Mikko Heikura

S60-SOVELLUKSEN LOKALISOINTI

S60-SOVELLUKSEN LOKALISOINTI

Mikko Heikura
Opinnäytetyö
Kevät 2011
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

ALKULAUSE

Insinöörityön aiheena on ”S60-sovelluksen lokalisointi”. Työ on tehty ResComi Oy:lle. Insinöörityön valvojana toimi Jouni Kivirinta Oulun seudun ammattikorkeakoulun Raahen tekniikan ja talouden kampukselta.

Kiitokset ResComi Oy:n toimitusjohtaja Raimo Seikkalalle insinöörityön aiheesta, joka kartutti osaamistani Symbian-ohjelmoinnin parissa. Kiitokset myös insinöörityöni valvojalle, Jouni Kivirinnalle, ResComi Oy:n työntekijöille ja ystäville, jotka jakoivat neuvoja ja osaamistaan tarvittaessa.

Oulussa 6.4.2011

Mikko Heikura

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu, Raahen tekniikan ja talouden kampus
Tietotekniikan koulutusohjelma, Sulautetut ohjelmistot

Tekijä: Mikko Heikura
Opinnäytetyön nimi: S60-sovelluksen lokalisointi
Työn ohjaaja: Jouni Kivirinta
Työn valmistumislukukausi ja -vuosi: Kevät 2011 Sivumäärä: 36 + 5

Insinöörityön tavoitteena oli toteuttaa ResComi Oy:n tekemään S60-sovellukseen lokalisointi. Tarkoituksena oli toteuttaa lokalisointi suomen kielen lisäksi englannin kielellä Symbian-sovelluskehityksen suosittelemalla tavalla sekä suunnitella ja toteuttaa lokalisointi tietokantapohjaisena ratkaisuna. Tarkoituksena oli myös helpottaa uusien kielien lisääminen sovellukseen.

Insinöörityön tuloksena saatiin toteutettua S60-sovelluksen lokalisointi, joka jatkokehityksen kannalta helpottaa uusien kielien lisäämisen sovelluksen käyttökieliksi. Tietokantapohjaisen toteutuksen myötä päästiin tutkimaan tietokantojen käyttöä lokalisoinnissa ja tutustumaan tietokantojen käyttöön Symbian-sovelluskehityksessä.

Koska insinöörityö tehtiin osana asiakasprojektia, ei salassapitovelvollisuuden vuoksi kaikkia vaiheita ja sovelluksen lähdekoodeja ole käsitelty tässä, vaan on tehty vastaavanlaisia esimerkkejä.

Asiasanat:

Symbian OS, Lokalisointi, DBMS, Tietokanta, C++

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Embedded Software

---

Author: Mikko Heikura
Title of thesis: Localization of S60 application
Supervisor: Jouni Kivirinta
Term and year of completion: Spring 2011 Number of pages: 36 + 5

The objective of this Bachelor's thesis was to implement a localization for the S60 application made by ResComi Oy. The purpose was to implement a localization both in Finnish and in English according to Symbian software development and also to design and implement a localization as a database based solution. Another was to make adding new languages to the application easier.

As a result of this thesis, we managed to implement the localization of the S60 application which in further development makes adding new languages to the application easier. With a database based solution it was also able to study the usage of databases in the localization and get to know the usage of databases in the Symbian software development.

Because this thesis was made as part of a customer project and because of the confidentiality every stage of the work process and source codes will not be handled in this document but similar examples are given.

Keywords:

Symbian OS, Localization, DBMS, Database, C++

SISÄLLYS

TERMIT JA LYHENTEET

API	Application Programming Interface. Ohjelmointirajapinta
DBMS	Database Management System. Tietokannan hallintajärjestelmä
DDL	Data Definition Language
DML	Data Modeling Language
IDE	Integrated Development Environment. Integroitu ohjelmistokehitysympäristö, ohjelmointiympäristö
J2ME	Java Platform, Micro Edition
SDK	Software Development Kit. Ohjelmiston kehittämisympäristö
SQL	Structured Query Language. Standardoitu kyselykieli
UI	User Interface. Käyttöliittymä

1 JOHDANTO

Insinöörityön aiheena on ”S60–sovelluksen lokalisointi”, joka tehtiin ResComi Oy:lle. ResComi Oy on tietoliikennealan ohjelmistoyritys, joka tuottaa ohjelmistosuunnittelupalveluja, tietojärjestelmiä ja ohjelmistotuotteita tietoliikennealan ja perinteisen teollisuuden yrityksille.

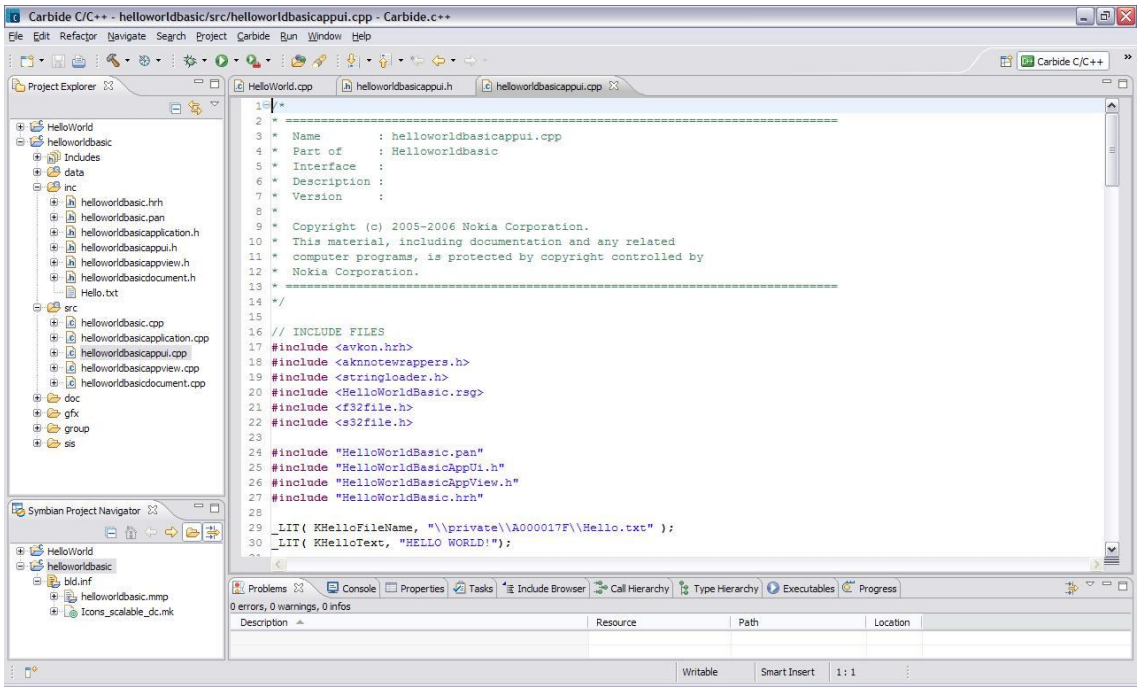
Insinöörityön tavoitteena oli tutkia, että onko lokalisointi nopeampi ja helpompi toteuttaa Symbianin virallisella tavalla toteutettuna vai tietokantapohjaisena ratkaisuna. Lokalisoinnin kohteena oli ResComi Oy:n tekemä S60-sovellus.

1.1 Symbian ja Carbide.c++-kehitysympäristö

Symbian Ltd. perustettiin vuonna 1998. Perustajina olivat Ericsson, Motorola, Nokia ja Psion. Symbianin historia ulottuu kuitenkin hieman kauemmas, vuoteen 1994, jolloin Psion rupesi kehittämään 32-bittisen käyttöjärjestelmän ydintä. Käyttöjärjestelmä sai nimekseen EPOC. Myöhemmin, vuonna 1998, uuden käyttöjärjestelmän nimen suunnittelussa vaihtoehtoina oli mm. ”Edgware” ja ”Everywhare”. Psion Softwaresta tuli kesäkuussa vuonna 1998 Symbian Ltd. ja 24.7.1998 käyttöjärjestelmä nimeltä ”Symbian OS” näki päivänvalon. Vuoden 2009 aikana Symbianista tuli säätiö ja käyttöjärjestelmä tuli saataville avoimena lähdekoodina ilman lisenssimaksuja. Kuitenkin jo vuoden 2010 aikana säätiö ajettiin alas eikä lähdekoodi enää ollut avoin. (5,6)

Carbide.c++ IDE on Nokian kehittämä ohjelmistokehitysympäristö, joka pohjautuu Eclipse–ohjelmistokehitysympäristöön ja Eclipse CDT Projectin C/C++-kehitystyökaluihin. Carbide.c++ tehtiin aikanaan korvaamaan Nokia CodeWarrior–ohjelmistokehitysympäristö. Carbide.c++ on saatavilla ilmaiseksi Windows–käyttöjärjestelmille. Carbide.c++-ohjelmistokehitysympäristöllä voi

tehdä Symbian C++-sovelluskehityksen lisäksi myös Qt–sovelluskehitystä. (4) Kuvassa 1 on perusnäkymä Carbide.c++ IDE:sta, jossa on projekti avoinna.



Kuva 1. Carbide.c++ IDE

Carbide.c++ tarjoaa myös valmiita projektipohjia, joiden avulla voidaan luoda projektirunkoja yleisimpien Symbian–ohjelmistotyyppien kehityksen pohjaksi. Nykyään myös Qt Designer on integroituna Carbide.c++-ohjelmistokehitysympäristöön. Carbide.c++ sisältää myös Code Scannerin, jonka avulla voidaan tunnistaa koodissa mahdollisesti esiintyviä ongelmia jo aikaisessa vaiheessa, ennen kuin niistä tulee liian työläitä tunnistaa ja poistaa. (4)

1.2 S60-ohjelmistoalusta

S60, joka aikaisemmin tunnettiin nimellä Series 60 Ui ja nykyään yksinkertaistetusti nimellä Symbian, on Nokian kehittämä ja omistama ohjelmistoalusta ja se pohjautuu Symbian OS–käyttöjärjestelmään. Series 60 Ui

–nimi muutettiin vuonna 2005 pelkistetympään nimeen S60. Vuonna 2010 S60:n nimi muutettiin jälleen ja nimeksi tuli Symbian. (7)

S60 on Nokian kehittämä älypuhelinten ohjelmistoalusta, joka on rakennettu Symbian OS–käyttöjärjestelmän päälle. Nokia on lisensoinut S60-ohjelmistoalustaa myös muille valmistajille. Muita valmistajia ovat olleet mm. Ericsson, Sony Ericsson ja Samsung.

Kuva 2. Nokia 7650.

Kuvassa 2 on Nokia 7650, Nokian ensimmäinen S60–ohjelmistoalustalla varustettu puhelin, joka julkaistiin vuonna 2001. Kyseisessä puhelimessa käyttöjärjestelmänä oli Symbian OS v6.1 ja ohjelmistoalustana Series 60 v0.9.

1.3 Sovelluskehitys

Symbian-sovelluskehitys, kuten monet muutkin sovelluskehitykset, voidaan tehdä vaihejakomallin avulla. Esimerkiksi vesiputousmallissa edetään vaihe kerrallaan. Vesiputousmallin mukaan mentäessä sovelluskehitys alkaa sovelluksen määrittelyllä. Kun määrittely on saatu tehtyä, ruvetaan suunnittelemaan sovellusta. Suunnittelun jälkeen vuorossa on sovelluksen varsinainen toteutus ja toteutuksen jälkeen suoritetaan sovelluksen testaus. Kun tehty sovellus on läpäissyt testauksen, jää jäljelle sovelluksen käyttöönotto.

1.3.1 Ohjelmointikieli

Koska Symbian OS –käyttöjärjestelmä on lähes kokonaan kirjoitettu C++-ohjelmointikielellä, S60-ohjelmistokehityksessä yleisesti ohjelmointikielenä on C++. Kuitenkin Symbian-ohjelmoinnissa on C++-kielen osalta joitain eroavaisuuksia. C++-kielen standardikirjastoja ei käytetä vaan Symbian OS:n kirjastot on suunniteltu mobiililaitteille tarkoituksena kuluttaa mahdollisimman vähän muistia. Koska Symbian OS:n käyttämät kirjastot ovat jossain määrin matalamman tason kirjastoja verrattuna standardin C++:n kirjastoihin, on ohjelmointi paikoin hankalaa ja hidasta. (1,6)

S60-sovelluksien kehitys onnistuu myös muilla ohjelmointikielillä, koska S60-ohjelmistoalustassa on tuki myös J2ME:lle ja Pythonille. Uusimmat S60-ohjelmistoalustat tukevat myös Qt-sovelluskehitystä. (4)

Tässä työssä sovelluskehitys tehtiin C++-ohjelmointikielellä, koska sovellus, johon lokalisointi tehtiin, oli natiivisovellus. Symbianin natiivisovellukset on toteutettu C++-ohjelmointikielellä.

1.3.2 Kehitysympäristö

Kehitysympäristönä toimi Nokian kehittämä Carbide.c++ IDE, joka pohjautuu avoimeen Eclipse–kehitysympäristöön. Carbide.c++ tehtiin aikanaan korvaamaan Nokia CodeWarrior. Carbide.c++ toimii Windows-käyttöjärjestelmissä. (4) Työtä tehdessä sovellusta käytettiin ja testattiin Nokia 6120–päätelaitteessa.

2 MÄÄRITELMÄ

Insinöörityössä otettiin selvää, onko tietokannan avulla toteutettu monikielisyyden tuki tehokkaampi kuin Symbianin suosittelema tapa. Tässä työssä monikielisyyden tukea lähdettiin laajentamaan lisäämällä sovellukseen suomen kielen lisäksi englannin kieli.

Symbianin suositteleman tavan ja tietokantapohjaisen toteutuksen välisiä eroja vertailtiin insinöörityötä tehdessä. Symbianin suosittelema tapa on yksinkertaisempi, koska ei tarvitse muokata kuin käytettävien kielten kielitiedostoja. Tämä olettaen, että ohjelmoija on huomioinut sovelluksen monikielisyyden eikä ole kovakoodannut merkkijonoja lähdekoodiin. Käännösten teettäminen esimerkiksi käännöstoimistolla onnistuu helposti, koska ei tarvitse kuin toimittaa yksi tekstitiedosto, mikä sisältää käännettävät tekstit. Tässä kuitenkin täytyy ottaa huomioon se, että tiedoston sisältämät tekstit täytyy kommentoida huolellisesti, että käännöksistä tulee varmasti asiatarkkoja eikä sanatarkkoja.

Tietokantapohjaisessa toteutuksessa taas joutuu tekemään ylimääräisen sovelluksen, jonka avulla luodaan tietokanta ja luetaan eri kielten merkkijonot tietokantaan. Tämä tietokanta täytyy sitten liittää varsinaisen lokalisoitavan sovelluksen mukaan ja tehdä sovellukseen tarvittavat koodimuutokset, että merkkijonojen haku tietokannasta onnistuu. Tietokantapohjaisessa ratkaisussa voidaan myös käännökset teetättää käännöstoimistolla toimittamalla tekstitiedosto, mikä sisältää käännettävät tekstit. Ongelmaksi muodostuu PC:llä käytettävien työkalujen puute, millä voisi helposti tarkastaa tietokannan rakenteen ja oikeellisuuden. Tätäkin varten joutuu käyttämään joko ylimääräistä sovellusta emulaattorissa, millä pystyy tarkastelemaan tietokannan rakennetta ja oikeellisuutta, tai sitten odottaa testausvaihetta ja katsoa, toimiiko tietokanta suunnitellusti.

2.1 Toiminnalliset vaatimukset

Lähtökohtana oli toteuttaa jo olemassa olevaan S60-sovellukseen lisätoiminnallisuus, monikielisyyden tuki. Olemassa olevaan sovellukseen ei saanut tulla toiminnallisia muutoksia. Lokalisointia tehdessä tämä ei haitannut, koska muutokset koskivat lähinnä vain merkkijonojen muokkausta sekä lokalisointiin liittyvien resurssitiedostojen muokkausta ja luontia. Sovellus oli ennen työn aloitusta toteutettu toimimaan suomenkielisenä.

2.2 Kielen valinta

Sovelluksessa käytettävän kielen valinta voidaan suorittaa joko päätelaitteen käyttöjärjestelmän kielen mukaan tai käyttäjän valinnan mukaan. Asennuspaketista riippuen sovelluksen asennusvaiheessa käyttäjää pyydetään valitsemaan haluttu kieli esitetyistä vaihtoehdoista tai asennusvaiheessa sovellus asentuu päätelaitteen käyttökielen mukaan, olettaen että sovelluksen kielivaihtoehdoista löytyy sama kielivaihtoehto, kuin on päätelaitteen käyttökieli. Muussa tapauksessa sovelluksen käyttökieleksi tulee ennaltamääritelty oletuskieli.

Tässä työssä käytettiin tapaa, missä sovelluksen kieleksi valitaan automaattisesti kieli, joka on puhelimen käyttökielenä. Sovelluksen kielet asentuivat käyttäjältä huomaamattomasti, eikä mitään vahvistuksia käyttäjältä vaadittu asennusvaiheessa. Kielet asentuivat sovelluksen mukana ns. hiljaisesti. Tämä tapa mahdollisti sovelluksessa kielen vaihtamisen ns. lennossa. Käytännössä päätelaitteen käyttökielen vaihtaminen edellytti myös päätelaitteen uudelleenkäynnistämistä, ennen kuin muutokset tulivat voimaan. Tämä johtui täysin päätelaitteen ominaisuuksista.

Sovelluksen kieli vaihtui automaattisesti siinä tapauksessa, että käyttäjä vaihtoi päätelaitteen käyttökieltä. Siinä tapauksessa, että puhelimen käyttökieli ei

vastaa sovelluksen kielivalikoimaa, käytetään ennalta määriteltyä oletuskieltä. Koska sovellukseen tehtiin suomen- ja englanninkieliset lokalisoinnit, oletuskieleksi valittiin englanti. Tämä tehtiin sen takia, että jos loppukäyttäjän päätelaitteessa käyttämä kieli on joku muu kuin suomen kieli, todennäköisesti loppukäyttäjä ymmärtää englannin kieltä paremmin kuin suomen kieltä.

3 TOIMINTAYMPÄRISTÖ

Toimintaympäristönä tässä insinöörityössä oli Microsoftin Windows-käyttöjärjestelmällä varustettu PC, johon asennettiin Symbian-ohjelmistokehityksessä tarvittavat ohjelmistot. Ensimmäisenä asennettiin sopiva ohjelmistokehitysympäristö, Carbide.c++. Carbide.c++ ei toimi Linuxilla, joten ei ollut muita vaihtoehtoja kuin Windows. Carbide.c++ IDE:n nykyiset versiot tarvitsevat toimiakseen PC:n, jossa on Microsoft Windows XP, Windows Vista tai Windows 7.

Ohjelmistokehitysympäristön lisäksi tarvitaan myös SDK, joka valitaan sen perusteella, mitkä ovat kohdelaitteet. Tässä insinöörityössä kohdelaitteina olivat Nokian S60-älypuhelimet, tarkemmin S60 3rd Edition –ohjelmistoalustaiset laitteet. Näille laitteille tarkoitettua ohjelmistokehitystä varten käytettiin S60 3rd Edition SDK for Symbian OS for C++, Maintenance Release –SDK:ta.

Sovelluksen testauksessa käytettiin sekä Epoc-emulaattoria että Nokia 6220 Classic –päätelaitetta, joka oli varustettu S60 3rd Edition, Feature Pack 2 –ohjelmistoalustalla. Nokia 6220:n käyttöjärjestelmänä on Symbian OS v9.3. Kuvassa 3 on S60 3rd Edition SDK:n mukana tuleva Epoc-emulaattori, jossa on käynnissä HelloWorld–esimerkkisovellus.

Kuva 3. Epoc-emulaattori.

Tietokantapohjaisen ratkaisun tarvitseman erillisen sovelluksen kehittämisessä käytettiin Carbide.c++-ohjelmistokehitysympäristön lisäksi S60 5th Edition SDK:ta. Tämän sovelluksen testaus suoritettiin pelkästään emulaattorin avulla. Emulaattorin kanssa testatessa pystyi samalla suorittamaan debuggausta, joka mm. helpotti tietokantaan menevän tiedon tarkastelua.

4 TOTEUTUS

Insinöörityössä lokalisointi tehtiin kahta tapaa käyttäen eli sekä Symbianin suosittelemalla tavalla että tietokantapohjaisella tavalla. Seuraavaksi on kerrottu insinöörityön toteutuksen vaiheista.

4.1 Kehitysympäristö

Kehitysympäristönä oli Carbide.c++ IDE, joka asennettiin Windows-käyttöjärjestelmällä varustettuun PC-tietokoneeseen. Carbide.c++ IDE:n lisäksi tarvitaan myös sovelluskehityksen kohteena olevan päätelaitteen mukainen SDK. SDK:ksi valittiin S60 3rd Edition SDK for Symbian OS for C++, Maintenance Edition –SDK, koska sovellus, johon lokalisointia lähdettiin tekemään, oli suunnattu Nokian S60 3rd Edition –ohjelmistoalustaisille päätelaitteille.

Ohjelmistokehitysympäristön ja SDK:n lisäksi Symbian-sovelluskehityksessä tarvitaan vielä ActivePerl. ActivePerl joudutaan asentamaan, koska scriptit, joita käytetään sovellusten kääntämisessä, on kirjoitettu Perlillä. ActivePerlin asennusta ei periaatteessa voi unohtaa, koska Carbide.c++:n asennuksen loppuvaiheessa asennusohjelma ilmoittaa, että tarvitaan ActivePerl ja ohjaa www-sivulle, josta voidaan ladata tarvittava versio ActivePerlistä.

4.2 Lokalisoinnin toteutus virallisella menetelmällä

Lähdettäessä tekemään sovelluksen lokalisointia oli sovelluksen lähdekoodista etsittävä kaikki kovakoodatut merkkijonot, ns. literaalit. Koska sovellus oli tehty yksikieliseksi, monet sovelluksessa käytettävät merkkijonot oli lähdekoodissa literaaleina ja osa .loc-tiedostossa. Lokalisointia tehtäessä nämä literaalit piti poistaa lähdekoodista ja korvata koodilla, jossa haetaan merkkijonot käytettävissä olevan kielen lokalisointitiedostoista. Näiden literaalien etsiminen

ja korvaaminen asianmukaisilla koodeilla vei kohtalaisen paljon aikaa, koska kyseessä oli kuitenkin tuhansia rivejä lähdekoodia, mitä piti käydä läpi. Samalla jouduttiin lokalisointitiedostoihin lisäämään sieltä puuttuvat tunnisteet ja merkkijonot, jotka oli aikaisemmin toteutettu literaalien avulla.

Insinöörityössä osakseni tuli myös alustavien englanninkielisten käännösten tekeminen. Tämä ei sinällään ollut ongelmallista, koska englannin kieli on ohjelmistotuotannossa pääasiallinen kieli, mitä käytetään erilaisten dokumenttien tekemisessä. Myös suuri osa kaikesta lähdemateriaalista on englanniksi, varsinkin Symbian-ohjelmistokehityksen kohdalla. Haasteellisinta käännösten tekemisessä oli saada käännöksistä mahdollisimman asiatarkkoja ja selkeitä.

Lokalisointia tehdessä eri kielten merkkijonot määritellään niille varatuissa lokalisointitiedostoissa. Koska sovellukseen tarvittiin sekä suomen- että englanninkieliset merkkijonot, käytettiin `.l01`- ja `.l09`–tiedostoja. Tiedostojen päätteiden numeroinnin perusteella tunnistetaan oikea kieli, 01 on brittienglanti ja 09 suomen kieli. (3)

Graafisen käyttöliittymän tarvitsemat resurssit määritellään `.rss`–tiedostossa. Näitä resursseja ovat mm. valikot ja valikoitten osat. Näissä resurssimäärittelyissä määritellään myös tunnisteet, joiden perusteella haetaan valikoitten osien tarvittavat merkkijonot lokalisointitiedostosta. Esimerkiksi valikon määrittely tapahtuu kuvan 4 osoittamalla tavalla:

```
RESOURCE MENU_PANE r_application_menu_pane
    {
    items =
        {
        MENU_ITEM
                {
                command = EAppMenu1; txt = qtn_appl_menu1;
                }
        }
    }
```

Kuva 4. Valikon määrittely .rss-tiedostossa.

Virallisella menetelmällä tehtynä lokalisointi tapahtuu siten, että määritellään sovelluksessa käytettävät merkkijonot `.l01`- ja `.l09`-tiedostoihin esimerkiksi kuvissa 5 ja 6 näkyvällä tavalla:

```
#define qtn_appl_menu1 "Localization"
#define qtn_example_command1 "Open"
#define qtn_example_command2 "Close"
```

Kuva 5. Englanninkielisten merkkijonojen määrittely .l01-tiedostossa.

```
#define qtn_appl_menu1 "Lokalisaatio"
#define qtn_example_command1 "Avaa"
#define qtn_example_command2 "Sulje"
```

Kuva 6. Suomenkielisten merkkijonojen määrittely .l09-tiedostossa.

Resurssit, joissa merkkijonoja käytetään, on määritelty `.rss`–tiedostossa, johon sisällytetään myös `.loc`–tiedosto, jossa voidaan myös määritellä resurssien tarvitsemat merkkijonot. Resurssitiedostossa on määritelty myös sovelluksessa näytettävien tekstien tunnisteet, joiden mukaan sitten merkkijonot haetaan tarvittaessa lokalisaatiotiedostoista.

Mutta koska tehdään monikielistä sovellusta, merkkijonojen määrittelyä ei tehdä `.loc`–tiedostossa, vaan omaan kielelle varattuun lokalisaatiotiedostoon, jonka tiedostopääte määräytyy kyseessä olevan kielen mukaan. Kyseiset

lokalisaatiotiedostot sisällytetään .loc–tiedoston alkuun kuvassa 7 näkyvällä tavalla.

```
#ifdef LANGUAGE_01  // Language code for UK English
    #include "..\data\Application.l01"

#elif defined LANGUAGE_09   // Language code for Finnish
    #include "..\data\Application.l09"
#endif
```

Kuva 7. Lokalisaatiotiedostojen sisällytys .loc-tiedostossa.

Monikielistä sovellusta tehdessä myös mahdollisten kuvien lokalisointi täytyy myös ottaa huomioon. Koska kuvissa voi olla myös tekstejä, on myös kuvat lokalisoitava. Jos sovelluksessa käytetään kuvia, jotka pitää muokata vastaamaan käytettäviä kieliä, on myös muistettava määritellä oikeat polut resurssitiedostoihin vastaamaan käytettäviä kieliä. Vaikka tässä työssä ei tarvinnut lokalisoida kuvia, oli asia kuitenkin otettava huomioon.

Laajemmassa lokalisoinnissa tulee ottaa myös huomioon mahdolliset kulttuurierot esimerkiksi kuvissa olevien symbolien merkitysten takia. Jollain symbolilla voi olla hieman erilainen tarkoitus eri kulttuuriympäristössä. Myös eri kielten lukusuunta vaihtelee maasta riippuen. Yleisesti käytetään vasemmalta oikealle luettavaa tekstiä, mutta esim. arabialainen kirjaimisto kirjoitetaan oikealta vasemmalle. Insinöörityössä ei tarvinnut perehtyä tämän tyyliseen lokalisointiin, mutta oli hyvä ottaa ajatustasolla huomioon erilaisia mahdollisuuksia.

4.2.1 Käytettävien kielten määritys

Sovelluksessa käytettävät kielet määritellään .mmp–tiedostossa. Mmp–tiedosto on projektin määrittelytiedosto, missä tarkennetaan projektin ominaisuudet, mm. sovelluksessa käytettävät kielet. Koska sovelluksessa tarvittiin sekä suomen että englannin kieli, määriteltiin ne kuvan 8 esimerkin mukaisesti.

```
LANG 01 09
```

Kuva 8. Käytettävien kielten määritys .mmp-tiedostossa.

4.2.2 Lokalisointi ja asennus kohteeseen

Kohteeseen asennuksessa käytettiin sovelluksen päivityspakettia, minkä rakentamiseen käytettyä .pkg-tiedostoa muokattiin niiltä osin, missä määritellään sovelluksessa käytettävät kielet.

Sovellukseen asennettavat kielet, esimerkiksi tässä tapauksessa englanti ja suomi, määritellään .pkg–tiedostossa kuvassa 9 näkyvällä tavalla.

```
;Languages
&EN, FI
```

Kuva 9. Asennettavien kielten määritys .pkg-tiedostossa

Asennettavien kielten resurssien tiedostopolut pitää myös määrittää .pkg–tiedostossa, kuten kuvan 10 esimerkissä on tehty.

```
"C:\Symbian\9.1\S60_3rd_MR\Epoc32\release\winscw\udeb\z\resource\apps\Application.r01
-"e:\resource\apps\Application.r01"

"C:\Symbian\9.1\S60_3rd_MR\Epoc32\release\winscw\udeb\z\resource\apps\Application.r09
-"e:\resource\apps\Application.r09"
```

Kuva 10. Asennettavien kielten resurssien tiedostopolkujen määrittäminen .pkg-tiedostossa.

Merkkijonojen hakeminen esimerkiksi informaation näyttämiseen päätelaitteen näytöllä tapahtuu kuvan 11 osoittamalla tavalla.

```
HBufC* noteText = StringLoader::LoadLC(R_COMMAND1_TEXT);
CAknInformationNote* infoNote;
infoNote = new (ELeave) CAknInformationNote;
infoNote->ExecuteLD(*noteText);
CleanupStack::PopAndDestroy(noteText);
```

Kuva 11. Merkkijonojen hakeminen.

`StringLoader`–luokan `LoadLC`–funktiolla haetaan resursseista haluttu merkkijono deskriptoriin jonka jälkeen se annetaan `CAknInformationNote`–luokan `ExecuteLD`–funktiolle parametrina, joka näyttää informaation päätelaitteen näytöllä. `StringLoader`–luokan avulla voidaan hakea halutut merkkijonot resurssitiedoista deskriptoriin, jonka jälkeen merkkijonoa voidaan käyttää tarvittavissa tilanteissa, kuten edellä ilmoituksen näyttämiseen.

4.3 Lokalisoinnin toteutus tietokannalla

Työssä tehtiin lokalisointi myös tietokantapohjaisena ratkaisuna, jossa haluttujen kielten sovelluksessa käytettävät merkkijonot ja merkkijonojen tunnisteet tallennettiin tietokantaan. Tarkoituksena oli tutkia, onko tietokantojen käyttö lokalisoinnissa helpompaa ja nopeampaa kuin tehdessä lokalisointi virallisella menetelmällä.

Tietokantapohjaisen ratkaisun tekemisessä lähdettiin liikkeelle suunnittelemalla, kuinka saadaan eri kielten merkkijonot tietokantaan. Tähän tarkoitukseen tehtiin sovellus, jonka avulla luodaan tietokanta ja tietokannan taulu. Tietokannan luomisen jälkeen sovellus lukee olemassa olevista tekstitiedostoista tunnisteet sekä merkkijonot ja tallentaa nämä luotuun tietokantaan. Tietokannan luomiseen tarkoitettu sovellus jouduttiin tekemään, koska PC:lle ei löytynyt työkaluja Symbianin tietokantojen luomiseen tai muokkaamiseen. Tämän sovelluksen tekovaiheessa löytyi yksi sovellus, jonka olisi pitänyt olla kykeneväinen käsittelemään Symbianin tietokantoja, mutta kokeilu osoitti toisin, joten siitä ei ollut mitään hyötyä tässä työssä.

4.3.1 Symbian OS DBMS ja tietokantojen rakenne

Symbianissa tietokantojen käsittelyyn on olemassa Symbian OS DBMS –ohjelmointirajapinta. Symbian OS DBMS tukee normaaleja toiminnallisuuksia, kuten lisäämistä, hakua, noutamista, päivittämistä ja poistamista. Myös SQL:n, DDL:n ja DML:n peruslauseiden käsittely on tuettu. Kehittyneempiä toimintoja, kuten SQL liitoksia (joins) ja herätteitä (trigger events), ei ole tuettu Symbian OS DBMS:ssä. (2)

Tietokantojen rakenne koostuu tauluista, sarakkeista ja riveistä. Tietokanta itsessään on säiliö tauluille ja taulut ovat säiliöitä riveille. Kentät taas ovat rivien rakenneosia. (2)

DBMS API:n avulla voidaan käsitellä relaatiotietokantoja S60-päätelaitteilla. Tietokannan luomisessa voidaan käyttää kahta eri luokkaa: `RDbStoreDatabase` ja `RDbNamedDatabase`. Tietokanta luodaan tiedostoon, riippumatta kumpaa luokkaa käytetään. Molemmissa tapauksissa tietokanta luodaan eksklusiivisesti. Tietokanta avataan `RFs`:n file-server-sessiolla. `RDbStoreDatabase`–luokan avulla voidaan tietokanta luoda ja avata eksklusiivisesti, eli silloin tietokantaan eivät muut pääse käsiksi ja operaatiot suoritetaan suoraan tiedostoon. Tässä insinöörityössä tietokannan luomiseen käytettiin `RDbStoreDatabase`–luokkaa. (2)

Tietokannan taulujen määrittelyssä pitää ottaa huomioon kolme ohjelmointirajapinnan avainasiaa: sarake, sarakeryhmä ja indeksiavain. DBMS:ssä taulujen nimien täytyy olla yksilöllisiä ja sarakkeiden nimet täytyy olla yksilöllisiä sen taulun sisällä, mihin ne kuuluvat. Sarakeryhmät määrittävät taulun tietokannassa siten, että jokaisella sarakkeella on atribuuttinsa, kuten nimi ja datatyyppi. Jos datatyyppinä on teksti tai binääri, atribuuttina on myös maksimipituus. Indeksiavaimien avulla tietokantaan voidaan määrittää avainsarake, jonka perusteella voidaan tietokannasta hakea tietoa nopeasti. (2)

Jos haluaa tehdä laajempia tietokantakokonaisuuksia, on tietokantaan mahdollista tallentaa useampia tauluja eri tietoja varten. Tietokantaa, jossa on useampia tauluja, kutsutaan relaatiotietokannaksi. Sen takia ei ole välttämättä tarpeen luoda useampia erillisiä tietokantoja. Tietokannan sisällä olevat taulut voidaan linkittää toisiinsa, eli määritellä, kuinka ne liittyvät toisiinsa. Insinöörityössä ei kuitenkaan perehdytty relaatiotietokantoihin syvemmin, koska sellaiselle ei ollut tarvetta tässä tapauksessa.

4.3.2 Tietokannan luonti erillisellä sovelluksella

Tietokannan luomista varten tehtiin sovellus, jolla oli mahdollista luoda tietokanta joko emulaattorilla tai S60-päätelaitteella. Sovellus jouduttiin tekemään sen takia, että DBMS–ohjelmointirajapinta on suljettu eikä sen vuoksi ole erillisiä työkaluja pc:lle, millä voisi Symbianin tietokantoja muokata.

Tietokannan luomista varten täytyy sovelluksessa ensin yhdistää client fileserveriin `RFs`–luokan `Connect()`–funktiolla. Tämän jälkeen haetaan sovelluksen nimi ja polku `Application()->AppFullName()`–funktiolla. Kun tietokantaa lähdetään luomaan, pitää ensin varmistaa, että tietokanta ei ole ennestään avoinna eikä `CFileStore`-olio ole luotuna. Tietokannan tarvitsema tiedosto luodaan sovelluksessa `CPermanentFileStore::ReplaceL()`–funktiolla, johon annetaan parametreina `RFs`, tietokannan tiedostopolku ja tiedostomoodi. Tiedostomoodilla määritellään, voidaanko tietokantaan kirjoittaa ja lukea. Tässä käytettiin sekä kirjoitus- että lukuoikeuksia, `EFileRead` ja `EFileWrite`. Tämän jälkeen asetetaan file storelle tyyppi `SetTypeL`–funktion avulla. Funktiolle annetaan parametreina kyseisen file storen tunniste, joka haetaan `Layout`-funktion avulla. Tämän jälkeen luodaan tietovirta-olio, `TStreamId`, joka sitten annetaan parametrina `SetRootL`–funktiolle, joka asettaa kyseisen tietovirran tietokannan tietovirtajuureksi. Lopuksi tallennetaan tehdyt muutokset `CommitL`-funktion avulla.

Kun tietokanta on luotu, tietokantaan pitää luoda taulu, johon voidaan tallentaa tietoa. Taulun luonti tapahtuu `RDbStoreDatabase`–luokan `CreateTable`-funktiossa, johon annetaan parametreinä luotavan taulun nimi sekä `CDbColSet`-olion osoitin joka sisältää sarakkeiden nimet jotka halutaan luoda tauluun. Sarakkeiden nimet on lisätty `CDbColSet`–luokasta tehtyyn olioon `AddL`–funktiolla. Luotuja sarakkeita ovat ”Id”, ”01” ja ”09”, kuten kuvassa 12 on esitetty.

Taulu	Languages	Merkkijonojen tiedot
Sarake	Id	Merkkijonon tunniste
Sarake	01	Englanninkielinen merkkijono
Sarake	09	Suomenkielinen merkkijono

Kuva 12. Tietokannan osat luotavassa tietokannassa.

Taulun luonnin jälkeen luodaan indeksi. Ensin määritellään indeksissä käytettävä avainsarake `TDbKeyCol`–luokan avulla. Avainsarakkeena toimii ”Id”- sarake, koska tietokannasta on tarkoitus hakea tietoa merkkijonojen tunnisteiden perusteella. Tämän jälkeen `CDbKey`–olion `AddL`-funktiolla laitettiin edellä määrtitelty sarake `CDbKey`–olioon. Indeksin luonti tapahtuu `RDbStoreDatabase`–luokan `CreateIndex`–funktiolla, johon annetaan parametreina indeksin nimi, taulun nimi ja `CDbKey`–olion osoitin.

Indeksiavaimien luominen nopeuttaa tiedon hakemista tietokannasta. Jos indeksiavaimia ei luoda, silloin tieto haetaan järjestyksessä tietokannasta. Tämä voi hidastaa tietokannan käyttöä varsinkin suurempia tietokantoja käytettäessä.

4.3.3 Tekstitiedostojen tietojen luku tietokantaan

Tietokannan luomisen jälkeen piti saada luettua tiedot tekstitiedostoista tietokantaan. Tämä toteutettiin siten, että sovellus hakee käytettävät tekstitiedostot määritellystä paikasta, jonka jälkeen tiedostojen sisältö luetaan rivi kerrallaan. Tekstien lukeminen rivi kerrallaan toteutettiin `CLineReader`-

luokan avulla. CLineReader–luokka tarjoaa toiminnallisuuden, missä luettu rivi tallennetaan 8-bittiseen modifioitavaan osoitindeskriptoriin, TPtr8:aan. Rivit luetaan tekstitiedostosta CLineReader–luokan ReadLineL–funktion avulla. Rivien lukeminen tiedostosta loppuu, kun on saavutettu tiedoston loppu.

Kun rivi on onnistuneesti luettu, se tallennetaan tietokantaan sille kuuluvalle paikalle. Jos kyseessä on merkkijonon tunniste, se tallennetaan sille varattuun ”Id”–sarakkeen sisältämään kenttään. Merkkijonon ollessa kyseessä tallennetaan se joko ”01”- tai ”09”–kenttään riippuen, onko luettavissa oleva tekstitiedosto englannin- vai suomenkielisiä merkkijonoja sisältävä tiedosto. Kuten kuvassa 12 näytettiin, sarakkeen ”01” kenttään tallennetaan englanninkieliset merkkijonot ja sarakkeen ”09” kenttään tallennetaan suomenkieliset merkkijonot. Molempien kielten merkkijonot tallennetaan aina niitä vastaavien tunnisteiden riville.

Tietoja lähdettiin tämän jälkeen tallentamaan tietokantaan siten, että avataan ensin taulu RDbTable–luokan Open–funktiolla, jonka jälkeen asetetaan tietokannan sarakkeet CDbColSet–luokasta luotuun olioon RDbTable–luokan ColSetL–funktion avulla. Tämän tietokantaan asetetaan uusi rivi InsertL-funktion avulla, jonka jälkeen tallennetaan tiedot kyseiselle riville halutun sarakkeen kohtaan. Tämä tapahtuu SetColL–funktion avulla, johon annetaan parametreina sarakkeen numero ja haluttu tieto. Sarakkeen numeron saa CDbColSet–luokan ColNo–funktion avulla, kun funktiolle antaa parametrina halutun sarakkeen nimen. Kun tiedot on tallennettu kenttiin, viimeistellään muutokset RDbTable–luokan PutL–funktion avulla ja suljetaan taulu luokan Close–funktion avulla.

Tekstitiedostoja tässä työssä käytettiin kolmea erilaista. Tunnisteille sekä englannin- että suomenkielisille merkkijonoille oli jokaiselle oma tekstitiedosto. Tässä piti ottaa huomioon, että tunnisteita vastaavat merkkijonot olivat täsmälleen samalla rivillä omissa tiedostoissaan kuin tunnisteet. Jos näin ei ollut, menivät merkkijonot vääriin paikkoihin tietokannassa. Jos kyseessä olisi

laajempi sovelluskokonaisuus, ei tällainen ratkaisu olisi välttämättä kovin toimiva, mutta näin pienessä sovelluskokonaisuudessa ei asialla ollut niin suurta merkitystä.

4.3.4 Tietokannan käyttö sovelluksessa

Kun tietokanta on luotu ja tarvittavat tiedot sinne tallennettu, sitä voidaan ruveta käyttämään siinä sovelluksessa, missä on tarkoitus käyttää tietokantaa lokalisoinnissa. Tietokanta kopioidaan sovelluksen mukaan päätelaitteeseen. Kopionti voidaan suorittaa joko manuaalisesti sovelluksen asentamisen jälkeen tai lisäämällä se asennuspakettiin mukaan. Asennuspaketin mukaan tietokannan lisäys tapahtuu muokkaamalla `.pkg`–tiedostoa siten, että siellä määritellään tiedostopolku, mistä tietokanta löytyy, sekä tiedostopolku, minne tietokanta kopioidaan päätelaitteessa asennusvaiheessa. Sovelluksessa pitää olla tietokannan sijainnin tiedostopolku määriteltynä, että sovelluksessa voidaan tietokantaa käyttää.

Sovelluksessa käytettävien merkkijonojen hakeminen tietokannasta tapahtuu `RDbTable`n–luokan `SeekL`–funktion avulla. Tälle funktiolle annetaan parametrina halutun kentän nimi, tässä tapauksessa tunnisteen nimi. Tunnisteen perusteella haetaan tietokannasta tunnistetta vastaava rivi. Sen jälkeen tältä riviltä haetaan halutun sarakkeen sisältämä tieto, eli joko suomen- tai englanninkielinen merkkijono. Halutun sarakkeen numero haetaan `CDbColSet`–luokan `ColNo`–funktiolla, jolle annetaan parametrinä sarakkeen nimi. Rivin ja sarakkeen numeron perusteella löydetään oikea kenttä, missä haluttu tieto sijaitsee. Haluttu rivi avataan `RdbColStream`–luokan `OpenL`-funktion avulla, johon annetaan parametreina `RDbTable` ja halutun sarakkeen numero. Tiedon hakeminen tapahtuu `RdbColReadStream`–luokan `ReadL`-funktion avulla. Tämä tieto laitetaan deskriptoriin, jonka jälkeen haettua merkkijonoa voidaan käyttää halutulla tavalla, esimerkiksi ilmoituksen näyttämisessä päätelaitteen näytöllä.

4.3.5 Käytettävien kielten määritys

Käytettävissä olevat kielet voidaan määrittää esimerkiksi joko samaan tietokantaan, minne on tallennettu myös käytettävissä olevien kielten merkkijonot. Voidaan käyttää vaihtoehtoisesti erillistä tekstitiedostoa, johon on tallennettu käytettävien kielten tunnisteet. Tämän jälkeen sovellukseen voidaan lisätä esimerkiksi asetusvalikko, jossa käytettävän kielen voi vaihtaa haluamakseen eikä näin ollen sovelluksessa käytettäisi samaa kieltä kuin on päätelaitteen käyttökielenä.

5 TESTAUS

Testaaminen on olennainen osa ohjelmistokehitystä. Kehitettävän sovelluksen testausta olisikin hyvä suorittaa jo projektin alkumetreillä, koska silloin mahdolliset virheet sovelluksessa ja sen toiminnassa saadaan korjattua kohtalaisen vähällä vaivalla. Jos sovelluksen testaaminen suoritetaan vasta sovelluskehityksen loppupäässä, ongelmakohtia on luultavasti paljon vaikeampi lähteä korjaamaan. Mitä suurempi kokonaisuus on kyseessä, sitä tärkeämpään rooliin testaus nousee. Suuremmassa sovelluskokonaisuudessa pienetkin ongelmien korjaukset voivat vaikuttaa kokonaisuuteen epätoivotulla tavalla. Sovelluksen kehityksen varhaisessa vaiheessa suoritettu testaus, virheiden löytäminen ja niiden korjaaminen voi säästää huomattavan määrän työtunteja ja näin ollen myös projektiin kuluvaa rahan määrää. Hyvin usein ohjelmoijatkin testaavat kehittämäänsä sovellusta, mutta yleensä vain niiltä osin, mitä ovat itse tehneet, eivätkä silloin välttämättä pysty näkemään kokonaiskuvaa sovelluksen toiminnasta ja mahdollisista ongelmista. Suuremmissa projekteissa olisikin hyvä olla ohjelmoijien lisäksi myös testaajat. Tässä insinöörityössä testaus oli oleellista, niin kuin ohjelmistokehityksessä yleensä, mutta ei niin laajassa mittakaavassa työn luonteen vuoksi.

Insinöörityössä testausta suoritettiin lähes koko ajan työn edetessä. Tämä säästi aikaa monessakin tapauksessa, koska virheet huomattiin ajoissa, eikä niiden etsimiseen ja korjaamiseen tarvinnut kovin pitkää aikaa.

Symbianin suositteleman lokalisointitavan testaus suoritettiin sekä SDK:n mukana olevalla emulaattorilla että varsinaisessa S60–päätelaitteessa. Testausvaiheessa huomattiin, että sovellus ei aina käyttäydy samalla tavalla päätelaitteessa kuin emulaattorissa ja tämä tuottikin aika ajoin joitain pieniä ongelmatilanteita.

Sovelluksen testauksessa oli tarkoituksena huomata mahdolliset kirjoitusvirheet näytettävien tekstien osalta, niin suomen- kuin englanninkielisissä

toteutuksissa. Testausvaiheessa testattiin myös, ovatko kaikki resurssit ja tekstit mukana käännöksen jälkeen. Lokalisoinnin testauksessa ei tarvinnut keskittyä muuhun kuin edellä mainittuihin asioihin, koska sovelluksen toiminnallisuuteen ei tehty mitään muutoksia.

Tietokantapohjaisen lokalisoinnin ratkaisun testaaminen suoritettiin pelkästään S60 5th Edition SDK:n emulaattorilla. Sovellus testattiin pelkästään emulaattorissa, koska kyseistä sovellusta ei ole välttämätöntä käyttää päätelaitteessa, vaan tietokannan luomisen voi suorittaa ajamalla sovellus emulaattorissa ja kopioimalla luotu tietokanta sen sovelluksen mukaan, missä sitä mahdollisesti käytetään. Sovelluksen testauksessa piti tarkastella, toimiiko sovellus odotetulla tavalla. Tietokantaan tallennuksen ja tietokannasta hakemisen toimiminen oli sovelluksen suunnitellun toiminnan ydinasioita. Myös tekstitiedostojen sisällön lukemisen piti tapahtua oikein. Tietokannan sisältämän tiedon oikeellisuuden, tietokantaan tallennuksen ja hakemisen pystyi todentamaan siinä vaiheessa, kun sovellus käytti tietokannasta haettuja merkkijonoja. Jos merkkijonot näytöllä olivat vääriä tai niitä ei ollut ollenkaan, oli syytä olettaa, että jotain oli epäonnistunut. Debuggauksen avulla pystyttiin selvittämään tällaiset ongelmat, koska siinä nähtiin, mitä tietokantaan ollaan tallentamassa ja mitä tietokannasta haetaan. Testauksessa ei pystytty havaitsemaan, onko tietokannan käyttö lokalisoinnissa nopeampi tai hitaampi verrattuna Symbianin suosittelemaan tapaan tehdä lokalisointi.

6 JATKOKEHITYSMAHDOLLISUUDET

Koska tietokantapohjaisen lokalisointiratkaisuun tehty sovellus oli vain merkkijonojen tietokantaan tallennukseen tarkoitettu demosovellus, ei siihen tehty kovinkaan monimutkaista toiminnallisuutta, siksi siihen voitaisiin toteuttaa toiminnallisuus, jonka avulla voitaisiin lukea tarvittavat tunnisteet ja merkkijonot jo suoraan varsinaisista lokalisointitiedostoista. Myös automaattinen lokalisointitiedostojen etsiminen ja niiden sisältämän tiedon lukeminen tietokantaan voitaisiin toteuttaa sovellukseen.

Insinöörityön teon loppuvaiheessa Nokia rupesi julkaisemaan Symbianin lähdekoodeja, joten tämä ehkä mahdollistaisi erillisen PC-sovelluksen kehittämisen Symbianin tietokantojen käsittelyyn. Mutta koska Symbianin elinkaari alkaa olla näillä näkymin lopuillaan ja kehitystyö loppuu luultavasti parin vuoden kuluessa, en näe juurikaan syytä tämän asian panostamiseen. Näin myös sen takia, että Symbian-sovelluskehityksessä Qt on parempi vaihtoehto UI:n tekemiseen. Qt:ssa myös lokalisointi on toteutettu paremmin kuin Symbianin natiivisovelluksissa.

7 YHTEENVETO

Insinöörityön teon aikana tutustuttiin Symbian-sovelluskehitykseen ja erityisesti lokalisaation toteuttamiseen. Insinöörityössä pääsi syventävästi tutustumaan sekä Symbian-sovelluskehityksen prosessiin alusta loppuun että varsinaiseen ohjelmointiin. Insinöörityön tuloksena saatiin S60-sovellukseen toimiva lokalisointi, joka mahdollisti uusien kielien helpomman lisäämisen sovellukseen. Tuloksena saatiin myös sovellus, jonka avulla voidaan luoda tietokanta ja tallentaa sinne merkkijonoja tietokantapohjasista lokalisointiratkaisua varten.

Tutkittu tietokantapohjainen ratkaisu osoittautui viralliseen menetelmään verrattuna epäkäytännöllisemmäksi tavaksi toteuttaa lokalisointi eikä sen vuoksi nähty järkeväksi toteuttaa lokalisointia tietokantapohjaisena. Lisäksi tietokantapohjainen ratkaisu on ehtinyt myös vanhenemaan, koska Symbian-sovelluskehitys on nykyään Qt-perustaista ja siinä lokalisointi on toteutettu toisin. Kokonaisuutenakin Symbianin elinkaari alkaa myös olla lopuillaan, koska sen kehitystyö lopetetaan vaiheittain lähitulevaisuudessa Nokian siirtyessä Microsoftin Windows Phone -käyttöjärjestelmään.

Insinöörityötä tehdessä tulivat myös tietokantojen perusrakenteet ja perustoiminnot tutuiksi, kuten tietokantojen luonti, tietojen tallennus tietokantaan ja tietojen haku tietokannasta. Insinöörityön tekemisessä pääsin perehtymään paremmin sovelluskehitysprosessiin sekä Symbian-ohjelmointiin ja sen mukana tuomiin haasteisiin.

Tässä insinöörityössä sovelluskehityksessä käytetyt työkalut käyttöjärjestelmää lukuunottamatta olivat ilmaisia ja saatavilla www.forum.nokia.com –sivustolta.

LÄHDELUETTELO

1. Edwards, L., Barker, R. & the Staff of EMCC Software Ltd. 2004. Developing Series 60 Applications. Boston: Addison-Wesley.

2. Nokia. 2006. S60 Platform: Using DBMS APIs. http://www.forum.nokia.com/info/sw.nokia.com/id/e0a66f34-092a-4a52-8003-6bbc3aa83c8f/S60_Platform_Using_DBMS_APIs_v2_0_en.pdf.html

3. Nokia. 2010. Handy table for TLanguage vs. PKG language translation. http://wiki.forum.nokia.com/index.php/TLanguage_enumeration

4. Nokia. 2011. Symbian C++. http://www.forum.nokia.com/Develop/Other_Technologies/Symbian_C++/

5. The Register. 2010. Symbian, The Secret History: Dark Star. Hakupäivä 10.3.2011. http://www.theregister.co.uk/2010/11/23/symbian_history_part_one_dark_star/

6. Wikipedia. 2011. SymbianOS. Hakupäivä 22.3.2011 http://fi.wikipedia.org/wiki/Symbian_OS

7. Wikipedia. 2011. S60. Hakupäivä 22.3.2011 http://fi.wikipedia.org/wiki/S60

LIITTEET

LIITE A Kielien enumeraatiot ja PKG-tunnisteet

LIITE A Kielien enumeraatiot ja PKG-tunnisteet

Kieli	Enum arvo	Enum	PKG
Test	ELangTest	0	NA
English (UK)	ELangEnglish	1	EN
French	ELangFrench	2	FR
German	ELangGerman	3	GE
Spanish	ELangSpanish	4	SP
Italian	ELangItalian	5	IT
Swedish	ELangSwedish	6	SW
Danish	ELangDanish	7	DA
Norwegian	ELangNorwegian	8	NO
Finnish	ELangFinnish	9	FI
English (American)	ELangAmerican	10	AM
Swiss French	ELangSwissFrench	11	SF
Swiss German	ELangSwissGerman	12	SG
Portuguese	ELangPortuguese	13	PO
Turkish	ELangTurkish	14	TU
Icelandic	ELangIcelandic	15	IC
Russian	ELangRussian	16	RU
Hungarian	ELangHungarian	17	HU
Dutch	ELangDutch	18	DU
Belgian Flemish	ELangBelgianFlemish	19	BL
English (Australian)	ELangAustralian	20	AU
Belgian French	ELangBelgianFrench	21	BF
German (Austrian)	ELangAustrian	22	AS
New Zealand English	ELangNewZealand	23	NZ
French (International)	ELangInternationalFrench	24	IF
Czech	ELangCzech	25	CS
Slovak	ELangSlovak	26	SK
Polish	ELangPolish	27	PL
Slovenian	ELangSlovenian	28	SL

Chinese (Taiwan)	ELangTaiwanChinese	29	TC
Chinese (Hong Kong)	ELangHongKongChinese	30	HK
Chinese (People's Republic of China)	ELangPrcChinese	31	ZH
Japanese	ELangJapanese	32	JA
Thai	ELangThai	33	TH
Afrikaans	ELangAfrikaans	34	AF
Albanian	ELangAlbanian	35	SQ
Amharic	ELangAmharic	36	AH
Arabic	ELangArabic	37	AR
Armenian	ELangArmenian	38	HY
Tagalog	ELangTagalog	39	TL
Belarussian	ELangBelarussian	40	BE
Bengali	ELangBengali	41	BN
Bulgarian	ELangBulgarian	42	BG
Burmese	ELangBurmese	43	MY
Catalan	ELangCatalan	44	CA
Croatian	ELangCroatian	45	HR
English (Canadian)	ELangCanadianEnglish	46	CE
English (International)	ELangInternationalEnglish	47	IE
English (South African)	ELangSouthAfricanEnglish	48	SA
Estonian	ELangEstonian	49	ET
Farsi	ELangFarsi	50	FA
French (Canadian)	ELangCanadianFrench	51	N/A
Gaelic	ELangScotsGaelic	52	GD
Georgian	ELangGeorgian	53	KA
Greek	ELangGreek	54	EL
Greek (Cyprus)	ELangCyprusGreek	55	EL?
Gujarati	ELangGujarati	56	GU
Hebrew	ELangHebrew	57	HE
Hindi	ELangHindi	58	HI
Indonesian	ELangIndonesian	59	IN
Irish	ELangIrish	60	GA

Swiss Italian	ELangSwissItalian	61	SZ
Kannada	ELangKannada	62	KN
Kazakh	ELangKazakh	63	KK
Khmer	ELangKhmer	64	KM
Korean	ELangKorean	65	KO
Laothian	ELangLao	66	LO
Latvian	ELangLatvian	67	LV
Lithuanian	ELangLithuanian	68	LT
Macedonian	ELangMacedonian	69	MK
Malay	ELangMalay	70	MS
Malayalam	ELangMalayalam	71	ML
Marathi	ELangMarathi	72	MR
Moldovian	ELangMoldavian	73	MO
Mongolian	ELangMongolian	74	MN
Norwegian Nynorsk	ELangNorwegianNynorsk	75	NN
Brazilian Portuguese	ELangBrazilianPortuguese	76	BP
Punjabi	ELangPunjabi	77	PA
Romanian	ELangRomanian	78	RO
Serbian	ELangSerbian	79	SR
Sinhalese	ELangSinhalese	80	SI
Somali	ELangSomali	81	SO
Spanish (International)	ELangInternationalSpanish	82	OS
Latin American Spanish	ELangLatinAmericanSpanish	83	LS
Swahili	ELangSwahili	84	SH
Finland Swedish	ELangFinlandSwedish	85	FS
Reserved for future use	ELangReserved1	86	NA
Tamil	ELangTamil	87	TA
Telugu	ELangTelugu	88	TE
Tibetan	ELangTibetan	89	BO
Tigrinya	ELangTigrinya	90	TI
Cyprus Turkish	ELangCyprusTurkish	91	CT
Turkmen	ELangTurkmen	92	TK
Ukrainian	ELangUkrainian	93	UK

Urdu	ELangUrdu	94	UR
Reserved for future use	ELangReserved2	95	NA
Vietnamese	ELangVietnamese	96	VI
Welsh	ELangWelsh	97	CY
Zulu	ELangZulu	98	ZU
Other (deprecated)	ELangOther	99	NA

English with terms as used by the device manufacturer, if this needs to be distinct from the English used by the UI vendor

	ELangManufacturerEnglish	100	ME

South Sotho, a language of Lesotho also called Sesotho (SIL code sot)

	ELangSouthSotho	101	ST

Basque	ELangBasque	102	BA
Galician	ELangGalician	103	GL

English as appropriate for use in Asia-Pacific regions

	ELangEnglish_Apac	129	EA

English as appropriate for use in Taiwan

	ELangEnglish_Taiwan	157	YW

English as appropriate for use in Hong Kong

	ELangEnglish_HongKong	158	YH

English as appropriate for use in the People's Republic of China

	ELangEnglish_Prc	159	YP

English as appropriate for use in Japan

	ELangEnglish_Japan	160	YJ

English as appropriate for use in Thailand

ELangEnglish_Thailand	161	YT

Malay as appropriate for use in Asia-Pacific regions

ELangMalay_Apac	326	MA