamista käytännössä. Lisäksi sivutaan mobiilisovelluksen tyypillistä arkkitehtuuria ja rajapintapyyntöjä. Lopussa on yhteenveto QtQuickin eduista ja ominaisuuksista sekä sen käyttämisestä Sailfish-sovelluskehityksessä.

2. QTQUICK

QtQuick on Qt-ohjelmistokehykseen kuuluva avoimen lähdekoodin moduulikirjasto, joka tarjoaa työkaluja alustariippumattomien työpöytä- ja mobiilisovellusten tekemiseen. QtQuickillä on mahdollista tehdä sovelluksia myös Android- ja iOS-käyttöjärjestelmille. Sovellukset kuvataan QML-kuvauskielellä (engl. Qt modeling language). Käyttöliittymää voidaan QML:n lisäksi laajentaa käyttämällä JavaScriptiä tai C++-kieltä. QtQuick tarjoaa myös kattavan määrän erilaisia visuaalisia animaatio- ja siirtymisefektejä sekä interaktiivisia dialogeja, mutta näihin ominaisuuksiin ei perehdytä tässä työssä. [1][2]

2.1 QML ja moduulit

QML on samaan aikaan kuvauskieli, mutta myös tämän ajamista varten tarkoitettu moottori (engl. QML engine). Käyttöliittymäkomponentit kuvataan QML-komponentteina, joista käytetään myös nimitystä instanssi. [1] Moduulilla tarkoitetaan tyypillisesti QML ympäristöön tuotua kirjastoa, kuten QtQuick itsessään, mutta myös yksittäisistä QML-tiedostoista saatetaan käyttää nimitystä moduuli. QML muistuttaa jonkin verran verkkosivustojen HTML-kieltä. Lisäksi QML:n syntaksi on tyypillisesti hierarkkinen, jolloin komponenttien välille muodostuu vanhempi-lapsi-suhteita (engl. Parent-child hierarchy). [3]

```
1    #include <QApplication>
2    #include <QQmlApplicationEngine>
3
4    int main(int argc, char *argv[])
5    {
6        QApplication app(argc, argv);
7
8        QQmlApplicationEngine engine;
9        const QUrl url(QStringLiteral("qrc:/main.qml"));
10       QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
11                        &app, [url](QObject *obj, const QUrl &objUrl) {
12           if (!obj && url == objUrl)
13               QCoreApplication::exit(-1);
14       }, Qt::QueuedConnection);
15       engine.load(url);
16
17       return app.exec();
18   }
```

Ohjelma 1. *QtQuick-sovelluksen pääohjelma.*

```
1     import QtQuick 2.2
2     import QtQuick.Controls 2.15
3
4     ApplicationWindow {
5        id: mainWindow
6        height: 500
7        width: 500
8        property bool pushButtonClicked: false
9
10       Button {
11             id: pushButton
12             text: qsTr(”Push Button”)
13             onClicked: mainWindow.pushButtonClicked = true
14       }
15    }
```

Ohjelma 2. *Yksinkertainen QML-sovellus, jossa pääikkuna sekä painike, joka muuttaa tämän tilaa. QML komponentit kuvataan deklaratiiviseen tyyliin.*

Ohjelmassa 1 on esitetty QtQuick-sovelluksen pääohjelmakehys ja ohjelmassa 2 tämän pääikkunan kuvaus. Pääohjelmassa luodaan sovelluksen suorittamisen kannalta välttämättömät komponentit, kuten esimerkiksi QML-moottori. Vaadittavat moduulikirjastot tuodaan QML-ympäristöön import-määreellä. Komponentit määritellään nimellä sekä aaltosulkeilla. Lisäksi komponenteille on mahdollista antaa tunniste ja muita ominaisuustietoja (engl. Property), jotka muistuttavat JavaScriptin objekti−arvo-pareja. Komponenteilla on myös niille ominaisia signaaleja, kuten ohjelmassa 2 painikkeen onClicked. Komponentit rekisteröityvät QML-moottoriin, mikä mahdollistaa tunnistetiedolla viittaamisen myös kokonaan toisesta komponentista tai instanssista. [3]

2.2 Mallit ja näkymät

QtQuick-sovelluksissa on hyvin keskeistä mallien (engl. models) ja näkymien (engl. views) hyödyntäminen. QtQuick tarjoaa mallien toteuttamista varten esimerkiksi ListModel-komponentin, johon voidaan lisätä sisältöä ListElement-komponenteilla. Lisäksi näkymiä varten tarjotaan List-, Grid- ja PathView. Malli-näkymä-periaatteella yhdessä C++-kielen kanssa voidaan QtQuick-sovelluksissa toteuttaa myös MVC (engl. Model-view-controller) eli malli-näkymä-ohjain-arkkitehtuuri. [4]

```
1     ListModel {
2        id: textModel
3        ListElement {name: "Model 1"}
4        ListElement {name: "Model 2"}
5        ListElement {name: "Model 3"}
6     }
7
8     ListView {
9        anchors.fill: parent
10       model: textModel
11       delegate: Component {
12             Text {
13                   text: name
14             }
15       }
16    }
```

Ohjelma 3. *Yksinkertainen malli-näkymä esimerkki QtQuick-sovelluksessa.*

Ohjelmassa 3 on esitetty yksinkertainen malli-näkymä esimerkki. ListModel määrittää mallin, jota hyödynnetään ListView-komponentissa. Delegatella määritetään, miten mallin tarjoama data halutaan näyttää näkymässä. [4] Ohjelmassa 3 ListElement pitää sisällään vain kentän name, mutta todellisuudessa komponentit voivat pitää sisällään useita kenttiä ja tietoja.

2.3 JavaScript ja C++

Yksi keskeisimmistä ominaisuuksista ja eduista on JavaScriptin ja C++-kielen liittäminen mukaan sovellukseen. JavaScript voidaan yhdistää kätevästi QML:n kanssa, kun taas C++ mahdollistaa esimerkiksi laitteisto- ja järjestelmätasolla erilaisten rajapintojen käyttämisen. [3]

```
1     import QtQuick 2.2
2     import QtQuick.Controls 2.15
3     import “logic.js” as Logic // tuodaan logiikkakirjasto
4
5     Button {
6        id: pushButton
7        text: qsTr("Change status")
8
9        function changeStatus() {
10             mainWindow.pushButtonClicked = true
11             Logic.changeTheme()
12       }
13       onClicked: changeStatus()
14    }
```

Ohjelma 4. *Ohjelmaan voidaan lisätä suoraan JavaScript-funktioita tai ne voidaan tuoda resursseina käyttäen import-määrettä.*

Ohjelmassa 4 on esitetty kaksi tapaa lisätä JavaScriptiä QML:n sekaan. Yleensä kätevin tapa pienille ja lyhyille toiminnoille on lisätä QML-moduulille suoraan omia funktioita. JavaScript logiikkaa voidaan lisätä myös tuomalla ne niin sanottuina QML-resursseina, jolloin lisäämiseen käytetään import-määrettä ja QML:n puolelta viitataan as-määreen jälkeiseen nimeen. [5]

C++-kieltä hyödynnetään tyypillisesti QtQuick-sovelluksen niin kutsutun backendin luomiseen sekä sovelluksen tehokkuuden lisäämiseen resurssi- ja laskentaintensiivisissä tapauksissa. Lisäksi C++ mahdollistaa Qt-kehyksen tarjoamien moduulien laajentamisen periyttämällä. [6]

```
1    int main(int argc, char *argv[])
2    {
3        QApplication app(argc, argv);
4        qmlRegisterType<BackEnd>(“AppLogic”, 1, 0, “Backend”);
5
6       ...
7    }
```

Ohjelma 5. *C++-luokka voidaan lisätä QML-moottoriin pääohjelmassa käyttäen qmlRegisterType-metodia.*

Ohjelmassa 5 on esitetty, miten C++ backend -luokan lisääminen tapahtuu pääohjelmassa käyttämällä qmlRegisterType-metodia. Metodi ottaa parametreinaan nimiavaruuden, version ja QML-instanssin nimen. Nimiavaruutta käytetään QML:n puolella moduulin tuomiseen ja instanssin nimeä komponentin deklaratiiviseen määritykseen. [6]

```
1    import AppLogic 1.0
2
3    Backend {
4       id: logic
5    }
6
7    Label {
8       id: header
9       text: logic.getText()
10   }
```

Ohjelma 6. *QML-moottoriin rekisteröity C++-luokka voidaan lisätä import-määreellä ja siitä voidaan luoda tuttuun tapaan QML-instanssi aaltosulkeilla.*

Ohjelmassa 6 on esitetty lyhyesti, miten C++-luokka voidaan esimerkin mukaisesti lisätä QML:n puolelle. C++-luokkia voidaan liittää mukaan myös yleisinä nimireferensseinä, jolloin QML-komponentit voivat kutsua luokan metodeja suoraan tunnistenimellä. Tällöin C++-luokasta ei myöskään tarvitse luoda erillistä QML-komponenttia. C++-luokkien liittämiseen ei kuitenkaan perehdytä tämän enempää tässä työssä.

2.4 Alusta- ja laitteistoriippumattomuus

Luvun 2 johdannossa mainittiin jo QtQuickin alustariippumattomuudesta sekä tuesta esimerkiksi Android- ja iOS-käyttöjärjestelmien sovelluskehitykseen. Lähtökohtaisesti Qt-kehyksen päälle rakennetut alustariippumattomat sovellukset hyödyntävät esimerkiksi samoja C++-luokkia, mutta toteuttavat käyttöliittymän toimintaa yksilöllisesti. QtQuickissä on useampia tapoja toteuttaa alustariippumattomuutta käyttäen ohjelmassa esimerkiksi direktiivejä tai Qt-kehyksen sisältämiä ympäristömuuttujia. [7, s. 147, 150]

```
1    void Backend::install() {
2       #if defined(Q_OS_IOS)
3             // iOS toteutus
4       #endif
5
6       #if defined(Q_OS_ANDROID)
7             // Android toteutus
8       #endif
9    }
```

Ohjelma 7. *Alustariippumattomuuden toteuttamista C++-luokassa käyttäen ehtodirektiivejä.*

```
1    Button {
2       id: pushButton
3       text: qsTr(“Push Button”)
4       onClicked: {
5             if (Qt.platform.os === "ios") {
6                   // iOS toteutus
7             } else if (Qt.platform.os === "android") {
8                   // Android toteutus
9             }
10       }
11   }
```

Ohjelma 8. *Alustariippumattomuuden toteuttamista QML:n puolella käyttäen Qt-kehyksen tarjoamia ympäristömuuttujia.*

Ohjelmassa 7 on esitetty, miten kuvitteellisessa C++-luokassa voidaan toteuttaa alustariippumattomuutta käyttämällä ehtodirektiivejä. Ohjelmassa määritellään koodinrivit, jotka ovat Android- tai iOS-käyttöjärjestelmäspesifisiä. Ohjelma 8 puolestaan esittää tilanteen, jossa käyttöliittymän puolella voidaan määrittää laitteistospesifinen toteutus painikkeen klikkaamiselle.

Chmielewski et. al [8, s. 186] kuvaa artikkelissaan laitteistoriippumattoman kehitysympäristön ja arkkitehtuurin edellyttävän, että

- sovelluksen suorittaminen ei tapahdu varsinaisesti laitteessa itsessään eli sovelluksen toimintalogiikka ei riipu laitteesta,
- laitteen sensoridataa luetaan vain standardoitujen rajapintojen kautta ja
- käyttöliittymän kuvaukseen käytetään universaalia kuvauskieltä.

QtQuick yhdessä Qt-kehyksen kanssa toteuttaa kaikki nämä kolme vaatimusta. Qt nimittäin tarjoaa kehitysympäristönä suuren määrän standardoituja moduuleja ja komponentteja, jotka eivät riipu laitteesta itsessään. Sovelluksen toimintalogiikka voidaan siis toteuttaa käyttäen näitä kirjastoja alusta- ja laitteistoriippumattomasti. Lisäksi QtQuick-moduuli yhdessä QML:n kanssa tarjoaa universaalin käyttöliittymän kuvauskielen Qt-ympäristön sisällä.

3. SAILFISH OS

Sailfish on avoimen lähdekoodin Linux-pohjainen mobiilikäyttöjärjestelmä, joka perustuu osittain Nokian ja Intelin kehittämään MeeGo-käyttöjärjestelmään. Sailfish julkaistiin marraskuussa 2013 beta-versiona yhdessä Jolla-puhelimen kanssa. Käyttöjärjestelmää kehittää suomalainen teknologiayritys Jolla Oy. Sailfish on valittu tähän työhön, koska se on tällä hetkellä ainoa aktiivisessa kehityksessä oleva käyttöjärjestelmä, jolla voidaan ajaa natiivina QtQuick- ja QML-pohjaisia sovelluksia. [9, s. 1–2][10]

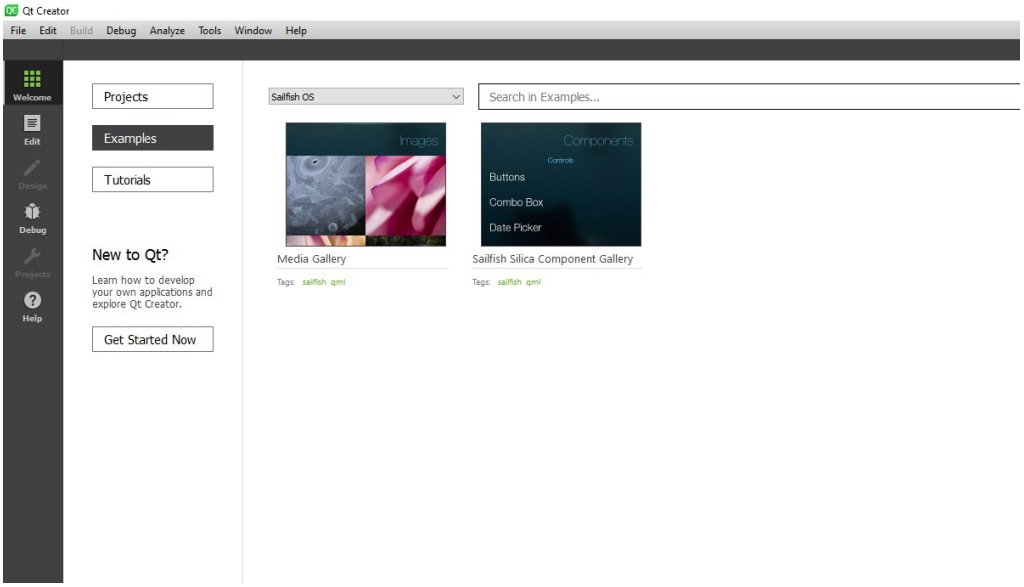
Kuva 1. *Sailfish-käyttöliittymän yleisilme.*

Kuvassa 1 on esitetty Sailfishin käyttöliittymän yleisilmettä. Vasemmalta oikealle ensimmäisessä kuvassa on esitetty lukitusnäyttö, toisessa moniajonäkymä ja kolmannessa alasvetovalikko (engl. Pulley menu). Sailfish-käyttöliittymässä on panostettu liikkeisiin ja eleisiin perustuvaan ohjaukseen, mistä hyvänä esimerkkinä on mainittu alasvetovalikko erilaisten nappien tai dialogien sijaan.

3.1 Sailfish SDK ja emulaattori

Sailfish SDK sisältää QtCreatorin päälle rakennetun ohjelmointiympäristön eli IDE:n (engl. Integrated development environment), Sailfish-sovelluksia varten oman kääntäjän sekä emulaattorin ja rajapintadokumentaation. Kääntäjä ja emulaattori tarjotaan virtualisoinnin kautta, joten niiden ajamista varten tarvitaan jokin virtualisointiteknologia, kuten

Docker tai VirtualBox. QtCreator IDE mahdollistaa myös useimpien Qt 5 -kehykseen kuuluvien komponenttien ja moduulien käyttämisen sovelluskehityksessä. [11]



Kuva 2. *Sailfish SDK:n tarjoama QtCreator IDE ja sen yleisnäkymä.*

Kuva 3. *SDK tarjoaa Sailfish-ajoympäristöä varten emulaattorin.*

Kuvassa 2 on esitetty Sailfish SDK:n tarjoama QtCreator IDE ja sen yleisnäkymä. IDE:ssä on tarjolla esimerkkejä Sailfishin omista komponenteista. Kuvassa 3 on esitetty emulaattorin yleisnäkymä VirtualBoxilla ajettuna.

3.2 Sailfish Silica -komponenttikirjasto

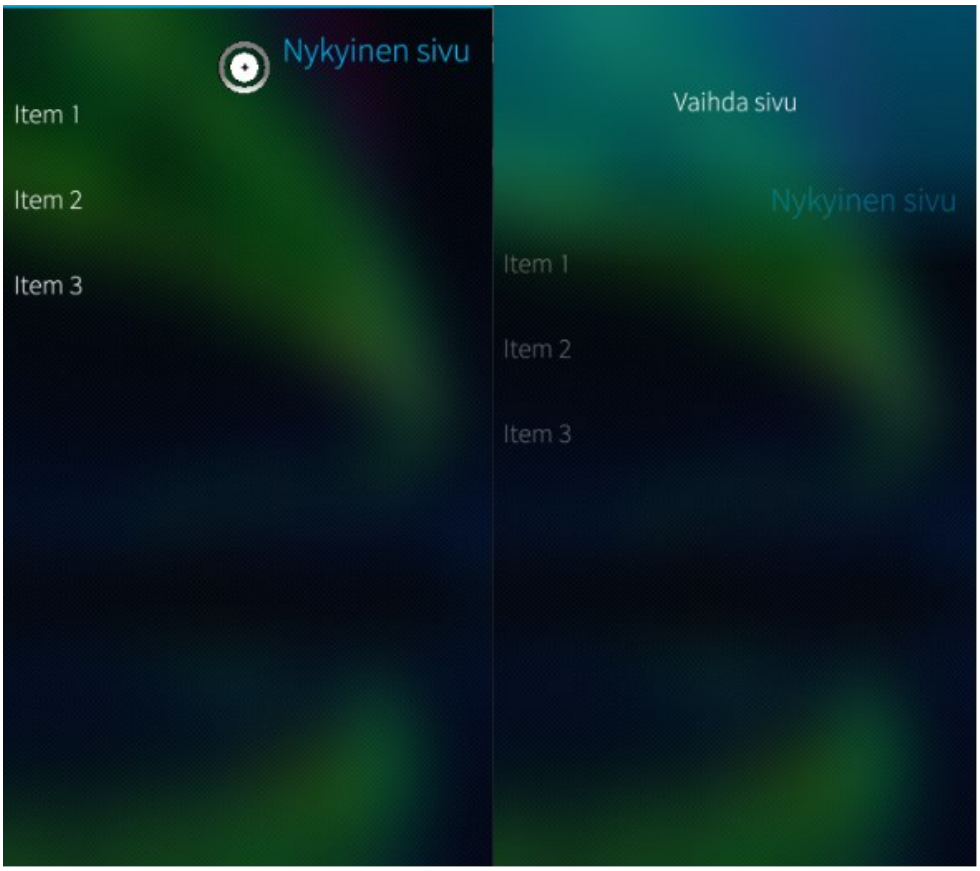
Silica on Sailfishin natiivikomponenttikirjasto, joka on rakennettu QtQuickin tapaan moduulikirjastona pohjautuen Qt-ohjelmistokehykseen. Silica sisältää suuren määrän Sailfishiin tyyliteltyjä komponentteja ja kontrolleja. Lisäksi Silica tarjoaa mahdollisuuden käyttää Sailfish-tyyliteltyjä ikoneja ja muita kuvakkeita. [12] Sailfish-käyttöliittymän ja sovelluksen kannalta merkittävimpiä Silica-komponentteja ovat esimerkiksi Page, PageStack, SilicaListView, PullDownMenu ja BackgroundItem.

```
1   import QtQuick 2.0
2   import Sailfish.Silica 1.0
3
4   Page {
5       ListModel {
6           id: textModel
7           ListElement {name: "Item 1"}
8           ListElement {name: "Item 2"}
9           ListElement {name: "Item 3"}
10      }
11
12      SilicaListView {
13          anchors.fill: parent
14          contentHeight: parent.height
15          model: textModel
16          PullDownMenu {
17              MenuItem {
18                  text: qsTr("Vaihda sivu")
19                  onClicked: pageStack.push("OtherPage.qml")
20              }
21          }
22          header: PageHeader {
23              title: qsTr("Nykyinen sivu")
24          }
25          delegate: BackgroundItem {
26              height: Theme.itemSizeMedium
27              Column {
28                  anchors.left: parent.left
29                  anchors.leftMargin: Theme.paddingMedium
30                  Label {
31                      text: name
32                  }
33              }
34          }
35      }
36  }
```

Ohjelma 9. *Merkittävimmät Silica-komponentit esitettynä ohjelmana.*

Ohjelmassa 9 on esitetty aiemmin mainitut merkittävimmät Silica-komponentit ohjelmassa. Page- ja PageStack-komponentit liittyvät Sailfish-sovelluksen sivujen ja sivunäkymien hallintaan. PageStackin avulla voidaan esimerkiksi liikkua eri sivujen välillä.

SilicaListView on QtQuickin ListView-komponentin päälle rakennettu tyylitelty näkymä. PullDownMenu-komponentilla luodaan alasvetovalikko, johon voidaan liittää toimintoja, kuten esimerkiksi toiselle sivulle siirtyminen. BackgroundItem on interaktiivinen ja tyylitelty komponentti, jota voidaan esimerkiksi klikata ja määrittää tälle toimintoja. [13]



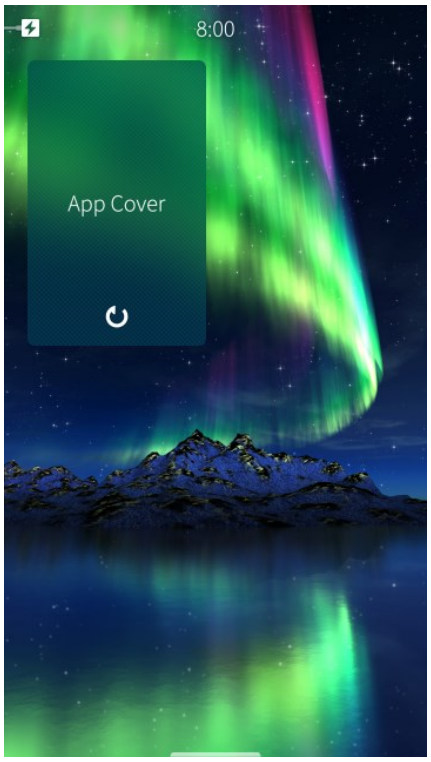
Kuva 4. *Ohjelman 9 mukainen sovellus, jossa vasemmalla yleisnäkymä ja oikealla alasvetovalikko.*

Kuvassa 4 on esitetty, miltä ohjelman 9 mukainen koodi näyttää Sailfish-sovelluksessa. Vasemmalla puolella on esitetty ohjelman 9 yleisnäkymä sekä oikealla puolella alasvetovalikko, josta voidaan vaihtaa sivua. Alasvetovalikon lisäksi Silica tarjoaa PushUpMenu-komponentin, jonka toiminta on käänteinen PullDownMenuun nähden. Huomattavaa on, että ohjelma 9 hyödyntää QtQuickin mallien ja näkymien vuorovaikutusta, jota käsiteltiin luvussa 2.2.

Sailfish-sovelluksille on myös tyypillistä tarjota pikatoimintoja moniajonäkymässä. Tätä varten Silicassa tarjotaan Cover- ja siihen liittyviä muita toiminnallisuuskomponentteja. Cover-toiminnallisuudet mahdollistavat sovelluksen interaktiivisuuden myös silloin, kun se ei ole aktiivisesti auki.

```
1     CoverBackground {
2        Label {
3              anchors.centerIn: parent
4              text: qsTr("App Cover")
5        }
6
7        CoverActionList {
8              CoverAction {
9                    id: refresh
10                   iconSource: "image://theme/icon-cover-refresh"
11                   onTriggered: sailfishapp.refresh()
12             }
13       }
14    }
```

Ohjelma 10. *Cover-komponentti tarjoaa sovellukselle pikatoimintoja moniajonäkymässä ja mahdollistaa interaktiivisuuden myös tausta-ajossa.*



Kuva 5. *Ohjelman 10 mukainen Cover-näkymä.*

Ohjelmassa 10 on esitetty yksinkertainen Cover-komponenttiesimerkki ja kuvassa 5 tämän visuaalinen näkymä. CoverAction-komponentit reagoivat esimerkiksi painallukseen ja onTriggered signaalilla voidaan määrittää haluttu toiminto. Ohjelmassa 10 CoverActioniksi on määritelty kuvitteellinen sovelluksen virkistäminen.

4. SAILFISH-SOVELLUKSEN TEKEMINEN

Verkkopalvelu finna.fi tarjoaa hakupalvelun, johon on koottu kotimaisten toimijoiden aineistoja kuten valokuvia, kirjallisuutta, lehtiä ja taidetta. Finna tarjoaa myös kattavan avoimen rajapinnan eli API:n (engl. Application programming interface), jonka tarkempi toiminnallisuus ja käyttäminen on kuvattu osoitteessa api.finna.fi. Tässä luvussa suunnitellaan ja toteutetaan yksinkertainen QML-sovellus, jolla voidaan hakea finnan rajapinnasta halutulla kyselyllä tietoa ja näytetään kooste tuloksista sovelluksessa. Lähdekoodi on saatavilla osoitteessa https://github.com/Moppa5/harbour-finna-viewer.

4.1 Rajapintakyselyt ja mobiilisovelluksen toiminta

Tyypillisesti mobiilisovellukset tehdään siten, että ne lähettävät pyyntöjä johonkin rajapintaan ja saavat vastauksena takaisin dataa. Rajapinnasta saatu data on yleensä tarjottu, joko JSON (engl. JavaScript Object Notation) tai XML (engl. Extensible Markup Language) muodossa, jonka käsittelemiseen (engl. Parsing) löytyy useimmista kielistä standardikirjasto. Mobiilisovellus voi JSON tai XML datan käsittelyn jälkeen tallentaa sen omaan paikalliseen ympäristöön sopivaksi myöhempää käyttöä varten. Esimerkiksi QtQuickissä data voidaan tallentaa ListModel-komponenttiin. QtQuickissä yksinkertaiset rajapintakyselyt voidaan suorittaa QML:n seassa käyttäen JavaScriptin XMLHttpRequest-luokkaa. JavaScriptissä on myös integroituna metodi JSON-muotoisen datan käsittelyyn. [14, s. 115–117]

```
1    function request(queryUrl) {
2       var xhr = new XMLHttpRequest()
3       xhr.open("GET", queryUrl)
4
5       xhr.onreadystatechange = function() {
6             if (xhr.readyState === XMLHttpRequest.DONE) {
7                   var results = JSON.parse(xhr.responseText)
8                   // halutun datan kerääminen tuloksista
9                   ...
10            }
11      }
12      xhr.send()
13   }
```

Ohjelma 11. *Yksinkertainen JavaScript request-funktio rajapintakyselyn suorittamista varten.*

Ohjelmassa 11 on esitetty yksinkertainen funktiorunko, jolla voidaan suorittaa rajapintakysely haluttuun verkko-osoitteeseen eli URL:iin (engl. Uniform Resource Locator). Sovelluksen yksinkertaisuuden vuoksi kerätään tuloksista lähinnä otsikko, teoksen tyyppi ja mahdollinen kuva.

```
1    for (var index = 0; index < results.records.length; index++) {
2        var item = results.records[index]
3        var image = "";
4
5        if (item.images[0])
6            image = item.images[0]
7
8        // Lisätään kerätyt tiedot ListModeliin
9        queryResults.append({"title": item.title,
10                            "imgurl": image,
11                            "type": item.formats[0].translated})
12   }
```

Ohjelma 12. *Tulokset voidaan käydä läpi for-silmukassa ja tallentaa halutut tiedot objekti-arvo-pareina.*

Ohjelmassa 12 on esitetty, miten sovelluksen kannalta olennainen data voidaan kerätä ListModel-komponenttiin. ListModel hyödyntää JavaScriptin objekti-arvo-pareja datan assosioinnissa. Kenttiä title, imgurl ja type voidaan hyödyntää seuraavaksi sovelluksen näkymässä.

4.2 Sovelluksen näkymä ja toiminnallisuus

Sailfish-sovelluksissa käytetään hyvin tyypillisesti näkymänä SilicaListView-komponenttia [13], joka tarjoaa samalla myös kätevän tavan integroida mukaan vetovalikkoja. PullDownMenu-valikkoon voidaan sijoittaa esimerkiksi sovelluksen haun tyhjentäminen tai palauttaminen vakioasetuksille. PushUpMenu-valikkoja sen sijaan käytetään yleensä lisätulosten hakemiseen, mikäli kyselyn tuottamien tulosten määrää on rajoitettu sovelluksen näkymässä.

```
1    TextField {
2        id: queryfield
3        description: qsTr("Hae nimellä tai muulla nimikkeellä")
4        anchors.top: headertitle.bottom
5        placeholderText: qsTr("Hae...")
6        EnterKey.iconSource: "image://theme/icon-m-enter"
7        EnterKey.onClicked: // hae hakutuloksella
8    }
```

Ohjelma 13. *Tekstikenttä käyttäjän antamaa syötettä varten.*

Ohjelmassa 13 on esitetty yksinkertainen TextField-komponentti [13] käyttäjän antamaa syötettä varten. Komponentilla voidaan kerätä käyttäjän antama hakulauseke ja kytkeä se esimerkin mukaisesti Enter-painikkeen signaaliin.

```
1     model: queryResults
2     delegate: BackgroundItem {
3        height: Theme.paddingLarge + queryrow.height
4             Row {
5                 id: queryrow
6                 spacing: Theme.paddingMedium
7
8                 Image {
9                     width: 100
10                    height: 130
11                    source: (imgurl) ? "https://api.finna.fi" +
12                         imgurl : "image://theme/icon-m-attach"
13                }
14                Column {
15                    Label {
16                        text: title
17                    }
18
19                    Label {
20                        text: type
21                    }
22                }
23            }
24    }
```

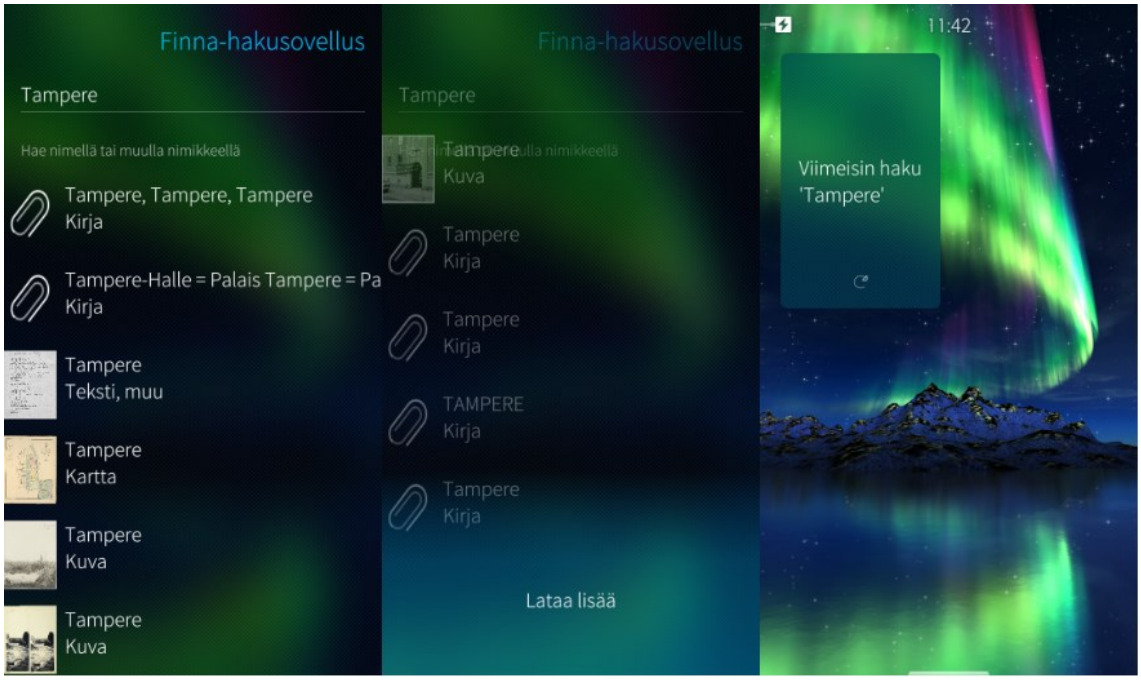
Ohjelma 14. *Delegate määrittää, miten mallin data näytetään näkymässä.*

```
1     PushUpMenu {
2
3         MenuItem {
4             text: qsTr("Lataa lisää")
5             onClicked: {
6                 // päivitä ja hae lisää tuloksia
7             }
8         }
9     }
```

Ohjelma 15. *PushUpMenuun voidaan liittää toiminto, jolla hakutuloksia ladataan ja näytetään lisää näkymässä.*

Ohjelmassa 14 on esitetty SilicaListViewin delegate, jolla voidaan kuvata mallin tarjoama data sekä määrittää, miten se näytetään. Lisäksi delegatessa käytetään ohjelmassa 12 esitettyjä imgurl, title ja type objekti-arvo-pareja. Ohjelmassa 15 on havainnollistettu PushUpMenun toimintaa. Sovellukseen voidaan lisätä myös hyvin yksinkertaisesti ohjelman 10 mukaisesti Cover-näkymä ja kytkeä virkistämistoimintoon sovelluksen mallin päi-

vittäminen. Käytännössä tämä tarkoittaa sitä, että ohjelma tyhjentää tallennetun ListModel-komponentin ja tekee uuden rajapintakyselyn. Mallin muuttunut data päivittyy myös dynaamisesti näkymälle.



Kuva 6. *Sovelluksen päänäkymä, vetovalikko sekä moniajonäkymä.*

Kuvassa 6 on esitetty toteutetun sovelluksen kolme keskeisintä näkymää. Lisäksi kuvassa keskellä on PushUpMenu lisätuloksien hakemiselle. Sovelluksen yksinkertaisuuden vuoksi siinä ei esiinny juuri käytettävyyden kannalta merkittäviä toimintoja. Sovellusta on kuitenkin mahdollista jatkokehittää melko yksinkertaisesti nyt, kun pääsuunnittelu ja valitut komponentit ovat valmiina. Hyödyllisiä toimintoja käyttäjiä ajatellen ovat esimerkiksi teosten saatavuus ja muut lisätiedot. Näitä varten sovellukseen voidaan luoda erillinen sivu nykyisen haku- ja näkymäsivun tapaan. Lisäksi hakutuloksista voidaan yksinkertaisesti tehdä interaktiivisia siten, että käyttäjä voi painaa hakutulosta ja PageStack siirtää sivun näkymän erilliselle sivulle. Erilliselle sivulle voidaan kerätä kooste yksittäisestä hakutuloksesta ja lisätä sinne samalla myös mahdolliset saatavuustiedot. Sovelluksen laajentamiseen ja jatkokehittämiseen ei kuitenkaan mennä tässä työssä sen pidemmälle.

YHTEENVETO

QtQuick yhdessä Qt-kehyksen kanssa on hyvä esimerkki deklaratiivisen kuvauskielen ja JavaScriptin tuomista hyödyistä, tehokkuudesta ja joustavuudesta sovelluskehityksessä. Lisäksi QtQuickiä voidaan laajentaa C++-kielellä, mikä mahdollistaa laitteistoläheisen ohjelmoinnin sekä kokonaisten kirjastojen tuomisen mukaan sovellukseen. C++ tarjoaa myös parhaimmillaan ratkaisuja ja laajentamismahdollisuuksia resurssi- ja laskentaintensiivisissä sovelluksissa. QtQuick sisältää itsessään jo kattavan määrän valmiiksi tehtyjä komponentteja, malleja ja näkymiä, joilla voidaan tehokkaasti luoda skaalautuvia sovelluksia eri laitteille. QtQuick ei siis rajoitu pelkästään työpöytäsovellusten ympäristöön, vaan sillä on mahdollista toteuttaa myös mobiilisovelluksia. Lisäksi QtQuick yhdessä Qt-kehyksen kanssa mahdollistaa alusta- ja laitteistoriippumattomuuden esimerkiksi käyttämällä ympäristömuuttujia ja ehtodirektiivejä laitteistospesifisten toteutuksien määrittämisessä.

Sailfish OS on eräs Linux-pohjainen mobiilikäyttöjärjestelmä, joka hyödyntää Qt ja QtQuick teknologioita. Sailfish on valittu työhön, sillä sen sovellukset toteutetaan ja ajetaan pääsääntöisesti QtQuick-sovelluksina. Sovelluskehitystä varten tarjotaan Sailfish SDK, johon kuuluu Qt-kehitysympäristö, oma kääntäjä ja emulaattori. Sailfish-ympäristöön on kehitetty oma natiivikomponenttikirjasto nimeltä Silica, joka sisältää paljon QtQuickistä tuttuja komponentteja ja kontrolleja. Ympäristöön tyyliteltyjen komponenttien lisäksi Silica sisältää myös suuren määrän käyttöjärjestelmän omia ikoneja. Sailfish ja Silica ovat hyviä esimerkkejä QtQuickin sovelluskohteista sekä laajennettavuudesta.

Sailfish-sovelluksen suunnittelussa konkretisoituivat QML-kuvauskielen ja JavaScriptin hyödyt sekä joustavuus. Samalla sovelluksessa tutustuttiin yleiseen mobiilisovelluksen arkkitehtuuriin, jossa sovellus tekee rajapintapyyntöjä ja luo sisällön rajapinnan tarjoaman datan pohjalta. Sovelluksen teossa havainnollistettiin konkreettisesti myös QtQuickin mallien ja näkymien vuorovaikutusta. Lisäksi sovelluksen tekeminen havainnollisti, että QtQuickillä on mahdollista toteuttaa yksinkertaisia ohjelmia pelkästään käyttämällä valmiita QML-komponentteja ja JavaScriptiä.

LÄHTEET

[1] Qt Documentation: Qt Quick. Saatavissa: https://doc.qt.io/qt-5/qmlapplications.html. [Viittauspäivä 29.1.2021]

[2] A. Nayyar, Developers - focus: A guide to qt programming, and the top tools for the task. Open Source for You, Dec 01 2019. Saatavissa: https://search-proquest-com.libproxy.tuni.fi/docview/2330972369

[3] Developers: Qt5: GUIs with QtQuick and QML, Open Source for You, 2015 Apr 01. Saatavissa: https://search-proquest-com.libproxy.tuni.fi/docview/1668236027

[4] Qt Documentation: QtQuick > Models and Views in Qt Quick. Saatavissa: https://doc.qt.io/qt-5/qtquick-modelviewsdata-modelview.html. [Viittauspäivä 12.3.2021]

[5] Qt Documentation: Integrating JavaScript in QML. Saatavissa: https://doc.qt.io/qt-5/qtquick-usecase-integratingjs.html. [Viittauspäivä 19.2.2021]

[6] Sailfish.org wiki: Combining C++ with QML. Saatavissa: https://sailfishos.org/wiki/Tutorial_-_Combining_C%2B%2B_with_QML. [Viittauspäivä 19.2.2021]

[7] D. Laure, A. Vasilyev, I. Paramonov, N. Kasatkina, Cross-platform development for Sailfish OS and Android: Architectural patterns and dictionary trainer application case study, 2016 19th Conference of Open Innovations Association (FRUCT), Jyväskylä, 2016, pp. 145–150, Saatavissa: https://ieeexplore.ieee.org/document/7892194

[8] J. Chmielewski, J. Flotyński, D. Rumiński, A. Wójtowicz, Declarative GUI descriptions for device-independent applications, Personal Ubiquitous Comput. Vol. 20, No. 2, April 2016, pp. 185–194. Saatavissa: https://doi-org.libproxy.tuni.fi/10.1007/s00779-016-0903-2

[9] K. Tzvetanov, U. Karabiyik, A first look at forensic analysis of sailfishos. Computers & security. Vol. 99, No. 102054, 2020 Dec. Saatavissa: https://doi.org/10.1016/j.cose.2020.102054

[10] Sailfish OS: Info. Saatavissa: https://sailfishos.org/info. [Viittauspäivä 29.1.2021]

[11] Sailfish Application Development. Saatavissa: https://sailfishos.org/wiki/Application_Development. [Viittauspäivä 5.2.2021]

[12] Sailfish Silica. Saatavissa: https://sailfishos.org/develop/docs/silica/. [Viittauspäivä 29.1.2021]

[13] Sailfish Silica -komponentit. Saatavissa: https://sailfishos.org/develop/docs/silica/sailfish-silica-all.html/. [Viittauspäivä 29.1.2021]

[14] A. Samoryadova, K. Kirpichenok, K. Kulakov, Experience in developing reference applications with Qt/QML, 2012 12th Conference of Open Innovations Association (FRUCT), Oulu, 2012, pp. 1–7. Saatavissa: https://ieeexplore-ieee-org.libproxy.tuni.fi/document/8122094