Tallinna Ülikool

Informaatika Instituut

FictionBook 2 lugeja loomine Maemole Python/Qt põhjal

Seminaritöö

Autor: Olga Kravtsenko

Juhendaja: Inga Petuhhov

Tallinn 2011

Sisukord

Sissejuhatus

Tänapäeva tarkvaraturul on pakkuda suur hulk mobiilplatvorme, mis hoiavad meie seadmeid töös. Osad, nagu Nokia loodud Symbian, on turul juba kaua eksisteerinud, samas kui teised, nt iOS või Android, on alles hiljuti esile kerkinud.

Kui võtta aluseks üha kasvavat mobiiltelefonide OS-ide arvu, siis on ühtselt ühilduva rakenduse loomine muutumas üha raskemaks ülesandeks, kuna need peavad olema kooskõlas kõikide operatsioonisüsteemide rakendusliidestega ning silmas tuleb pidada ka võimalikke tuleviku uuenduste mõju ning nendega ühildumist.

Väljavõte FictionBook 2.0 kohta avaldatud tööst kirjutab tõlgituna järgnevat: „Internetis on tohutu hulk digitaalseid tekste, mis ei pruugi alati olla samas formaadis, mis muudab sobiliku teksti leidmise seadmetele nagu lauaarvuti, tahvelarvuti või isegi elektroonilise raamatu, muuta väga keeruliseks. FictionBook 2.0 standard on loodud pakkuma tuge ja ühilduvust üle kõikide platvormide" (*FictionBook - short description*).

Seminaritöö aluseks sai valitud just Maemo, kuna uusimaks seadmeks turul on N900, mis omab suurt ekraani, olles sobilik elektrooniliste raamatute lugemiseks. Lisaks on tegemist mobiilse seadmega, mis on meiega igal hetkel kaasas, olgu selleks siis ühistranspordis, arsti järjekorras või mõnes muus olukorras, kus on ootamist. Selle rakenduse abil on võimalik kõiki oma lemmik teoseid endaga kaasas kanda ja seda kõike enda taskus.

Antud seminaritöö eesmärgiks on tutvuda Python/Qt tehnoloogiatega FictionBook 2 lugeja loomise näitel. Selleks on vaja järgmist:

1. Tutvuda FictionBook 2 formaadi, Qt 4 raamistiku ning Maemole Python ja Qt tehnoloogiate põhjal arendamisega.

2. Nende teadmiste põhjal lihtsa nn. *proof-of-concept* rakenduse luua.

Oma diplomitöös soovin sellest täielikku lõpptarbija rakenduse luua.

1. Teooria

1.1. Peamised mobiilplatvormid

Praegune mobiilsete seadete maastik pakub laia valikut platvorme rahuldamaks erinevate klientide vajadusi. Tähtsaimad mobiiloperatsioonisüsteemide tootjad turul on:

- Nokia
- Google
- Apple
- Research In Motion
- Microsoft
- Hewlett-Packard
- Samsung
- Intel

2010. aasta 3. kvartali seisuga on turu liidriks Nokia, kelle Symbiani platvormiga tooteid on müüdud 37,6% kogu turu nutitelefonide mahust, temale järgneb Google´i Android 22,7% turuosaga, Apple´i iOS 15,7% turuosaga ning RIM´i BlackBerry 16,0% turuosast (Christy Pettey, Laurence Goasduff, veebruar, 2011). Lisaks eelmainitud suurtele tegijatele tasuks mainida ka järgnevaid väiksema turuosaga operatsioonisüsteeme:

- Windows Mobile (Microsoft)
- Windows Phone (Microsoft)
- Linux (open source, GPL)

- Palm webOS (HP)
- bada (Samsung)
- MeeGo (Nokia ja Intel) – avaldamata
- Maemo (Nokia)

Ülalmainitud operatsioonisüsteemide osakaal turul on 2010 aasta seisuga tühine, kuna Microsoft produktide osakaal on pelgalt 4,2% turuosast ning kõikidel teistel kokku 3,8% turuosast (Christy Pettey, Laurence Goasduff, veebruar, 2011).

Tänini on mobiilplatvormide arenduses puudus ühtse programmeerimise standardi järgi, kuna kõik platvormid, nii kasutaja- kui ka rakendus platvorm, kasutavad erinevaid liideseid. Seetõttu ongi üpris keerukas luua universaalset rakendust kõikidele platvormidele, eriti kui iga platvorm nõuab oma lähenemist ning pahatihti on igaühel neist oma programmeerimise põhikeel.

Näitena võib tuua iOS (Apple iPhone OS) programmeerimise põhikeele, milleks on Objective-C, mis ei ole just kõige tuntum ja levinum. Sellele platvormile rakenduse arendamine tähendab tavaliselt seda, et tarkvara loomisel tuleb programmeerijatel nn uus "võõrkeel" ära õppida.

Google seevastu lähenes Androidi loomisel hoopis teisest küljest, luues modifitseeritud Linuxi tuuma põhjale virtuaalse masina nimega Dalvik, mis võimaldas Androidil käitada Java´t. Selle sammu tulemuseks on see, et Dalvik võimaldab kirjutada rakendused valmis Java´s, mis on hiljem Androidiga ühilduvad. Kuna Java tarkvara arendajaid on palju rohkem, kui Objective-C omi, siis ei ole ka imekspandav, miks Androidi turuosa tõusis vahemikus 2009 II kvartal kuni 2010 II kvartal tühiselt 1,8%-lt 17,2%-le (Christy Pettey, Laurence Goasduff, august, 2010).

Tänu C/C++ kestvale populaarsusele on ka Nokia loodud Symbian/Maemo edukas tänase päevani. Tänu tõsiasjale, et Nokia on hetke tuntuim bränd turul ning Maemo nende avatud lähtekoodiga platvorm, otsustasingi valida Maemo enda seminaritöö aluseks..

1.2. Maemo

Maemo on tarkvara platvorm, mis on loodud spetsiaalset mobiilsete seadmete, nagu nutitelefonide ja tahvelarvutite, peal kasutamiseks. Selle loomisest alates on platvorm läbi teinud märkimisväärse hulga muudatusi ning tänase seisuga koosneb see Maemo OS-st ning Maemo SDK-st. Maemo OS on peaasjalikult Linuxi distributsioon, põhineb Debian´il ning on avatud lähtekoodiga, kui välja arvata mõned patenteeritud komponendid.

Praegune väljalase on versioon 5, koodnimega Fremantle. Võrreldes eelneva versiooniga on sel rakendatud valik uuemat tehnoloogiat ning palju enam sõbralikumat kasutajaliidest. See versioon on näiteks baassüsteemiks Nokia N900 telefonile. Erinevalt Symbianist, mis on massidele mõeldud OS, on Maemo loodud peaasjalikult arvutihuvilisi inimesi silmas pidades. Kindlasti mitte ei ole ta kasutaja vastu ebasõbralik, vaid annab pigem oskuslikule tarbijale antud OS-ga toote kasutamisel palju enam valikuvõimalusi.

Nagu iga teine Linuxi distributsioon, omab ka Maemo terminali, kasutajakontode seadistamise võimalust (ka juurkasutajaga sisenemise võimalust) ning palju tuntud Linuxi utiliite ja rakendusi. Näitena võib tuua võimaluse käitada OS-l SSH taustprogrammi, mis võimaldab turvalise sissepääsu teisest seadest, nt arvutist.

Maemo kasutab enda rakenduste installeerimiseks laialdaselt tuntud dpkg paketihaldussüsteemi. Antud süsteemiga on võimalik uuendada tarkvara läbi USB ühenduse või siis kasutades APT utiliiti. Siinkohal vajab mainimist, et antud graafilist kasutajaliidest luues on silmas peetud nn keskmist kasutajat, kes üldjuhul ei leia vajadust või siis ei oska uuendada tarkvara kasutades terminali.

Kui välja tuua Maemo parim omadus, siis kindlasti on selleks vabadus luua ja arendada rakendusi sellele platvormile. Nii Maemol, kui ka Maemo SDK-l on baaskeeleks C, lisaks on ka võimalus kirjutada programme Java´s, kuid selleks on vaja kasutada Jalimo JVM-i. Need ei ole aga siiski ainsad võimalused, Maemo võimaldab programme luua ka Ruby, Mono ja Python´iga. Graafilist kasutajaliidest on võimalik kirjutada kasutades kas GTK+ vidinate tööriistaribaga või siis täiesti uuena kasutuses oleva Qt raamistiku abil, millele Fremantle pakkus kõige esimesena tuge.

Järgmine Maemo väljalase lubab viia Qt järgmisele tasemele ning muuta selle vaikimisi graafilise kasutajaliidese teegiks tervele platvormile.

Antud seminaritöös käsitlen Maemole rakenduse loomist, kasutades Python´it ja Qt 4.

1.3. Python ja Qt

Qt on C++-s kirjutatud rakenduste arendamisel kasutatav tasuta, avatud lähtekoodiga ning platvormidega ühilduv raamistik, mis on sobilik nii graafilistele kasutajaliidestele, kui ka nendele rakendustele, mis seda ei oma. Tuntumad Qt baasil tehtud programmid on Skype, Google Earth, KDE, Adobe Photoshop Album, VLC meediapleier, VirtualBox ja paljud muud.

Algselt arendati Qt-d Norra firma Trolltech poolt, kuid alates 2008. aastast on Trolltech Nokia omanduses ning sellest ajast alates on seda arendanud Nokia osakond järgnevatele platvormidele:

- Linux/X11
- Mac OS X
- Windows
- Embedded Linux
- Windows CE / Mobile
- Symbian
- Maemo

Tänu raamistiku avatud lähtekoodile, on sellest mitmeid erinevaid lahendusi tehtud peale Nokia esmast väljalaset, nt. Qt-iPhone, Android-Lighthouse, Qt webOS-le, Qt Amazon Kindle DX-le.

2010 aastalõpu seisuga on uusimaks Qt väljalaskeks 4.7, mis kujutab endast lihtsalt kasutatavat, samas äärmiselt võimsat raamistikku, loomaks kasutajaliideseid mistõttu osutus see ka minu poolt valituks.

Olgugi, et keeleks Qt-le on C++, on programmi kirjutamise võimalusi alates Javast, Adast ja C#-st Ruby, Perl´i ja Pascal´ini välja ning PHP (*Programming Language Support — Qt - A cross-platform application and UI framework)*.

Kuna C++ pole ainuke toetatud keel rakenduse loomiseks, siis otsustasin luua programmi Python´is ning seda alljärgnevatel põhjustel:

- Programmi loomine Python´is on palju kergem kui C++
- Maemo toetab igakülgselt Pythonit (nagu peaaegu enamik, kui mitte kõik Linuxi distributsioonid)
- Jõudluse osas ei tohiks erinevusi C++ ja Python´i vahel tekkida, kuna loodav rakendus on väikesemahuline. Juhul kui tekivad takistused, siis alati on võimalik kitsaskohad C++ umber kirjutada

Kuna Maemo ühildub Python 2.5.4-ga, siis antud töö tegemisel kasutan eelmainitud versiooni.

Pythonit toetavad vähemalt kolm Qt sidusprogrammi: PySide, PythonQt ja PyQt, millest viimane on vast kõige küpsem ja võimaluste rohkeim, mis on peamiseks põhjuseks, miks hakkan kasutama neid seminaritöö tegemisel.

1.4. FictionBook kui üks eBook´i vorminguid

FictionBook on eBooki formaat, mis algselt on loodud Venemaal Dmitry Gribov´i poolt ning muutus peatsel laialdaselt kasutatavaks nii programmeerijate kui ka tarbijate seas. See on XML põhine ning tõlgendab dokumendi struktuuri kasutades erinevaid silte, märgistamaks erinevaid struktuuriüksusi, nt. epigraafe, tsitaate jpm. Lisaks eelmainitule sisaldab FictionBook fail ka meta-andmeid dokumendi kohta, nagu autori nime, pealkirja jt. andmeid.

Tänu lihtsale, kuid võimekale XML formaadi omapärale, on see sobilik automaatse töötlemise jaoks, konverteerimiseks teistesse vormingutesse, indekseerimiseks ning ka teekide haldamiseks.

Antud formaadi huvitavaks omaduseks on võime salvestada binaar (nt pilte) andmeid, kasutades `<binary>` silti ning Base64 kodeeringut (*XML Schema Fictionbook 2.1*).

Tänase seisuga on FictionBook´il teine väljalase, olgugi, et FictionBook vormingus failis omavad `.fb2` laiendit.

2. Praktika

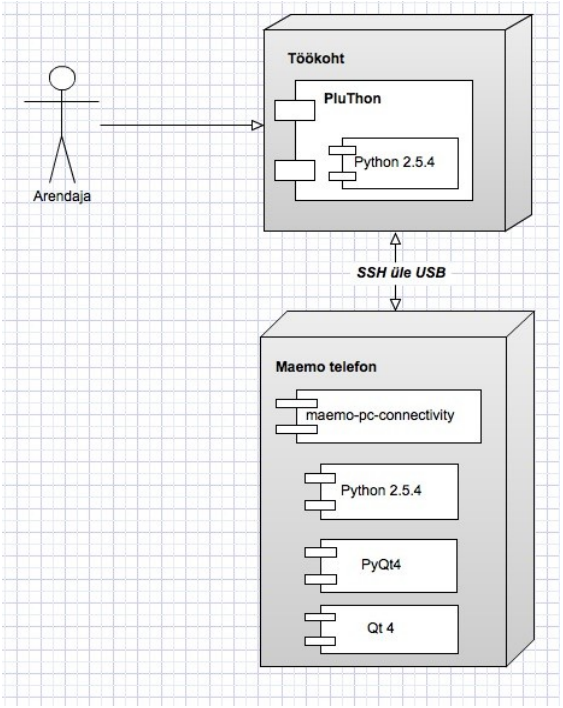
2.1. Programmi loomise keskkond

Rakenduse arenduskeskkonna ettevalmistamine nõuab paari lihtsalt tegevust. Vaja on IDE-d ja turvalist ühendust seadega, mis võimaldaks ühe klikiga tarkvara rakendada seadmel. Antud seminaritöös kavatsen kasutada PluThon´it IDE-na, rakendan SSH serverit Maemoga varustatud seadmel ning kopeerin andmed USB kaabli abil.

PluThoni installeerimine arvutisse on lihtne, kuna see on versioon Eclipse´st, mis on loodud puhtalt mobiilse tarkvara arenduseks. Sisuliselt oleks võimalik kasutada ka Eclipse´i pistikprogrammi, kuid minu valitud lahendus on sobilikum kahel põhjusel: esmalt on tunduvalt kindlam kasutada eraldi IDE-d, mis pakub meile täpselt seda, mida vajame; teiseks ei ole PluThon´il ebavajalikke Eclipse´i pistikprogramme nagu JDT (Java Development Tools) ning palju teisi, mis muudavad selle tunduvalt kiiremini ning mugavamalt kasutatavaks.

Maemo arendamiseks on vaja järgmist (Joonis 1: Arendamise keskkonna diagramm, *PyQt4 for Maemo*):

1. Python 2.5.4
2. maemo-pc-connectivity
3. PyQt4
4. Qt 4



Joonis 1: Arendamise keskkonna diagramm

PyQt4

Et Qt kasutada Pythonis, on vaja PyQt4 telefonile installeerida. Selleks on vaja:

1. Avada oma tarvikul terminal
2. Sisestada ´`sudo gainroot`´ - eelistatuim viis, kuidas saada ligipääsu Maemo juurandmetele
3. Lubada "Extras-devel" hoidla avades oma tarviku veebibrauseris lehekülg: http://pyqt.garage.maemo.org/pyqt4.install
4. Käivitada järgnev töökäsk terminalis:

```
apt-get install python-qt4-core python-qt4-gui
```

maemo-pc-connectivity

Peale seda on vaja installeerida maemo-pc-connectivity, ehk Maemo ühenduse pakk, võimaldamaks turvaline ühendus PluThoni ja seadme vahel. Selleks on vaja järgnev töökäsk telefonil terminalis käivitada:

```
apt-get install maemo-pc-connectivity
```

See installeerib kõik vajalikud graafika rakendused ning SSH taustprogrammi samuti. Kui hiljem peaks vajadus tekkima seda kontrollida, siis võib anda alloleva töökäsu tarvikule:

```
ps axu | grep ssh
```

Kui on näha allolevaga sarnast vastust ridade seas, siis on seadistus olnud edukas ja kõik toimib (tärnid väljendavad numbreid, mis võivad erineda, nagu protsessi ID)

```
**** root     **** S    /usr/sbin/sshd -D
```

Python 2.5.4 ja Qt 4 peavad juba olema installeeritud Maemo 5 tarvikul, sellepärast viimase sammuna seadistamisel tuleb minna Seadetesse ning sealt valida PC-Connectivity manager, ehk ühenduste haldur, ning valida ühenduseks USB. WLAN ja Bluetooth ühendusega on samuti võimalik kasutada PluThon´it, kuid soovituslik on siiski kasutada USB ühendust, mis on kindlasti kiirema ühenduskiirusega. Samal ajal võimaldab see ka tarvikul olla vooluvõrguga seotud, mis on väga oluline, kuna teatavasti tarkvara arendamine on väga suure energiakuluga.

Olles aktiveerinud USB ühenduse enda seadmel, viimaseks sammuks on ühendada tarvik arvutiga USB kaabli abil. Antud juhul oleks arukas PluThon´i “Hello,world” näidisega kontrollida, kas kõik sammud on korrektselt läbitud ja süsteem toimib.

Kõik alltoodud koodinäited on autori enda programmeeritud.

2.2. Lihtsa FictionBook lugeja järk-järguline loomine

FictionBook on XML-põhine formaat, mistõttu on loogiline, et esimeseks sammuks saab olema XML parseri hankimine dokumentidega manipuleerimiseks ning struktuuriüksustes navigeerimiseks. Selles töös olen otsustanud kasutada teek nimega lxml, mis on mõeldud XML-ga manipuleerimiseks.

```
from lxml import etree

tree = etree.parse("thinkpython.fb2")

iter = tree.getiterator()

result = []

for element in iter:
    print element
```

Koodinäide 1: Näide lxml kasutamisest

See toob nähtavale terve XML puu koos kõikide siltide ja nimeruumidega. See ei ole küll see, mida lõpptulemuseks soovime, kuid on piisav, et testida seda osa, mida soovime kasutada lxml teegist.

Meie lihtsal lugejal saavad olema järgnevad elemendid:

- “Open book” nupp võimaldamaks avada .fb2 faile seadest
- “Rich text” ala, mis kuvab raamatu sisu, tugiliidest, poolpaksus ning kaldkirja kirjastiile
- Kuvatakse ainult teksti osa, kogu muu info, st. kujundid, metaandmed ja muu vorming jääb kuvamata

- Ainsad vormingu reeglid, mida kuvame on taandread ja lõigud, mis aitavad meil mõista reavahetusi

Esimese väljalaske jaoks on see piisav, kuna meie eesmärk on hoida asju lihtsana ning seeläbi eksimise võimalused minimaalsuseni viia. Hiljem on meil võimalus samale platvormine tuginedes rakendust täiendada.

Alustame klassi loomisega käivitamise meetodiga ja anname nupule vajaliku töötleja:

```
class Application(QApplication):

    def __init__(self):
        QApplication.__init__(self, sys.argv)
        QApplication.setOverrideCursor(QCursor(QtCore.Qt.BlankCursor))

        self.window = QWidget()
        self.window.setWindowTitle(self.tr("Fb2Reader"))

        textBrowser = QTextBrowser()
        textBrowser.setReadOnly(True)

        self.textBrowser = textBrowser

        self.button = QPushButton(self.tr("Open book"), self.window)

        layout = QVBoxLayout()
        layout.addWidget(self.button, QtCore.Qt.AlignTop)
        layout.addWidget(self.textBrowser, QtCore.Qt.AlignCenter)

        self.window.setLayout(layout)

        self.connect(self.button, QtCore.SIGNAL('clicked()'),
self.handlePressed)

        self.window.show()
```

Koodinäide 2: Rakenduse klassi käivitamise meetod

Oma rich text alana kasutame QTextBrowser´it ning süsteemi nupuna kasutame QPushButton´it. Lisaks neile kahele vajame “handlePressed” meetodit, käitlemaks nupu klikke:

```
    def handlePressed(self):
        filename = self.chooseFile()
```

```
        self.book = self.parseBook(str(filename))
        text = self.book
        self.textBrowser.setHtml(text)
```

Koodinäide 3: "Open book" nupu käitleja

Antud tegevuse idee seisneb selles, et nupule vajutus avaks meile "Choose file", ehk ava fail tekstikasti, mispeale me töötleme raamatu sisu ning kuvame selle kasutajale. Tähtis on teada, et FB2 kasutab silte, mis on sarnased HTML siltidele, mistõttu kasutamegi Qt QTextBrowserit lihtsamaks teksti tõlgendamiseks. Peame ainult saama vajalikud XML osad raamatu failist (antud väljalaske väljundiks seadsime kindlal hulgal silte) ning söödame need ette eelpool mainitud rakenduse klassile ja ülejäänu teksti tõlgendamise eest kannab hoolt juba Qt tugiraamistik.

Meil on nüüd vaja kaht meetodit rakendada: chooseFile() ja parseBook(str) – vt. „Lisa 1: XML töötlemiseks ja teisendamiseks vajalikud meetodid„

Esimene meetod dialoogi aken, mis laseb kasutajal valida seadme mälust faili, filtreerides välja kõik laiendid mida programm ei toeta ning jättes nähtavale ainult .fb2 laiendid. Teine meetod kordab uuesti XML puud, sarnaselt seminaritöö kõige esimesele välja toodud skriptile, ning pakib lahti ainult toetatavad sildid (`<p>`, `<emphasis>` and `<strong>`) vajaliku *namespace*-iga (FB2 XML *namespace*-i nimeks ehk selle URI-ks on "http://www.gribuser.ru/xml/fictionbook/2.0", *XML Schema Fictionbook 2.1*).

Sisuliselt lisab programm kõik leitud sildid nimekirjana puhvrisse, optimeerimaks teksti juppide kokkupanemist. Lihtsalt suurel hulgal tekstijuppide kokku sobitamine viib üks hetk kindlalt läbikukkumise ja kehvade tulemusteni, kuna jupid on muutumatud, seega Python peaks iga jupiga tegema uue ahela, et neid kokku panna. Kasutades nimekirja puhvrit ja lisades kõik jupid üksteise järgi pikaks ketiks, väldime me antud meetodi kitsaskohta (parseBook() meetod, "result" puhver realiseeritud massiivi põhjal).

Kolmas ja neljas meetod on kõigest teise tugimeetodid, mis sisuliselt jagavad koodi väiksemateks loogilisteks osadeks. Esmalt lisame selle meetodi rakenduse klassile:

```
    def run(self):
        return self.exec_()
```

Koodinäide 4: Sisenemise meetod run()

Ja siis lisame selle oma main.py Python mooduli lõppu

```
if __name__ == "__main__":
    app = Application()
    sys.exit(app.run())
```

Koodinäide 5: põhiprogrammi funktsioon – objekti loomine ja selle käitamine

See lihtsalt loob rakenduse objekti ja käitab seda, juhul kui see käivitatakse _main_ ülemise klassi skripti keskkonnast.

Meie FB2 lugeja on põhimõtteliselt valmis, kui välja arvata mõned pisivead, mille parandamine on järgmine arendamise samm – refaktoreering.

Esmalt on meil paar puudust koodis:

5. Kui kasutaja ei vali ühtegi faili, lähtestab chooseFile() meetod valiku kui None ning kui see edastatakse lxml parserile, see tõstatab probleemi.

6. HTML-l ei ole ühtegi <emphasis> silti. Vastav HTML silt rõhutatud tekstile on , mis tähendab seda, et meie teksti ei rõhutata, kui see edastatakse Qt HTML tõlgendamise klassile.

7. Tavaline FB2 lõik näeb välja selline:

```
<p>This is a <strong>typical</strong> FB2 <emphasis>rich</emphasis> text</p>
```

Praegust koodi kasutades kaotame me teksti osa, mis jääb lõppsildi ja avamise sildi <emphasis> vahele ning ka selle osa tekstist, mis jääb lõppsildi </emphasis> ja lõppsildi järgi. Põhjus seisneb selles, kuidas lxml töötleb puud ja seetõttu ei kuulu tekst avamise sildi hulka, vaid pigem salvestatakse vasaku külje elementide “tail” väljasse, mis iseenesest on hea omadus, sest seeläbi saame meie lihtsamini väljundit määrata, mida muidugi tuleb meeles pidada, kuna esmases versioonis me seda ei teinud.

Teine väike probleem selle koodi puhul seisneb selles, et kogu loogika on meil paigutatud ühte klassi. Isegi sellise väikesemahulise rakenduse puhul nagu see, tuleks meil olulisi asju eraldi paigutada, niisiis loome me eraldi Book klassi, samalajal parandades osad eelpoolmainitud pisivead – vt. „Lisa 2: refaktoreerimine – uus Book klass koos töötlemise meetodite ja FB2st HTML teisendamine – esimene samm“.

Allpool oleme vastavusse seadnud nii FB2 kui ka HTML sildid, millega kõrvaldame vea number 2. Lisasime tuge veel ühele sildile nimega <section>, mis nagu nimest juba lugeda võib tähistab teksti lõiku. Me oleme selle sildi vastavusse seadnud vastavalt HTML-i ühilduva sildi -ga. Lisaks eelmainitule parandasime vea number 3, lisades juurde *if* bloki meetodile, mis lisab HTML sildid sõnajuppide puhvrile, mis omakorda kontrollib “tail” väljad ja lisab neid vaste korral puhvrisse.

Vt. „Lisa 3: refaktoreerimine – main.py moodul koos kõikide raamatuga seotud meetoditega lahtipakituna Book klassis “ – siin on näha, mismoodi meie uus main.py moodul välja näeb.

Siinkohal tooksin välja, et esimsese pisivea parandasime sellega, et lisasime lihtsa kontrolli faili nime tarbeks, ehk siis kui fail ei ole saadaval, siis naaseme tagasi nupu töötleja juurde ja rakenduse kulg jätkub. Me nimetasime ka Application klassi umber märksa arusaadavama Fb2ReaderApp vastu, tähistamaks, et tegemist on meie rakenduse juht klassiga.

See näide ilmestab seda, kuidas pelgalt 116 Pythoni koodi reaga (nagu on näha «`cat `find src/ -name "*.py"` | wc -l`» terminali aknast, mis sisaldab kõiki import ja stereotüüpseid koode) on Qt-d kasutades võimalik luua korralik Maemo platvormil FB2 lugeja.

Kokkuvõte

Kaasaaegne mobiilsete tarvikute maailm tunneb mitmeid platvorme, mille suurim osa kuulub siiani Nokiale ja nende Symbiani ning Maemo platvormile. Sellest olenemata on Google teinud tiigrihüppe ning suutnud enda Androidi edukalt turustada ja seni tundub, et massidele see meeldib. Mitte kuigi kaugel maas ei ole Apple enda loodud iOS-ga, ainus takistus selle OS rakenduste arenduse puhul on muidugi selle vähemlevinud Objective-C, mis on kahjuks ainus sobilik ja ühilduv iOS-ga.

Käesolevas seminaritöös võtsin enda eesmärgiks luua avatud lähtekoodiga Maemo platvormile luua Qt ning Pythoni abil lihtne, kuid funktsionaalne ning kasulik FictionBook lugeja.

Käesolev seminaritöö tõestab, et C, milles Maemo on ise kirjutatud, ei ole ainsaks selle arenduse keeleks. Samuti näitasin kuivõrd lihtne on dünaamilise ning üsna kergesti õpitav Pythoni ja võimeka Qt C++ raamistiku abil rakendust luua. Viimaseks annab see ülevaate FictionBook´i formaadist, mis on kasutusel struktureeritud eBook´ide taasesitamisel.

Lisaks eelmainitule pakun ma järk-järgulist ülevaadet ning juhendust kuidas seadistada Python/Qt/Maemo arendamise keskkonda ning juhendan lugejat läbi tegema samme lihtsa ning funktsionaalse rakenduse loomiseks.

Kasutatud allikad

Christy Pettey, Laurence Goasduff (august 2010). *Gartner Says Worldwide Mobile Device Sales Grew 13.8 Percent in Second Quarter of 2010, But Competition Drove Prices Down.* Kasutamise kuupäev 6. märts 2011, allikas http://www.gartner.com/it/page.jsp?id=1543014

Christy Pettey, Laurence Goasduff (veebruar 2011). *Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010*. Kasutamise kuupäev 6. märts 2011, allikas http://www.gartner.com/it/page.jsp?id=1421013

FictionBook – short description. Kasutamise kuupäev 6. märts 2011, allikas http://www.gribuser.ru/xml/fictionbook/abstract2.0.html

Maemo PC Connectivity. Kasutamise kuupäev 6. märts 2011, allikas http://pc-connectivity.garage.maemo.org/2nd_edition/index.html

PluThon Project. Kasutamise kuupäev 6. märts 2011, allikas http://pluthon.garage.maemo.org/2nd_edition/

Programming Language Support — Qt - A cross-platform application and UI framework. Kasutamise kuupäev 6. märts 2011, allikas http://qt.nokia.com/products/programming-language-support

PyQt4 for Maemo. Kasutamise kuupäev 6. märts 2011, allikas http://pyqt.garage.maemo.org/

Qt Reference Documentation. Kasutamise kuupäev 6. märts 2011, allikas http://doc.qt.nokia.com/4.7/index.html

XML Schema Fictionbook 2.1. Kasutamise kuupäev 6. märts 2011, allikas http://www.fictionbook.org/index.php/Eng:XML_Schema_Fictionbook_2.1

Lisad: koodinäited

Lisa 1: XML töötlemiseks ja teisendamiseks vajalikud meetodid

```
def chooseFile(self):
        parent = self.window

        # self.tr(str) is a method for i18n (QObject: tr)
        title = self.tr("Open book")
        basePath = "."
        filter = self.tr("FB2 book files (*.fb2)")

        return QFileDialog.getOpenFileName(parent, title, basePath,
filter)

    def parseBook(self, filename):
        # parse the XML tree for a given filename
        tree = etree.parse(filename)

        # get the tree iterator to traverse the tree node by node
        iter = tree.getiterator()

        # create a string buffer for optimal string concatenation
        result = []

        # walk the tree
        for element in iter:
            # ... and parse each element
            self.parseElement(result, element)

        # construct the whole string from the buffer
        return "".join(result)

    def parseElement(self, buf, element):
        # for this version we are only parsing the following three
        # tags
        allowedTags = ["p", "emphasis", "strong"]

        # for each supported tag
        for tagName in allowedTags:
            # first, give the tag a proper namespace
            tagWithNs = "{%s}%s" %
("http://www.gribuser.ru/xml/fictionbook/2.0", tagName)

            # then see if the tag matches any of the supported ones
            # and if it's not empty
```

```
        if element.tag == tagWithNs and element.text:
            self.parseTag(buf, tagName, element)

# parses the tag, adding it to the buffer
def parseTag(self, buf, tagName, element):
    buf += "<%s>" % tagName
    buf += element.text
    buf += "</%s>" % tagName
```

Lisa 2: refaktoreerimine – uus Book klass koos töötlemise meetodite ja FB2st HTML teisendamine – esimene samm

```
from lxml import etree

class Book:

    # FictionBook 2 XML namespace URI
    FB2_NS = "http://www.gribuser.ru/xml/fictionbook/2.0"

    # FB2 to HTML tags mapping
    fb2ToHtml = {"p": "p",
                 "strong": "strong",
                 "emphasis": "em",
                 "section": "div"}

    # get the supported FB2 tags
    supportedFb2Tags = fb2ToHtml.keys()

    def __init__(self, filename):
        self.filename = filename
        self.html = None

    # parse the book lazily and cache its html content for easier
    # and faster access later on
    def asHtml(self):
        if not self.html:
            self.parseBook()

        return self.html

    def parseBook(self):
        # parse the XML tree for a given filename
        tree = etree.parse(self.filename)

        # get the tree iterator to traverse the tree node by node
        iter = tree.getiterator()

        # create a string buffer for optimal string concatenation
        result = []

        # walk the tree
        for element in iter:
            # parse each element and append it to the buffer
            Book.appendElement(result, element)
```

```
        # construct the whole string from the buffer
        self.html = "".join(result)

    @classmethod
    def appendElement(cls, buf, element):
        # for each supported tag
        for tag in cls.supportedFb2Tags:
            # first, give the tag a proper namespace
            tagWithNs = "{%s}%s" % (cls.FB2_NS, tag)

            # then see if the tag matches any of the supported ones
            # and if it's not empty
            if element.tag == tagWithNs and element.text:
                cls.appendTagAsHtml(buf, cls.fb2ToHtml[tag], element)

    # parses the tag, adding it to the buffer
    @staticmethod
    def appendTagAsHtml(buf, tag, el):
        buf += "<%(tag)s>%(text)s</%(tag)s>" % {"tag": tag, "text":
el.text}
        # fix the missing text bug by using the "tail" field
        if el.tail:
            buf += el.tail
```

Lisa 3: refaktoreerimine – main.py moodul koos kõikide raamatuga seotud meetoditega lahtipakituna Book klassis

```
#!/usr/bin/env python2.5

import sys

from PyQt4 import QtCore

from PyQt4.QtGui import QApplication
from PyQt4.QtGui import QCursor
from PyQt4.QtGui import QWidget
from PyQt4.QtGui import QPushButton
from PyQt4.QtGui import QVBoxLayout
from PyQt4.QtGui import QMessageBox
from PyQt4.QtGui import QFileDialog
from PyQt4.QtGui import QTextBrowser

from Book import Book

class Fb2ReaderApp(QApplication):

    def __init__(self):
        # call the super constructor
        QApplication.__init__(self, sys.argv)

        # set a blank cursor
        QApplication.setOverrideCursor(QCursor(QtCore.Qt.BlankCursor))

        # create the window
        self.window = QWidget()

        # set the window title to something sensible
        self.window.setWindowTitle(self.tr("Fb2Reader"))

        # create an HTML-supporting text browser widget
        # this will be our reading area
        textBrowser = QTextBrowser()

        # we only want to read, so set it to read-only
        textBrowser.setReadOnly(True)

        self.textBrowser = textBrowser

        # initialise the "Open book" button
```

```
        self.openButton = QPushButton(self.tr("Open book"),
self.window)


        # initialise the layout - we are going to use Vertical Box
        # layout to line up widgets vertically
        layout = QVBoxLayout()

        # add the open button to the top
        layout.addWidget(self.openButton, QtCore.Qt.AlignTop)

        # ... and the text browser at the bottom
        layout.addWidget(self.textBrowser, QtCore.Qt.AlignCenter)

        # apply the layout to the window
        self.window.setLayout(layout)

        # add a click handler for the open button
        self.connect(self.openButton, QtCore.SIGNAL('clicked()'),
                     self.openClicked)

        # show the window
        self.window.show()

    # wrap the exec_() method
    def run(self):
        return self.exec_()

    # open book handler method
    def openClicked(self):
        # open the "choose file" dialogue window, get the filename
        filename = self.chooseFile()

        # if no file was chosen, don't do anything further, return
        if not filename:
            return

        # create a book from the file, set it to a field
        self.book = Book(str(filename))

        # pass the HTML of the parsed book to the text browser,
        # so that it can be rendered
        self.textBrowser.setHtml(self.book.asHtml())

    def chooseFile(self):
        parent = self.window
        title = self.tr("Open book")
        basePath = "."
        filter = self.tr("FB2 book files (*.fb2)")
```

```
        # open the "choose file" dialogue window and return the result
        return QFileDialog.getOpenFileName(parent, title, basePath,
filter)

if __name__ == "__main__":
    app = Fb2ReaderApp()
    sys.exit(app.run())
```