UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jernej Jerebic

PREVAJALNIK ZA GENERIRANJE ENOTNEGA UPORABNIŠKEGA VMESNIKA

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Boštjan Slivnik

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01574/2009
Datum: 01.09.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JERNEJ JEREBIC**

Naslov: **PREVAJALNIK ZA GENERIRANJE ENOTNEGA UPORABNIŠKEGA VMESNIKA**

A COMPILER FOR GENERATION OF A UNIFIED USER INTERFACE

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Pri sodobnem rokovanju z elektronskimi napravami si pogosto želimo upravljati te naprave preko enega skupnega uporabniškega vmesnika. Sestavite sistem za enostaven opis uporabniškega vmesnika, ki omogoča uporabo velikega števila zelo različnih naprav. Napišite prevajalnik, ki prevede tak opis uporabniškega vmesnika v izvorno kodo v programskem jeziku C++. Celoten sistem opisa uporabniškega vmesnika in prevajalnik naj omogočata enostavno dodajanje opisov novih naprav z minimalnim uvajanjem sprememb v obstoječo izvorno kodo uporabniškega vmesnika.

Mentor:
B. Slivnik
doc. dr. Boštjan Slivnik

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Franc Solina

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Jernej Jerebic,

z vpisno številko 63020066,

sem avtor/-ica diplomskega dela z naslovom:

PREVAJALNIK ZA GENERIRANJE ENOTNEGA UPORABNIŠKEGA VMESNIKA

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  doc. dr. Boštjan Slivnik
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 16.09.2009 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se doc. dr. Boštjanu Slivniku, ki je strokovno vodil potek diplomskega dela in je s svojimi nasveti prispeval k dokončanemu oblikovanju diplomskega dela.

Posebna zahvala gre mojim staršem, mami Rozaliji in očetu Jožefu, ki sta me podpirala čez celoten študij.

Zahvaljujem se tudi zaposlenim v podjetju Hermes Softlab d.o.o., še posebej Petru Marinšku za strokovno pomoč pri izvedbi praktičnega dela.

To diplomsko delo posvečam svojim staršem, ki sta mi vedno stala ob strani in me vzpodbujala na celotni poti. Hvala, brez vaju mi ne bi uspelo.

Kazalo

Povzetek **1**

Abstract **2**

1 Uvod **3**

2 Uporabniški vmesnik **6**
2.1 Kaj je uporabniški vmesnik 6
2.2 Zgodovina uporabniškega vmesnika 7

3 Enotni vmesnik storitev **10**
3.1 Kaj je enotni vmesnik storitev 10
3.2 Definicija vmesnika API za enotni vmesnik storitev 12
3.3 Dodajanje novega servisa 15
3.4 Vmesnik za grafične knjižnice 16
3.5 Omejitve in prednosti izvorne kode uporabniškega vmesnika in strežnika storitve 17

4 Prevajalnik **18**
4.1 Kaj je prevajalnik 18
4.1.1 Zgodovina prevajalnikov 19
4.1.2 Zgradba prevajalnika 19
4.2 Delovanje prevajalnika 22
4.2.1 Problemi 24
4.2.2 Pomanjkljivosti 24
4.2.3 Nekaj o uporabljenem orodju Xerces 25

5 XML notacija **27**
5.1 Kaj je XML 27
5.2 AGUI XML notacija 27

5.3 RULE XML notacija 34

6 Navodila za uporabo prevajalnika 38

7 Primeri XML opisa grafičnega vmesnika 41
7.1 XML izvorna koda primera 41
7.2 C++ izvorna koda prevedenih XML datotek 43
7.3 Posnetek aplikacije 49

8 Zaključek 50

A XML shema 51
A.1 XML shema za AGUI datoteko 51
A.2 XML shema za RULE datoteko 55

Seznam slik 57

Seznam tabel 58

Literatura 59

Seznam uporabljenih kratic in simbolov

XML	Razširljiv označevalni jezik
AGUI	Abstraktni opis grafičnega uporabniškega vmesnika
RULE	Opis lastnosti grafičnih elementov
C++	Objektno orientiran programski jezik
FPGA	Field Program Gate Array
UNIX	Računalniški operacijski sistem
GNOME	Programska oprema namiznega okolja
MP3	Digitalni kodirni format za kompresijo glasbe
CD	Optični disk, ki služi za zapis podatkov ali glasbe
HVAC	Sistem za kontroliranje temperature znotraj avtomobila
CE	Consumer electronics
W3C	World Wide Web Consortium

Povzetek

V diplomski nalogi je predstavljena rešitev za sistem avtomatskega generiranja enotnega uporabniškega vmesnika za podporo različnih naprav. Sistem podpira enostavno dodajanje novih naprav, ki jih nato lahko enostavno krmilimo preko generiranega enotnega uporabniškega vmesnika. Tak sistem pride do izraza predvsem v avtomobilih, saj je tam pomembno, da uporabnik pride do želenih funkcij na čim bolj enostaven način. Tako lahko s tem sistemom krmili uporabnik svoj MP3 predvajalnik kar preko ukazov na volanskem obroču avtomobila, vožnja avtomobila pa je nato ob uporabi mobilne naprave varnejša. V okviru diplomske naloge je bil razvit način opisa naprave z XML-om, kateri omogoča enostavno kreiranje uporabniškega vmesnika. Za ta način opisa je realiziran prevajalnik, kateri ta opis prevede v C++ izvorno kodo.

Za lažje delo s prevajalnikom je bil tudi razvit uporabniški vmesnik zanj. Uporabniški vmesnik je spisan s pomočjo orodja QT, ki zagotavlja prenosljivost med operacijskimi sistemi.

Ključne besede:

prevajalnik, uporabniški vmesnik, enotni vmesnik servisov, xml

Abstract

The bachelor thesis covers the solution for automatic generation of user interface with support for various mobile consumer electronics (CE) devices. This tool provides interface to simplified adding new devices. These devices then can be controlled with unified user interface. This become very useful in cars, because there the main concern is safety. In that way user can easily access to the functions of CE device. For example, driver can operate MP3 player with buttons on a steering wheel. So the driver can safely uses the CE devices. In addition the method how to discribe user interface with XML was developed. With this method creating the user interface is easy and quick. Also for that method the compiler was developed. This compiler compiles this method in to C++ source code.

For convenience the user interface for compiler was created. This user interface was created with QT tool. Because of this, the compiler is supported on various operating systems.

Key words:

compiler, user interface, unified servis interface, xml

Poglavje 1

Uvod

Prenosne mobilne naprave vsak dan bolj prodirajo v naša vsakdanja življenja. Funkcionalnosti takšnih naprav rastejo eksponentno, uporabniki teh naprav pa pričakujejo, da bodo lahko do teh funkcionalnosti dostopali v vsaki situaciji. V tej diplomski nalogi se bomo osredotočili na uporabo teh naprav v avtomobilu.

Zaradi varnostnih razlogov avtomobilski uporabniški vmesnik strmi k temu, da voznik pride čim prej do želenih funkcij. Da bi voznik upravljal svojo prenosno napravo čim varneje, je tako najboljši način ta, da jo upravlja preko avtomobilskega uporabniškega vmesnika. Za primer naj navedemo MP3 predvajalnik. Seznam skladb MP3 predvajalnika je tako lahko prikazan na avtomobilskem centralnem zaslonu, funkcija za začetek predvajanja želene skladbe pa je lahko dostopna preko tipk na volanu. Tako voznik lahko začne poslušati svojo najljubšo skladbo dokaj varno.

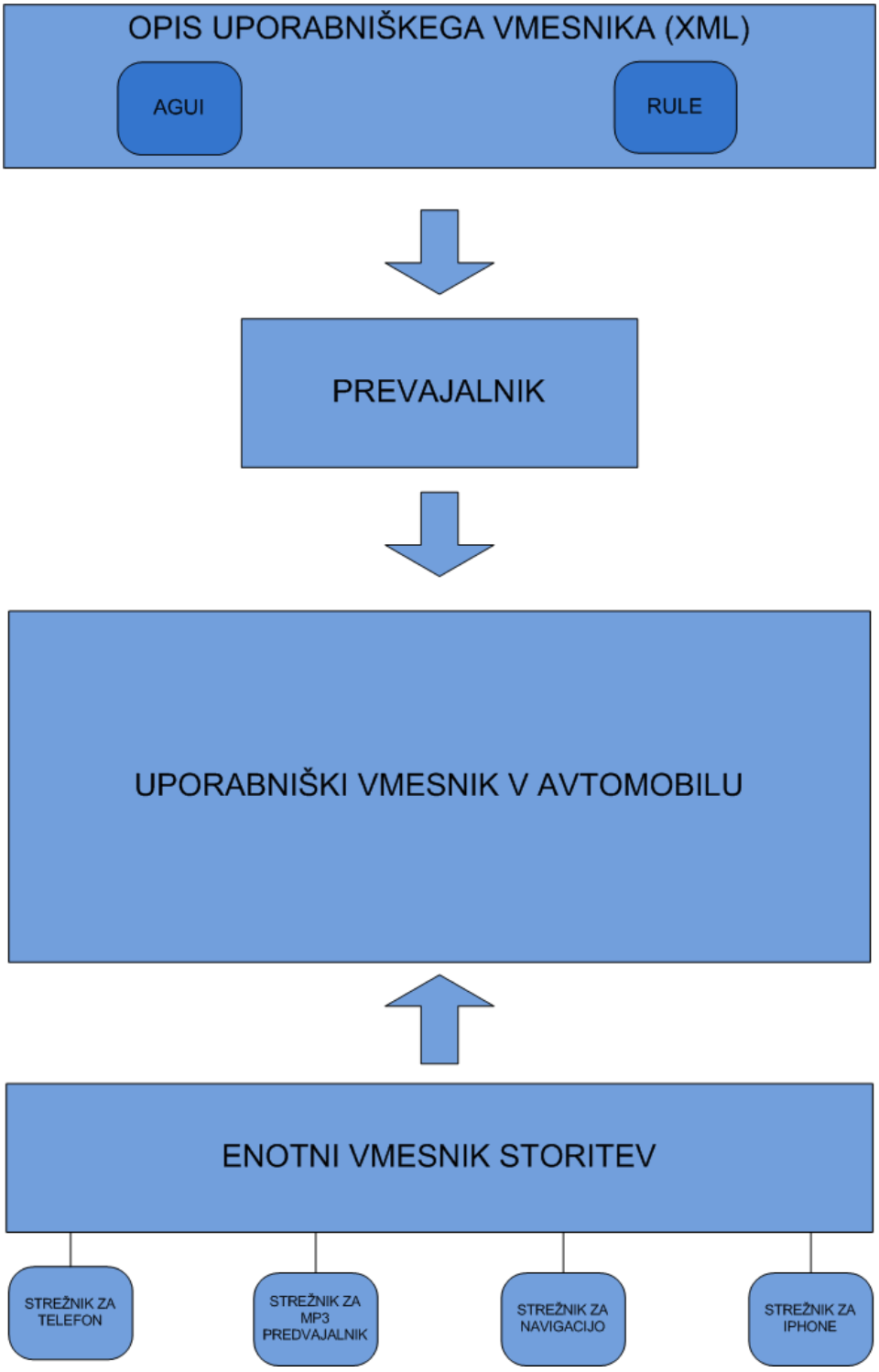
Razvojni cikel novih mobilnih naprav je v primerjavi z življenjskim ciklom avtomobila zelo kratek. Tako pridemo do dveh problemov, ki se nanašata na integracijo mobilne naprave v avtomobilski uporabniški vmesnik. Prvi problem je časovni, saj je ročno dodajanje nove naprave v sistem časovno zelo potratno. Tako bi naprava že zastarela, ko bi bila pripravljena za uporabo. Drugi problem je ta, da uporabniki želijo v avtomobilu uporabljati novejše mobilne naprave, ki še sploh niso obstajale, ko je bil avtomobil zgrajen. Tako je ena izmed možnih rešitev ta, da se uvede sistem enotnih storitev s sistemom za enostavno razširitev uporabniškega vmesnika. Ta rešitev tako omogoča enostavno integracijo novih naprav.

Da se tak sistem razvije, je bilo potrebno rešiti kar nekaj problemov. Poglavitna problema sta bila poenotenje podatkov ter univerzalni sistem za krmiljenje in komunikacijo z mobilnimi napravami. Tako rabimo še sistem za enostavno kreiranje ter razširjanje uporabniškega vmesnika.

Namen te diplomske naloge je izdelava sistema za generiranje enotnega uporabniškega vmesnika za podporo različnih naprav. Ta sistem omogoča enostavno dodajanje novih naprav ter enostavno kreiranje uporabniškega vmesnika s pomočjo XML opisa. Za enostavno dodajanje novih naprav je bil razvit enotni vmesnik servisov. Razvit je bil tudi način opisa uporabniškega vmesnika z XML-jem. Ta opis omogoča enostavno kreiranje uporabniških vmesnikov. Ob tem je bil razvit tudi prevajalnik, ki prevede ta opis v C++ izvorno kodo.

Da bi dodali novo napravo v sistem, mora proizvajalec naprave ali inženir narediti sledeče stvari: Potrebno je napisati strežnik, ki poskrbi za komunikacijo med napravo in sistemom. Funkcije, ki jih mora ta strežnik podpirati, so opisane v poglavju 3.3. Nato je potrebno napisati RULE in AGUI datoteki, ki opišeta uporabniški vmesnik za to napravo. Opis teh dveh datotek se nahaja v poglavju 5.2 in 5.3. Ko je opis uporabniškega vmesnika narejen, se ti dve datoteki prevedeta s prevajalnikom, tako da dobimo C++ izvorno kodo. To izvorno kodo ter izvorno kodo strežnika storitve nato integriramo v sistem avtomobila. Ta integracija v sistem se lahko potem izvrši na samem servisu avtomobila. Slika 1.1 prikazuje zgradbo celotnega sistema.

Besedilo diplomske naloge bo korak po koraku pojasnilo osrednje točke sistema, ki so kritične za integracijo mobilnih naprav v sam sistem, ter povezavo med uporabniškim vmesnikom in mobilno napravo. Vsaka enota bo pojasnila osnovne principe ter pravila delovanja in uporabe. Na koncu je še podan testni primer ter navodilo za uporabo prevajalnika.



Slika 1.1: Slika celotnega sistema

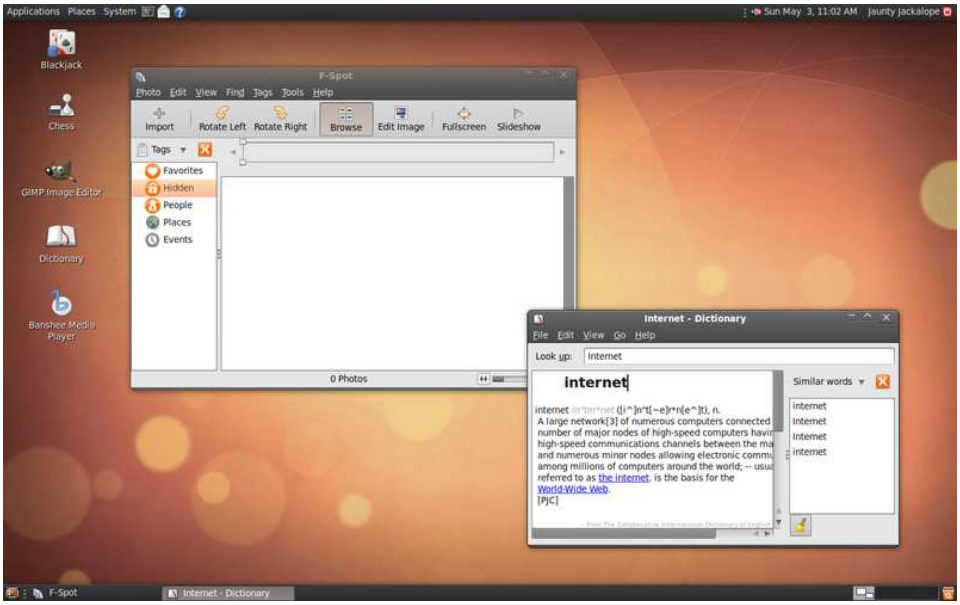
Poglavje 2

Uporabniški vmesnik

2.1 Kaj je uporabniški vmesnik

Za delo z napravo potrebujemo kontrolni sistem, preko katerega krmilimo napravo. Preko tega sistema lahko tudi razberemo stanje naprave. Za primer vzemimo avtomobil. Za spreminjanje smeri uporabljamo volan, hitrost pa uravnavamo s pedali in menjalnikom. Pozicijo avtomobila določimo s pogledom skozi okno, hitrost pa odčitamo preko prikazovalnika hitrosti. V tem primeru uporabniški vmesnik sestavljajo volan, pedali, menjalnik, okna ter prikazovalnik hitrosti.

Termin uporabniški vmesnik se pogosto uporablja v kontekstu računalništva in elektronskih naprav. V zadnjem času pa se je uveljavil termin grafični uporabniški vmesnik, saj so že vse elektronske naprave precej zmogljive in zlahka prikažejo tudi slike. Te naprave tako lahko krmilimo z direktno interakcijo s slikovnimi elementi. Tako uporabnik dobi tudi boljšo predstavo o tem, kaj se z napravo dogaja in v katerem stanju je. Slika 2.1 prikazuje primer grafičnega uporabniškega vmesnika.

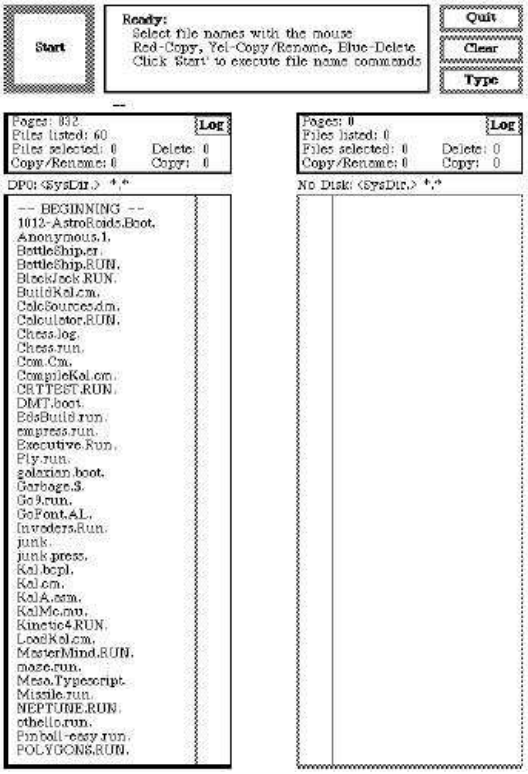


Slika 2.1: Grafični uporabniški vmesnik GNOME okolja

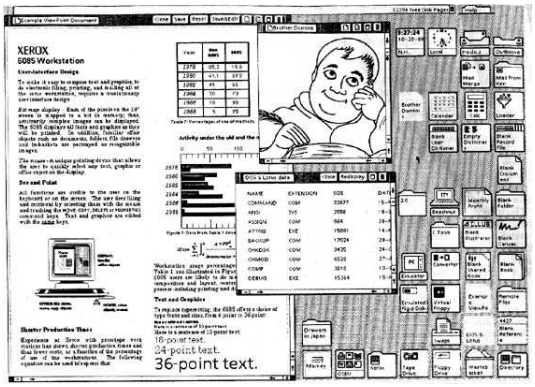
2.2 Zgodovina uporabniškega vmesnika

Prvi uporabniški vmesniki so bili tekstovne narave. Ti vmesniki so bili zelo neintuitivni in so zahtevali dodatno izobraževanje uporabnikov, saj so se morali uporabniki naučiti, katera tekstovna komanda je za določeno akcijo. To je zahtevalo precej časa in dodatnega denarja.

Tako je leta 1970 podjetje Xerox predstavilo alternativo tekstovnemu upravljanju, ki temelji na osnovni metodi komuniciranja, ki jo poznamo pod imenom kretnje. Xerox sistema Altos in STAR sta tako predstavila miško ter princip pokaži in označi kot primarno metodo za komunikacijo človeka z računalnikom. Primer sistema Altos prikazuje Slika 2.2, primer sistema STAR pa Slika 2.3.

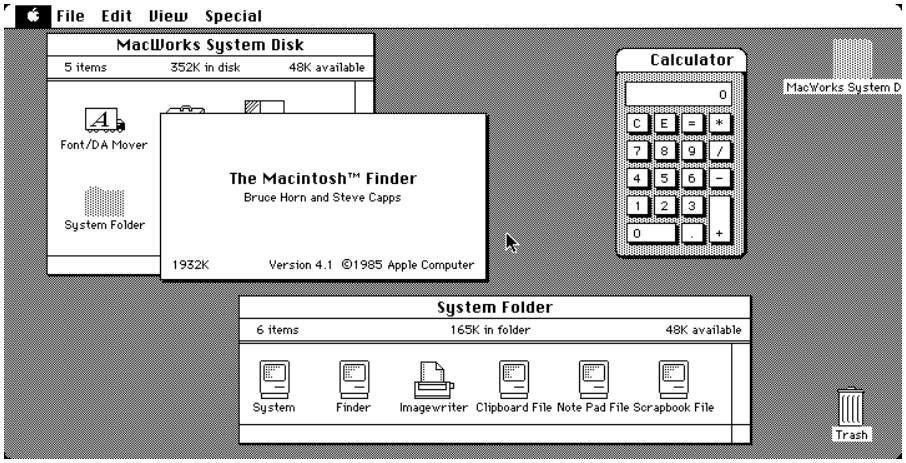


Slika 2.2: Sistem Xerox Alto



Slika 2.3: Sistem Xerox STAR

Vendar Xeroxu ni uspelo uspešno prodajati sistem STAR. Tako je podjetje Apple hitro prevzelo idejo miške in jo uporabilo v svojem sistemu Machintosh, izdanem leta 1984, ki je nato postal prvi široko uporabljen sistem z miško. Primer sistema Machintosh prikazuje Slika 2.4.



Slika 2.4: Sistem Xerox Machintosh

Leta 1985 je Microsft izdal svoj operacijski sistem Windows 1.0, ki je prav tako uporabljal miško. Leta 1987 je Apple predstavil Machintosh II, prvi barvni Machintosh. Tako je X Window sistem postal dostopen širši javnosti. Po tem letu je sledilo še kar nekaj drugih proizvajalcev s svojimi operacijskimi sistemi. Tu naj omenimo dva večja, in sicer NeXTStep podjetja NeXT ter grafični vmesniki, ki temeljijo na UNIX platformi.

Poglavje 3

Enotni vmesnik storitev

3.1 Kaj je enotni vmesnik storitev

Trg elektronskih naprav je v polnem zagonu. Skoraj vsak dan pride na tržišče neka nova naprava. Ljudje se na te naprave kar hitro navežemo in jih hočemo imeti ob sebi tako rekoč vsepovsod. Zaradi hitrega tempa življenja preživimo tudi veliko časa v avtomobilih. Kot vemo, pa je upravljanje teh naprav med vožnjo zelo nevarno in lahko pripelje do katastrofalnih posledic. Zato bi bilo dobro, da bi lahko uporabniki upravljali te naprave kar preko avtomobilskega uporabniškega vmesnika. Integracija elektronske naprave v obstoječi avtomobilski sistem se je izkazala za težavno, saj je na trgu veliko različnih proizvajalcev elektronskih naprav, ki pa nimajo nekega univerzalnega vmesnika, ki bi omogočal komunikacijo s temi napravami.

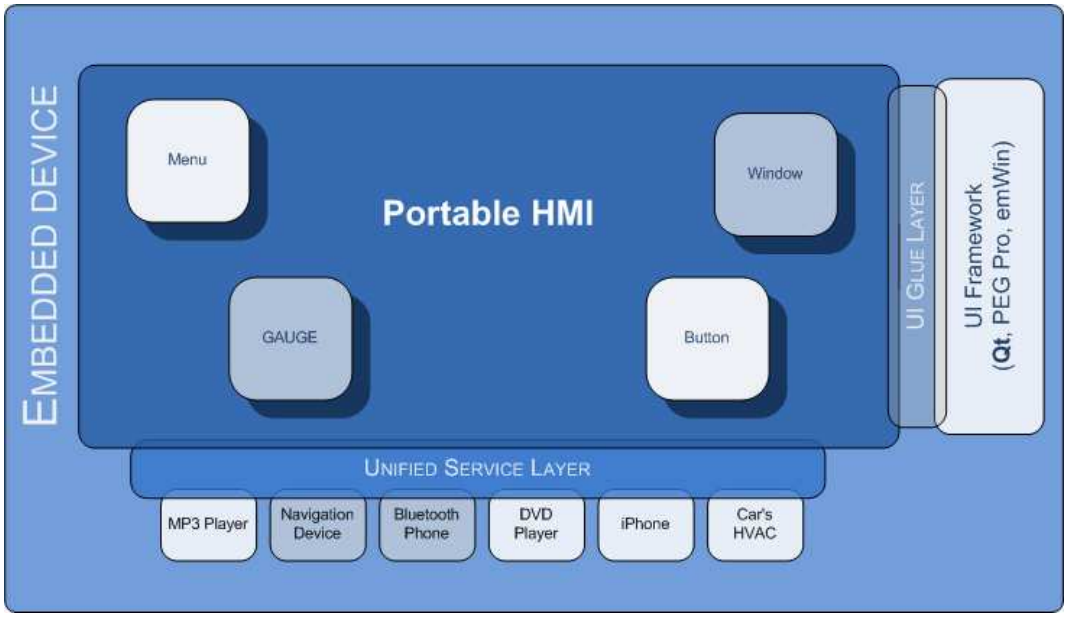
Da bi bila ta integracija elektronskih naprav v avtomobilski sistem enostavnejša, smo pri podjetju HERMES SoftLab d.o.o. razvili enotni vmesnik storitev. Shemo enotnega vmesnika storitev prikazuje Slika 3.1. Tako vsako elektronsko napravo obravnavamo kot neko storitev. Storitve delimo na štiri velike razrede:

- Multimedija: To so razne naprave kot MP3 predvajalnik, mobilni telefon, CD-predvajalnik ... Te naprave sistem avtomatsko prepozna kot vir seznama pesmi ter kot glasbeni predvajalnik.
- Navigacija: Naprave te vrste sistem prepozna kot napravo, ki nudi mape, prometne informacije, pozicijo ...
- HVAC: Specifične informacije o avtomobilu, to so trenutna poraba, temperatura motorja, notranja temperatura, zunanja temperatura ...

- Bluetooth: Razne naprave, ki podpirajo povezavo preko Bluetooth-a.

Enotni vmesnik storitev enako obravnava podatke, ki so istega tipa. Tako je vseeno, če naslov dobi iz imenika ali pa ga uporabnik vnese ročno. S tem smo dosegli to, da so podatki konsistentni in ni odvisno, od katere naprave pridejo. Tako je naš sistem zelo modularen in omogoča enostavno dodajanje novih naprav. Vse, kar moramo storiti, je to, da napišemo strežnik za napravo, ta strežnik potem poskrbi, da se podatkovni tipi ter signali pretvorijo v pravilno obliko.

Kot poseben servis je tu še vmesnik za grafične knjižnice. Ta omogoča uporabo različnih grafičnih knjižnic.



Slika 3.1: Enotni vmesnik servisov

3.2 Definicija vmesnika API za enotni vmesnik storitev

Za komunikacijo z določeno napravo komuniciramo preko vmesnika API, ki je enak za vse naprave v sistemu. Te funkcije so uporabljene v C++ izvorni kodi, katero naredi prevajalnik iz AGUI in RULE datotek. Funkcije, ki so na voljo, so sledeče:

- `HList &getAllClassServices(void):`

  Funkcija nam vrne seznam razredov servisov, ki so trenutno registrirani v sistemu.

- `HList &getServicesOfClass(ServiceClass servClass):`

  Funkcija nam vrne seznam servisov podanega razreda s parametrom servClass.

- `bboolean registerOnEvent(ServiceClass clasServ, HString &serviceName, HRegEvent_t *rEvStr ):`

  S to funkcijo se prijavimo na nek dogodek naprave. Parametri funkcije so sledeči:

  - `clasServ:` podamo ime razreda, kateremu pripada servis;
  - `serviceNme:` ime servisa;
  - `rEvStr:` struktura tipa HRegEvnt, ki vsebuje informacije za vračanje podatkov.

- `bboolean deregisterOfEvent(ServiceClass clasServ, HString &serviceName, HRegEvent_t *rEvStr ):`

  S to funkcijo se odjavimo od nekega dogodka. Parametri funkcije pa so enaki kot pri funkciji *registerOnEvent*.

- `void setPropertyOfService(ServiceClass classType, HString &serviceName, HString &prop, pointer val):`

  Funkcija je namenjena nastavljanju določenega parametra servisa. Parametri funkcije so sledeči:

  - `classType:` podamo ime razreda, kateremu pripada servis;

  - `serviceNme`: ime servisa;
  - `prop`: ime parametra;
  - `val`: kazalec na želeno vrednost parametra.

- `pointer getPropertyOfService(ServiceClass classType, HString &serviceName, HString &prop):`

  Funkcija je namenjena branju določenega parametra servisa. Primer takšnega parametra je prisotnost CD ploščka v predvajalniku. Parametri funkcije so sledeči:

  - `classType`: podamo ime razreda, kateremu pripada servis;
  - `serviceNme`: ime servisa;
  - `prop`: ime parametra.

- `FunctionState invokeFunction(ServiceClass classType, HString &serviceName, HString &funName, HInvokeFunction_t *iFunStr):`

  Funkcija, namenjena klicanju neke funkcije servisa. Primer takšne funkcije bi bila zamenjava trenutno predvajane pesmi. Parametri funkcije so sledeči:

  - `classType`: podamo ime razreda, kateremu pripada servis;
  - `serviceNme`: ime servisa;
  - `funName`: ime funkcije ki jo hočemo poklicati;
  - `iFunstr`: struktura, ki vsebuje dodatne parametre, kot je kazalec na povratno funkcijo, razni podatki ...

- `pointer getServiceHandle(ServiceClass classType, HString &serviceName):`

  Funkcija nam vrne kazalec na instanco servisa. Parametri funkcije so sledeči:

  - `classType`: podamo ime razreda, kateremu pripada servis;
  - `serviceNme`: ime servisa.

- `pointer getFunctionHandle(ServiceClass classType, HString &serviceName, HString &funName):`

Funkcija nam vrne kazalec na želeno funkcijo servisa. Parametri funkcije so sledeči:

- `classType:` podamo ime razreda, kateremu pripada servis;
- `serviceNme:` ime servisa;
- `funName:` ime funkcije.

- `FunctionState invokeFunction(pointer functionHandle, HInvokeFunction_t *iFunStr):`

Funkcija, namenjena klicanju neke funkcije servisa. Primer takšne funkcije bi bil zamenjava trenutno predvajane pesmi. Parametri funkcije so sledeči:

- `functionHandle:` pointer na želeno funkcijo servisa;
- `iFunStr:` struktura, ki vsebuje dodatne parametre, kot je kazalec na povratno funkcijo, razni podatki ...

- `bboolean registerOnEvent(pointer serviceHandle, HRegEvent_t *rEvStr):`

S to funkcijo se prijavimo na nek dogodek naprave. Parametri funkcije so sledeči:

- `serviceHandle:` kazalec na instanco servisa;
- `rEvStr:` struktura tipa HRegEvnt, ki vsebuje informacije za vracanje podatkov.

- `bboolean deregisterOfEvent(pointer serviceHandle, HRegEvent_t *rEvStr):`

S to funkcijo se odjavimo z nekega dogodka. Parametri funkcije pa so enaki kot pri funkciji *registerOnEvent*.

- `void setPropertyOfService(pointer serviceHandle, HString &prop, pointer val):`

Funkcija je namenjena nastavljanju določenega parametra servisa. Parametri funkcije so sledeči:

- `serviceHandle:` kazalec na instanco servisa;

  - `prop`: ime parametra;
  - `val`: kazalec na želeno vrednost parametra.

- `pointer getPropertyOfService(pointer serviceHandle, HString &prop):`

  Funkcija je namenjena branju določenega parametra servisa. Parametri funkcije so sledeči:

  - `serviceHandle`: kazalec na instanco servisa;
  - `prop`: ime parametra.

3.3 Dodajanje novega servisa

Če hočemo v v sistem dodati novo napravo, moramo napisati strežnik za to napravo. Ti strežniki so vidni na sliki 1.1. Da sistem napravo pravilno prepozna, mora strežnik vsebovati naslednje funkcije:

- `HList *getEventList(void):`

  Funkcija vrne seznam imen vseh dogodkov.

- `HList *getPropertiesList(void):`

  Funkcija vrne seznam imen vseh parametrov, ki jih lahko beremo ali nastavimo.

- `HList *getFunctionPointerList(void):`

  Funkcija vrne seznam kazalcev na funkcije servisa.

- `HList *getFuntionList(void):`

  Funkcija vrne seznam imen vseh funkcij servisa.

- `bboolean registerEvent(HRegEvent_t *rEvStr):`

  S to funkcijo se prijavimo na nek dogodek naprave. Parametri funkcije so sledeči:

  - `rEvStr`: Struktura tipa HRegEvnt, ki vsebuje informacije za vračanje podatkov.

- `bboolean deregisterEvent(HRegEvent_t *rEvStr):`

  S to funkcijo se odjavimo od nekega dogodka. Parametri funkcije pa so enaki kot pri funkciji *registerOnEvent*.

- `HString serviceName(void):`

  Funkcija vrne ime servisa.

- `ServiceClass getServiceClass(void):`

  Funkcija vrne ime razreda, kateremu pripada servis.

- `void setProperty(HString &prop, pointer val):`

  Funkcija je namenjena nastavljanju določenega parametra servisa. Parametri funkcije so sledeči:

  - `prop`: Ime parametra;
  - `val`: kazalec na želeno vrednost parametra.

- `pointer getProperty(HString &prop):`

  Funkcija je namenjena branju določenega parametra servisa. Parametri funkcije so sledeči:

  - `prop`: ime parametra.

3.4 Vmesnik za grafične knjižnice

S tem vmesnikom za grafične knjižnice smo dosegli to, da je koda, ki jo kreira prevajalnik, prenosljiva. Poleg tega omogoča, da lahko na isti platformi uporabimo različne grafične knjižnice. Ta vmesnik moramo napisati za vsako grafično knjižnico posebej ter mogoče dodati še kakšne specifike za različne platforme. To, da moramo narediti vmesnik za vsako grafično knjižnico posebej, ni velika omejitev, saj to moramo narediti samo enkrat. Katere grafične gradnike in katere funkcije morajo podpirati ti gradniki, pa je opisano s prototipi. V tem vmesniku lahko naredimo tudi kar nekaj optimizacije za določeno platformo in grafično knjižnico.

3.5 Omejitve in prednosti izvorne kode uporabniškega vmesnika in strežnika storitve

Kot smo že omenili, je celoten sistem prenosljiv med različnimi procesorji ter operacijskimi sistemi. Zaradi te prenosljivosti smo uvedli nove podatkovne tipe. Tako je koda, ki je napisana za strežnik storitve, ter koda, ki jo naredi prevajalnik, enostavno prenosljiva, saj jo ni potrebno spreminjati. Za programski jezik C in C++ obstajajo različni modeli podatkovnih tipov. Ti modeli podatkovnih tipov so sledeci: LP64, ILP64, LLP64, ILP32 in LP32 [9]. Večina današnjih 64-bitnih prevajalnikov uporablja LP64 model, to so prevajalniki za Solaris, AIX, HP-UX, Linux, Max OSX, FReeBSD ter IBM z/OS. Microsoftov VC++ prevajalnik pa uporablja LL64 model. LP32 model pa uporablja C prevajalnik za Windows 3.1.

Iz tabele 3.1 je lepo razvidno, kako so različni podatkovni tipi pri posameznem modelu podatkovnih tipov. Tako je priporočljivo, da razvijalci strežnika za napravo uporabljalo le specificirane podatkovne tipe in se tako izognejo poznejšim nevšečnostim pri prenosu programske opreme na drug operacijski sistem ali na drug procesor.

Datatype	**LP64**	**ILP64**	**LLP64**	**ILP32**	**LP32**
char	8	8	8	8	8
short	16	16	16	16	16
_int32		32			
int	32	64	32	32	16
long	64	64	32	32	32
long long			64		
pointer	64	64	64	32	32

Tabela 3.1: Tabela modelov podatkovnih tipov prevajalnikov. Velikosti posameznih podatkovnih tipov so podani v bit-ih.

Poglavje 4

Prevajalnik

4.1 Kaj je prevajalnik

Prevajalnik je bil na začetku program, ki je "prevedel" podprograme (ang. subrutines), to je tako imenovani povezovalnik (ang. link-loader). Ko je leta 1954 prišla v uporabo besedna zveza algebraični prevajalnik (ang. algebraic compiler), se je pomen termina prevajalnik že prenesel v sedanjega [6].

Tako danes pod besedo prevajalnik razumemo računalniški program oz. skupek programov, ki pretvori izvorno kodo, ki je napisana v računalniškem jeziku, v neki drugi računalniški jezik. Ponavadi je ta drugi računalniški jezik kar binarna koda. Ta koda se potem lahko izvaja direktno na procesorju računalniškega sistema. Poznamo tudi novejše programske jezike, kot je na primer Java. Ko prevedemo program, napisan v teh programskih jezikih, dobimo tako imenovano bajt kodo (ang. byte-code). To bajt kodo lahko potem poganjamo na različnih sistemih. Vendar za izvrševanje te bajt kode potrebujemo posebni virtualni stroj (ang. virtual machine). Poznamo še tudi interpreterje. Interpreter pomeni računalniški program, ki izvršuje inštrukcije, napisane v nekem programskem jeziku. Znani predstavniki interpreterjev so MATLAVB, Lisp, PHP ... Vsak programski jezik se lahko interpretira ali pa prevede v binarno kodo. Tretja vrsta prevajalnikov so tako imenovani strojni prevajalniki. Ti prevajalniki so sposobni narediti iz izvorne kode nek del strojne opreme, na primer procesor, pomnilno celico itd. Dva taka najbolj znana jezika sta FPGA in ASIC. Ti strojni prevajalniki ne naredijo izvajalne kode, ampak povezave med tranzistorji ali iskalne tabele (ang. lookup tables).

4.1.1 Zgodovina prevajalnikov

Programi za prve računalnike so bili napisani v zbirnem programskem jeziku. S časom so postajali procesorji vedno zmogljivejši ter kompleksnejši. Pisanje programov v zbirnem jeziku za te procesorje je tako postalo težavno in časovno potratno, posledično se je s tem tudi višala cena programov. Tako so se razvili višje nivojski programski jeziki. Pisanje prevajalnika je opravičilo to, da se je program v višje nivojskem jeziku napisal enkrat in potem se je samo prevedel za različne procesorje. Tako je bilo treba na začetku napisati le prevajalnik za določen procesor in nato ni bilo več potrebe po programiranju v zbirnem jeziku za ta procesor. Lepa lastnost višje nivojskih programskih jezikov je tudi ta, da je pisanje programov dosti lažje in hitrejše. S tem se je nižala cena programski opremi in je tako postala dostopnejša širši množici. Vendar pa je pisanje dobrih prevajalnikov bilo na začetku precej težavno opravilo, saj so bili prvi računalniki zelo omejeni s spominom.

Proti koncu 1950 se je porajala ideja o strojno neodvisnih programskih jezikih. Tako je bilo razvitih kar nekaj eksperimentalnih prevajalnikov. Prvi prevajalnik je razvil Grace Hopper, leta 1952, za programski jezik A-0. FORTRANOV glavni inženir Jhon Backus, zaposlen pri podjetju IBM, je kot prvi leta 1957 predstavil celoten prevajalnik. Leta 1960 pa je COBOL kot prvi programski jezik bil preveden za različne računalniške arhitekture.

Ideja o uporabi višje nivojskih programskih jezikov se je hitro uveljavila v različnih aplikacijskih domenah. Zaradi razširjanja funkcionalnosti novih programskih jezikov ter povečevanje kompleksnosti računalniške arhitekture, so postali tudi prevajalniki kompleksnejši.

Na začetku so bili prevajalniki napisani v zbirnem jeziku. V programskem jeziku Lisp je bil narejen prvi prevajalnik, ki je prevajal programski jezik, v katerem je bil tudi sam narejen. Ta prevajalnik je leta 1962 naredil Tim Hart iz MIT-ja. Od leta 1970 je bilo pogosto, da je bil prevajalnik napisan v programskem jeziku, katerega je potem tudi prevajal. Vseeno pa so bili prevajalniki najpogosteje napisani v programskem jeziku C ali Pascal.

4.1.2 Zgradba prevajalnika

Na zasnovo prvih prevajalnikov je vplivala predvsem izkušenost osebe, ki je pisala prevajalnik, kompleksnost prevajalnika ter razpoložljiva sredstva. Pod razpoložljiva sredstva štejemo predvsem zmogljivost računalniškega sistema, na katerem bo tekel ta prevajalnik.

Prevajalnik za dokaj enostaven programski jezik, napisan s strani ene osebe, je lahko enostaven monolitni program. Ko pa imamo obsežni, kompleksni

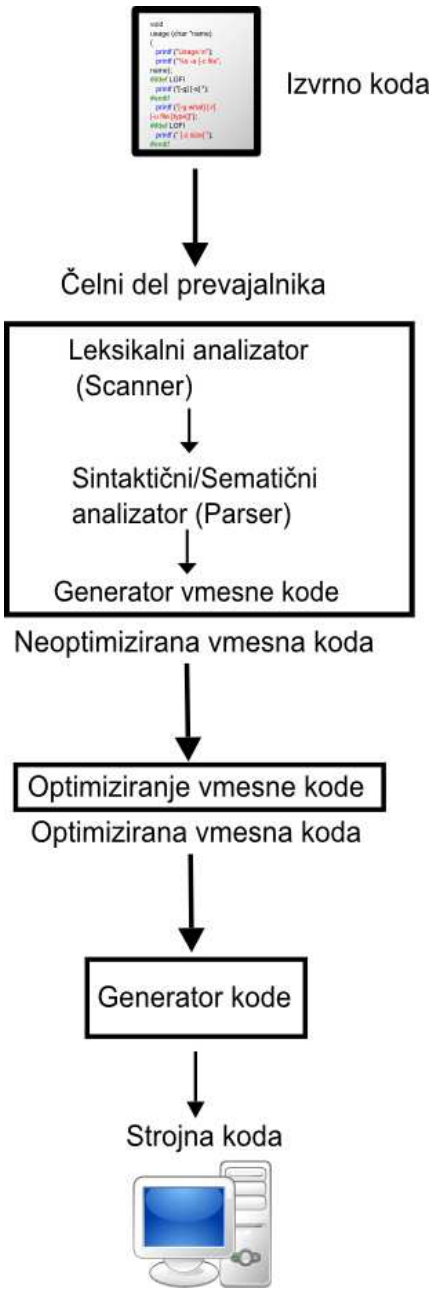
višje nivojski programski jezik, pri katerem je pomemben tudi kvaliteten prevod v izvorno kodo, se izkaže, da je potrebno prevajalnik zasnovati po fazah. To, da imamo prevajalnik sestavljen po fazah, pomeni, da lahko razvoj prevajalnika razdelimo v manjše dele in tako lahko prevajalnik razvija več ljudi hkrati. S tem ko imamo prevajalnik razdeljen na manjše dele, lahko enostavno zamenjamo kak del z bolj optimalnim oziroma pozneje lažje dodamo kako novo funkcionalnost.

Vsi, razen majhnih prevajalnikov, so razdeljeni vsaj v dva dela Slika 4.1. Ti deli so v grobem opredeljeni v dva dela, imenovana čelni del (eng. front end) in hrbtni del (eng. back end). Točka, kjer se ta dela delita, je vedno stvar debate. Nikoli ne moremo točno opredeliti, kaj še spada v prednji in kaj v zadnji del. Prednji del je v splošnem sestavljen iz sintaktičnega in semantičnega procesiranja skupaj s prevodom v neko vmesno kodo.

Vmesni del je ponavadi zasnovan tako, da izvaja optimizacijo na vmesni kodi. Optimizacija vmesne kode je neodvisna od izvorne/strojne kode. S tem tudi dosežemo to, da lahko enostavno zamenjamo programski jezik izvorne kode ali jezik strojne kode in je optimizacija še vedno enaka.

Hrbtni del prevajalnika potem prevzame izhod od vmesnega dela. Ta del nato lahko izvede nekaj dodatne analize, transformacije ter tudi optimizacijo. Sama optimizacija je odvisna od ciljne strojne kode. Na koncu zgenerira kodo, ki je vezana na določeno strojno opremo in pa tudi operacijski sistem.

Ta pristop k zgradbi prevajalnika nam omogoča, da lahko kombiniramo različne programske jezike z različnimi hrbtnimi deli za različno strojno opremo. Praktični primer takšnega pristopa je GNU Compiler Collection, LLVM in Amsterdam Compiler Kit. Ti prevajalniki imajo več različnih čelnih in hrbtnih delov.

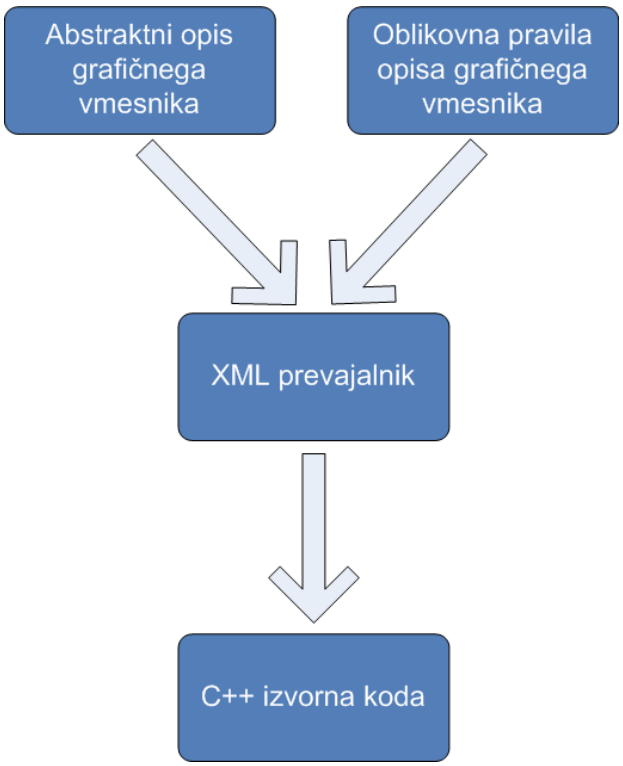


Slika 4.1: Zgradba enostavnega prevajalnika

4.2 Delovanje prevajalnika

Prevajalnik, ki smo ga razvili pri podjetju HERMES SoftLab d.o.o, je prevajalnik, ki prevaja iz višje nivojskega jezika v višje nivojski jezik. V našem primeru prevede način opisa uporabniškega vmesnika v C++ izvorno kodo. Za ta korak smo se odločili zato, ker je razvijanje uporabniškega vmesnika precej zamudno in težavno opravilo. To še posebej pride do izraza pri manjših elektronskih napravah, za katere ne obstaja nek grafični urejevalnik za kreiranje grafičnega vmesnika. Drugi problem je optimizacija grafičnega vmesnika, saj so elektronske naprave precej omejene s hitrostjo procesorja in količino pomnilnika.

Vhod Prevajalnika predstavljata dve XML datoteki, ki morata imeti enako ime. V eni datoteki je abstraktni opis uporabniškega grafičnega vmesnika, v nadaljevanju AGUI datoteka , ta datoteka ima končnico *agui*. V drugi datoteki pa so opisana oblikovna pravila grafičnega vmesnika, v nadaljevanju RULE datoteka, končnica te datoteke pa je *rule*. Podrobnejši opis AGUI datoteke je v Poglavju 5.2, opis RULE datoteke pa v Poglavju 5.3. Izhod prevajalnika je C++ izvorna koda Slika 4.2.



Slika 4.2: Potek prevajanja XML datotek

Kot vsi kompleksnejši prevajalniki, je tudi ta prevajalnik razdeljen na čelni in hrbtni del. V čelnem delu poteka sintaktična in semantična analiza. Za čelni del prevajalnika smo uporabili odprto kodno programsko knjižnico Xerces. To orodje na podlagi XML sheme avtomatično preveri pravilnost XML datoteke. XML shema za AGUI in RULE XML datoteki se nahaja v Dodatku A. Če datoteka ustreza shemi, orodje vrne drevo XML znack (ang. XML tag).

Hrbtni del prevajalnika predstavlja optimizacija ter kreiranje C++ kode iz drevesa XML značk. Ob prvem sprehodu drevesa prevajalnik naredi deklaracijsko datoteko (ang. header file) ter si shrani v seznam vse grafične elemente. Nato prevajalnik kreira glavno datoteko (.cpp). Kreiranje te datoteke poteka po naslednjem postopku: Prevajalnik vzame element iz seznama grafičnih elementov in gre gledat v drevo XML značk vse njegove lastnosti ter akcije. S tem smo dosegli to, da so vse lastnosti in akcije nekega grafičnega elementa na enem mestu. Tako je koda lepo pregledna in razumljiva.

Uporabniški vmesnik XML prevajalnik pa je narejen s pomočjo QT knjižnice. Za to knjižnico smo se odločili, ker deluje na večini operacijskih sistemih za osebne računalnike. Tako naš prevajalnik dela na Windows, Linux, Mac ... operacijskih sistemih.

4.2.1 Problemi

Kot pri vsakem večjem projektu, so se tudi pri tem pojavile določene težave. Največja težava je bila določiti primerne XML značke. Pri določanju značk je bilo treba paziti, da so bile smiselne in da se s čim manj opisa doseže čim večji efekt. Takšna zasnova je bila potrebna zato, ker smo hoteli doseči, da je kreiranje uporabniškega vmesnika z XML opisom hitro in enostavno.

Drug problem je bil preverjanje pravilnosti XML datoteke ter razčlenjevanje (ang. parsing) XML značk. Ta problem smo nato rešili z uporabo odprto kodne programske knjižnice Xerces.

Še en tak problem je bil prenosljivost programske kode. To prenosljivost smo rešili z vpeljavo enotnega vmesnika servisov. Ta enotni vmesnik servisov pa je opisan v Poglavju 3.

4.2.2 Pomanjkljivosti

Glavna pomanjkljivost prevajalnika je ta, da še ni vseh XML značk za opis uporabniškega vmesnika. Definirane so le osnovne značke za opisovanje, kot je na primer pozicija ter velikost grafičnega elementa. Način opisa uporabniškega vmesnika pa ni popoln zaradi tega, ker je projekt razvojne narave in je bil

namen pokazati, kako takšna stvar funkcionira in kaj vse je treba narediti. Ker pa je XML enostavno razširljiv, v prihodnosti ne bo težav z dodajanjem novih XML značk za razširitev funkcionalnosti opisa.

Pomanjkljivost prevajalnika je tudi ta, da je velikost zaslona fiksno določena na 320×240 pikslov.

XML značke morajo biti napisane v enakem vrstnem redu, kot so napisane v XML shemi.

Prevajalnik je zasnovan modularno, zato so vse te pomanjkljivosti enostavno rešljive.

4.2.3 Nekaj o uporabljenem orodju Xerces

Za knjižnico Xerces-C++, v nadaljevanju Xerces, smo se odločili zato, ker je podprta na različnih operacijskih sistemih. V Tabeli 4.1 je razvidno, na katerih operacijskih sistemih je knjižnica Xerces podprta [7]. Tako je XML prevajalnik podprt na različnih sistemih in ni problema s prenosljivostjo.

Xerces je validacijski XML razčlenjevalnik, napisan v prenosljivi C++ kodi. Knjižnica omogoča enostavno branje in pisanje XML podatkov. Xerces prav tako podpira različne metode za delo z XML podatki. Te metode so sledeč: DOM, SAX, SAX2. Xerces se drži XML 1.0 priporočil ter drugih pripadajočih standardov.

Operacijski sistem	**Prevajalnik**
32-bit	
Windows x86	MS Visual C++ 7.1 (2003)
Windows x86	MS Visual C++ 8.0 (2005)
Windows x86	MS Visual C++ 9.0 (2008)
Linux x86	GCC 3.4.x or later
Solaris 10 x86	GCC 3.4.x or later
Solaris 10 x86	Sun C++ 5.7 (Studio 10) or later
Solaris 10 SPARC	GCC 3.4.x or later
Solaris 10 SPARC	Sun C++ 5.7 (Studio 10) or later
AIX 5.3 PowerPC	IBM XL C++ 7.0 or later
HP-UX 11i PA-RISC	HP aCC A.03.x
HP-UX 11i IA-64	HP aCC A.06.x
MacOS X 10.4 "Tiger"x86	GCC 4.0.x (Xcode 2.x) or later
MacOS X 10.4 "Tiger"PowerPC	GCC 4.0.x (Xcode 2.x) or later
64-bit	
Windows x86-64	MS Visual C++ 8.0 (2005)
Windows x86-64	MS Visual C++ 9.0 (2008)
Linux x86-64	GCC 3.4.x or later
Solaris 10 x86-64	GCC 3.4.x or later
Solaris 10 x86-64	Sun C++ 5.7 (Studio 10) or later
Solaris 10 SPARC	GCC 3.4.x or later
Solaris 10 SPARC	Sun C++ 5.7 (Studio 10) or later
AIX 5.3 PowerPC	IBM XL C++ 7.0 or later
HP-UX 11i PA-RISC	HP aCC A.03.x
HP-UX 11i IA-64	HP aCC A.06.x

Tabela 4.1: Tabela podprtih operacijskih sistemov in pripadajočih prevajalnikov knjižnice Xerces

Poglavje 5

XML notacija

5.1 Kaj je XML

Razširljiv označevalni jezik (ang. Extensible Markup Language), krajše XML, je splošno namenska specifikacija za kreiranje individualnih označevalnih jezikov. Termin razširljiv je uporabljen zato, da poudari dejstvo, da ima načrtovalec označevalnega jezika veliko svobodo pri določanju označevalnih elementov.[8]

Namen XML-a je, da poudari predstavljene dokumente z enostavnostjo, splošnostjo ter uporabnostjo preko interneta. XML je postal široko uporabljan kot splošno namenski format za izmenjavo podatkov.

Zbirka orodij za delo z XML-om pomaga razvijalcem pri kreiranju spletnih strani, vendar je njihova uporabnost šla še dlje. XML, s kombinacijo z drugimi standardi, omogoča kreiranje vsebine dokumenta, ločeno od vizualnega izgleda. Tako lahko uporabimo vsebino dokumenta v različnih aplikacijah ali reprezentativnih okoljih. Najbolj pomembno je to, da XML določa osnovno sintakso za izmenjavo informacij med različnimi računalniki, aplikacijami, organizacijami, brez da bi morali dokument poslati skozi različne plasti konverzacije.

XML je priporočen standard s strani World Wide Web Consortium (W3C) in je zastonjski odprti standard.

5.2 AGUI XML notacija

V AGUI datoteki se nahaja samo abstraktni opis uporabniškega grafičnega vmesnika. To pomeni, da v tej datoteki le povemo, kateri grafični elementi so prisotni na zaslonu ter opis interakcij z njimi.

V tem poglavju bomo samo opisali pomen posamezne XML značke, kako

se te XML značke uporabljajo pri posameznem elementu in kako si sledijo po vrstnem redu, je podano z XML shemo, ki je priložena v Prilogi A.

- **HMIDescription** je koren AGUI XML opisa. Ima atribut name, s katerim povemo ime grafičnega zaslona, ki ga opisujemo. To ime mora biti enako imenu AGUI datoteke, hkrati pa je atribut name tudi referenca na ta zaslon, ki se uporablja, ko povemo, na kateri zaslon hočemo priti. Z atributom xmlns:xsi povemo, da bomo uporabljali XML shemo, atribut xsi:noNamespaceSchemaLocation pa pove ime XML sheme.

  Primer uporabe HMIDescription XML znacke:

  ```
  <HMIDescription name="EnostavniZaslon"
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:noNamespaceSchemaLocation="aguiSchema.xsd" >
  </HMIDescription>
  ```

- **GUIElement** je koren za XML opis grafičnega elementa. Primer grafičnega elementa je gumb. Ima en obvezen atribut name, s katerim podamo ime grafičnega elementa in se pozneje uporablja za referenco v RULE XML datoteki.

  Primer enostavnega grafičnega elementa:

  ```
  <GUIElement name="TestniGumb">
    <category>button</category>
  </GUIElement>
  ```

- **category** s to XML značko povemo, katere kategorije je grafični element. Trenutno podprte kategorije so:

  - *button* za gumbe;
  - *label* za labele;
  - *scrollList* za seznam;
  - *container* to je element, ki združuje grafične elemente v skupino;
  - *lineEdit* je vnosno polje;
  - *keyboard* za tipkovnico, ki je prikazana na zaslonu;
  - *comboBox* za kombinirano polje.

Primer kategorije:

```
<category>label</category>
```

- **testItems** ta XML značka je dodana le za predstavitev grafičnega elementa za sezname. S to značko povemo, koliko testnih elementov naj doda v seznam.

  Primer uporabe pri seznamu:

```
<GUIElement name="testniSeznam">
  <category>scrollList</category>
  <testItems>30</testItems>
</GUIElement>
```

- **textId** s to značko povemo, kateri tekst iz jezikovne datoteke naj prikaže v želenem grafičnem elementu.

  Primer uporabe:

```
<textId>11</textId>
```

- **text** uporablja se za prikaz teksta na želenem grafičnem elementu.

  Primer uporabe:

```
<text>Pozdravljen svet</text>
```

- **img** ta XML značka se uporablja za prikaz slike na želenem grafičnem elementu. Podamo le ime slike brez končnice. Podprte slike so png formata in se morajo nahajati v imgages imeniku.

  Primer uporabe:

```
<img>back</img>
```

- **register** s to z značko povemo, da se hočemo registrirati na nek dogodek nekega servisa. Primer takšnega dogodka je sprememba trenutno predvajane pesmi na MP3 predvajalniku.

  Primer registracije na dogodek:

```
<register>
  <onEvent name="SongChanged">
    <serviceClass>ENTERTAINMENT</serviceClass>
    <serviceName>MP3PlayerSim</serviceName>
    <returnValType>HString</returnValType>
    <invokeFun>setText</invokeFun>
  </onEvent>
</register>
```

- **onEvent** se uporablja znotraj register XML značke. Ima obvezen atribut name, s katerim povemo ime dogodka, na katerega se hočemo registrirati. Primer uporabe te XML značke je viden pri register XML znački.

- **serviceClass** s to XML značko povemo razred servisa, na katerega se hočemo nanašati. Primer uporabe te XML značke je viden pri register XML znački.

- **serviceName** s to XML značko povemo ime servisa, na katerega se nanašamo. Primer uporabe te XML značke je viden pri register XML znački.

- **data** se uporablja znotraj register XML značke za prenos kakšnih dodatnih podatkov. Ta XML značka lahko ima poljubno število atributov s poljubnimi imeni.

  Primer uporabe data XML značke za nastavitev velikosti slike iz navigacijske naprave:

```
<data width = "318" height = "238">
  <type>HNaviImgSize_t</type>
</data>
```

- **type** se uporablja znotraj data XML značke. S tem povemo, kakšnega tipa so podatki. Primer uporabe je viden pri data XML znački.

- **returnValType** s to XML značko povemo, kakšnega tipa je vrnjeni podatek, ki nam ga vrne servis. Ta XML značka se uporablja znotraj register XML značke. Primer uporabe te XML značke je viden pri register XML znački.

- **invokeFun** s to XML značko povemo, katero funkcijo hočemo poklicati, ko se zgodi nek dogodek. Primer uporabe te XML značke je viden pri register XML znački.

- **events** je korenska XML značka znotraj grafičnega elementa za dogodke. Ti dogodki so vezani na grafični element znotraj katerega se ta XML značka nahaja. Tipičen primer takšnega dogodka je klik na gumb.

  Primer uporabe:

```
<events>
  <event type="clicked">
    <action name="goToMainScreen">
      <type>screenChg</type>
```

```
        <serviceClass>GUI</serviceClass>
        <serviceName>GUI</serviceName>
        <param>MainScreen</param>
        <param>MOVE</param>
        <param>UP</param>
      </action>
    </event>
  </events>
```

- **event** ta XML značka se uporablja za opis določenega dogodka. Ima en obvezni atribut type, s katerim povemo, kakšnega tipa je dogodek. Lahko ima še dodatne atribute, s katerimi povemo, kakšnega tipa so vrnjeni podatki dogodka. Primer takšnega dogodka je, ko izberemo element na seznamu. Potem nam seznam sproži dogodek, da je bil element izbran in nam vrne tudi ID izbranega elementa. Primer uporabe te XML značke je viden pri events XML znački.

- **action** uporablja se znotraj event značke. S to XML značko povemo, katero akcijo hočemo izvršiti, ko se bo zgodil opisani dogodek. Ima tudi en obvezen atribut name, s katerim povemo ime akcije, ki jo hočemo izvršiti. Primer uporabe te XML značke je viden pri events XML znački.

- **type** ta XML značka se uporablja znotraj action XML značke. Z njo povemo, kakšnega tipa je akcija, ki jo hočemo izvesti. Trenutno podprti tipi akcij so sledeči:

  - screenChg pove, da gre za prehod na drugi zaslon;
  - function pove, da gre za klic funkcije podanega servisa;
  - spawn pove, da bomo priklicali nek grafični element. Trenutno je podprt le prikaz pogovornega okna (ang. dialog box).

  Primer uporabe te XML značke je viden pri events XML znački.

- **param** ta XML značka se uporablja za podajanje dodatnih parametrov funkciji oz. akciji, ki jo hočemo izvesti. Če je želena akcija tipa screenChg, potem prvi param določa ime zaslona, ki ga hočemo prikazati, drugi param določa tip animacije, tretji param pa smer animacije. Primer uporabe te XML značke je viden pri events XML znački.

- **dialogBox** ta XML značka se lahko uporabi znotraj register XML značke in pri tem mora biti akcija tipa spawn. Uporablja pa se za prikaz pogovornega okna.

Primer uporabe:

```
<dialogBox>
  <type>information</type>
  <title>Processing</title>
  <textId>11</textId>
  <display>param1</display>
  <buttons>OK</buttons>
</dialogBox>
```

- **type** ta XML značka znotraj dialogBox določa tip pogovornega okna. Tipi, ki so na voljo, so sledeči:
  - critical
  - information
  - question
  - warning
  - about

  Primer uporabe te XML značke je viden pri dialogBox XML znački.

- **title** s to XML značko določimo naslov pogovornega okna. Primer uporabe te XML značke je viden pri dialogBox XML znački.

- **display** s to XML značko lahko prikažemo vrednost podatka, ki nam ga je vrnil dogodek. Podprta tipa sta int32 ter HString. Primer uporabe te XML značke je viden pri dialogBox XML znački.

- **buttons** s to XML značko povemo, katere gumbe hočemo prikazati na pogovornem oknu. Primer uporabe te XML značke je viden pri dialogBox XML znački.

- **returnVar** uporablja se znotraj action XML značke. S to XML značko povemo, v katero začasno spremenljivko si želimo shraniti vrnjeni podatek.

  Primer uporabe:

```
<event type="clicked">
  <action name="Eject">
    <type>functionRet</type>
    <serviceClass>ENTERTAINMENT</serviceClass>
    <serviceName>MP3PlayerSim</serviceName>
    <returnVar>ejectVal</returnVar>
```

```
    </action>
  </event>
```

- **showFunReturn** ta XML značka se uporablja za prikaz vrednosti neke spremenljivke, ki smo si jo začasno shranili. Uporabno je predvsem za prikaz kakšnih podatkov, ki jih dobimo iz drugega grafičnega elementa ali pa iz nekega servisa. Pred uporabo spremenljivke moramo to spremenljivko definirati z XML značko variable.

  Primer uporabe:

```
<showFunReturn>
  <showVariable name="ejectVal">
    <if status = "PROCESSING">
      <text>Ejecting CD</text>
    </if>
    <if status = "FERROR">
      <text>Ejecting Failed!</text>
    </if>
    <else>
      <text>CD is ejected</text>
    </else>
  </showVariable>
</showFunReturn>
```

- **showVariable** uporablja se znotraj showFunReturn XML značke. Ima en obvezen atribut name, s katerim povemo ime spremenljivke, katere vrednost hočemo prikazati. Primer uporabe te XML značke je viden pri showFunReturn XML znački.

- **if** ta XML značka se uporablja za pogojne stavke pri prikazu vrednosti spremenljivke. Atribut pri tem je pogoj, ki ga preverjamo. Primer uporabe te XML značke je viden pri showFunReturn XML znački.

- **else** se uporablja le skupaj z if XML značko. Stvar znotraj else XML značke se izvede le, če pogoj v if XML znački ne uspe. Primer uporabe te XML značke je viden pri showFunReturn XML znački.

- **variable** s to XML značko definiramo začasno spremenljivko. Obvezni atribut je name, s katerim označuje ime spremenljivke. Znotraj vsebuje type XML značko, s katero povemo, kakšnega tipa je začasna spremenljivka.

Primer uporabe:

```
<variable name="ejectVal">
  <type>int32</type>
</variable>
```

- **contains** ta XML značka označuje, da grafični element vsebuje druge grafične elemente. Element mora biti kategorije container. Vsebovani grafični elementi imajo potem pozicijo relativno na pozicijo starša. Če premaknemo starša, se skupaj z njim premaknejo vsi otroci. To je uporabno predvsem takrat, ko hočemo povedati, da določeni grafični elementi spadajo skupaj.

  Primer uporabe:

```
<GUIElement name="controllsCon">
  <category>container</category>
  <contains>

    <GUIElement name="playBut">
      <category>button</category>
    </GUIElement>

    <GUIElement name="stopBut">
      <category>button</category>
    </GUIElement>

    <GUIElement name="ejectBut">
      <category>button</category>
    </GUIElement>

  </contains>
</GUIElement>
```

5.3 RULE XML notacija

V RULE XML datoteki je opis pozicije, velikosti ter ostalih lastnosti grafičnih elementov. Ime RULE XML datoteke mora biti enako, kot je ime, pripadajoče AGUI XML datoteke.

XML značke, ki so podprte v RULE XML datoteki, so sledeče:

- **rule** je korenska XML značka za RULE XML datoteko. Ima atribut name, ki določa ime zaslona, za katerega velja opis. Ime mora biti enako, kot je ime datoteke. Z atributom xmlns:xsi povemo, da bomo uporabljali

XML shemo, atribut xsi:noNamespaceSchemaLocation pa pove ime XML sheme.

Primer uporabe:

```
<rule name="PlayerScreen"
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:noNamespaceSchemaLocation="ruleSchema.xsd" >
</rule>
```

- **onType** je korenska XML značka za opis lastnosti določenega tipa grafičnega elementa. Ima obvezni atribut name, s katerim povemo, za kateri tip grafičnega elementa veljajo podane lastnosti. Prevajalnik gre najprej iskat lastnosti za gumb glede na njegovo ime. Če katera lastnost ni podana posebej, gre gledat lastnosti, ki so podane pri tipu. Tako lahko na primer podamo privzeto velikost gumbov na določenem zaslonu.

  Primer uporabe:

```
<onType name="button">
  <size>
    <width>140</width>
    <height>50</height>
  </size>
  <imgProperty>
    <pos>
      <xCor>7</xCor>
      <yCor>10</yCor>
    </pos>
    <size>
      <height>30</height>
      <width>30</width>
    </size>
  </imgProperty>
  <textProperty>
    <size>12</size>
    <type>Times New Roman</type>
  </textProperty>
</onType>
```

- **pos** XML značka za opis pozicije grafičnega elementa. Vsebuje dve XML znački, in sicer *xCor*, ki določa x koordinato, ter *yCor*, s katero določimo y koordinato.

  Primer uporabe:

```
<pos>
```

```
    <xCor>2</xCor>
    <yCor>20</yCor>
  </pos>
```

- **size** ta XML značka opiše velikost grafičnega elementa. Vsebuje dve XML znački, in sicer *width* za določitev širine ter *height* višine grafičnega elementa. Primer uporabe te XML značke je viden pri onType XML znački.

- **imgProperty** s to XML značko določimo lastnosti sliki, ki jo prikazuje grafični element. Sliki lahko določimo pozicijo. To naredimo s *pos* XML značko ter velikost, ki pa jo določimo s *size* XML značko. Ima tudi opcijski atribut name, s katerim lahko določimo, da specificirane lastnosti veljajo le za določeno sliko. Primer uporabe te XML značke je viden pri onType XML znački.

- **textProperty** s to XML značko določimo lastnosti tekstu, ki ga prikazuje grafični element. Tekstu lahko določimo velikost. To storimo s *size* XML značko ter vrsto pisave, ki pa jo določimo s *type* XML značko. Primer uporabe te XML značke je viden pri onType XML znački.

- **onGUIElement** je korenska XML značka za opis lastnosti določenega grafičnega elementa. Ima obvezni atribut name, s katerim povemo, za kateri grafični element veljajo opisane lastnosti.

  Primer uporabe:

```
<onGUIElement name="changeKBbut">
  <type>label</type>
  <pos>
    <xCor>157</xCor>
    <yCor>0</yCor>
  </pos>
  <size>
    <width>79</width>
    <height>23</height>
  </size>
</onGUIElement>
```

- **type** ta XML značka je uporabljena znotraj onGUIElement XML značke. Določa, kakšnega tipa je grafični element. Primer uporabe te XML značke je viden pri onGUIElement XML znački.

- **style** s to XML značko lahko določimo stil grafičnega elementa. Trenutno je podprta le prosojnost.

  Primer uporabe:

  ```
  <style>
    <transparency>50</transparency>
  </style>
  ```

- **transparency** ta XML značka se uporablja znotraj style XML značke. Z njo določimo prosojnost nekega grafičnega elementa. Številka, ki jo podamo, pove, kolikšen odstotek je grafični element prosojen. Primer uporabe te XML značke je viden pri style XML znački.

Poglavje 6

Navodila za uporabo prevajalnika

Za lažje delo s prevajalnikom je na voljo grafični vmesnik. Grafični vmesnik prevajalnika je sestavljen iz enega okna. Slika 6.1 prikazuje grafični vmesnik.

Kratko navodilo, kako uporabljamo HXParser program:

- V vnosno polje *GUI Lib* vpišemo končnico uporabljene grafične knjižnice. Če smo grafično knjižnico definirali brez posebne končnice, to polje pustimo prazno.

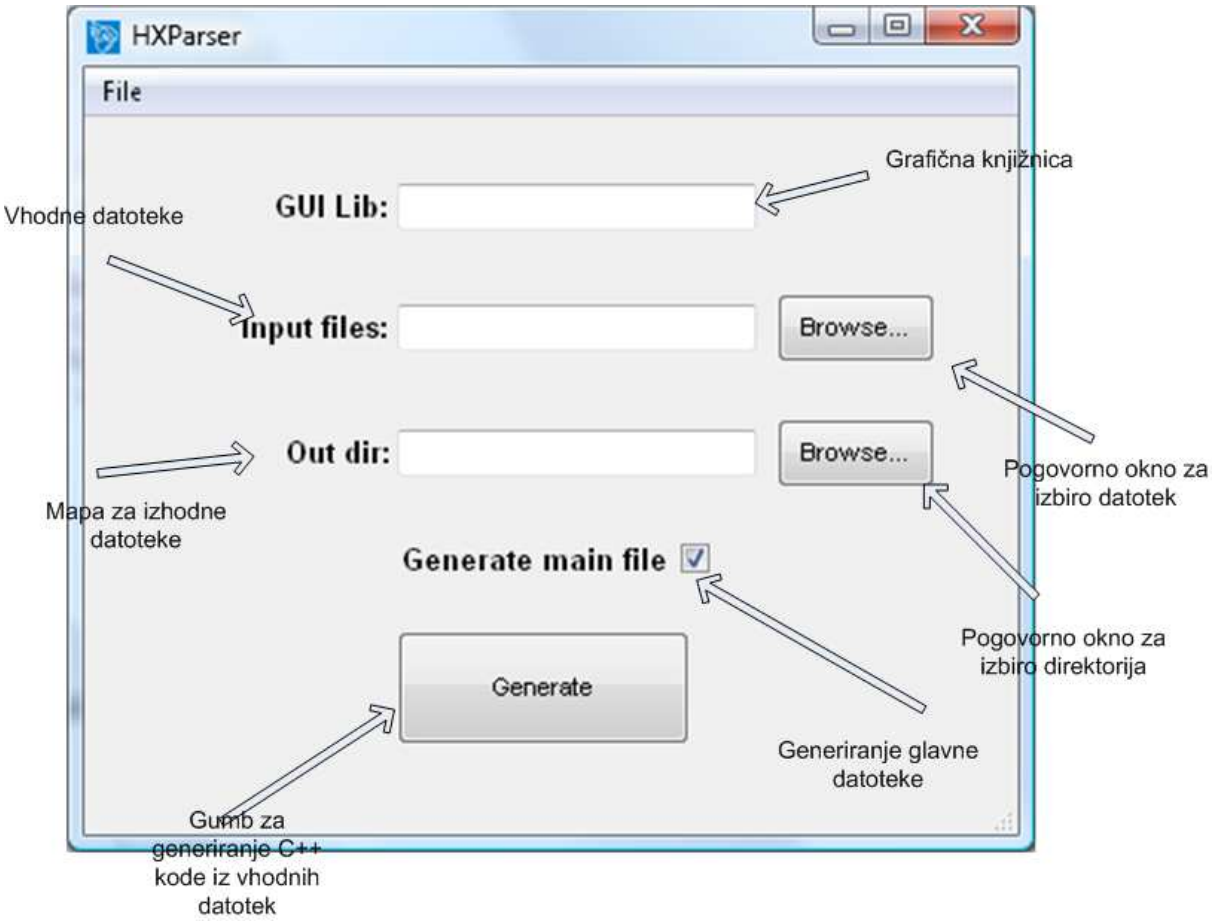
- V vnosno polje *Input files* vpišemo vhodne AGUI ter RULE XML datoteke. Vnesemo vse datoteke hkrati, ki jih hočemo prevesti. Datoteke ločimo z ";" ločilom. Najlažje pa to storimo tako, da kliknemo na gumb Browse in v pogovornem oknu za izbiro datotek izberemo želene datoteke.

- V vnosno polje *Out dir* pa vnesemo pot do mape, kamor želimo shraniti izhodne datoteke. To lahko storimo z direktnim vnosom ali preko pogovornega okna za izbiro mape. To pogovorno okno prikličemo tako, da kliknemo na gumbek Browse.

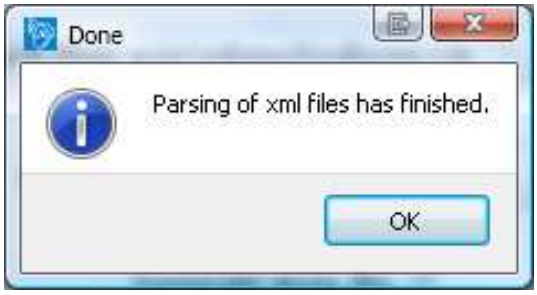
- *Generate main file* omogočimo le, če imamo zbrane vse vhodne datoteke vseh zaslonov, iz katerih hočemo kreirati C++ kodo. Če pa smo popravljali samo eno datoteko in imamo main datoteko že kreirano, te opcije ne označimo.

- Na koncu, ko smo nastavili vse parameter, samo kliknemo gumb Generate in počakamo, da prevajalnik konča z delom. Ob uspešnem prevajanju

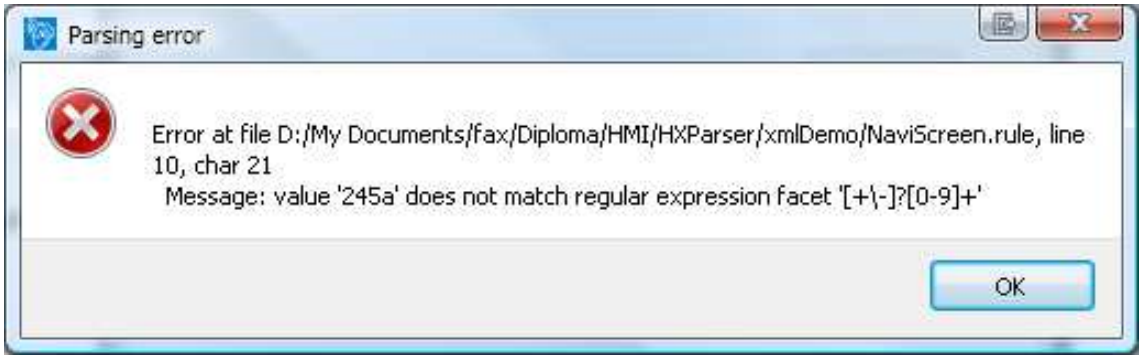
dobimo sporočilo, da je prevajalnik uspešno končal z delom. V nasprotnem primeru pa dobimo sporočilo o napaki. Slika 6.2 prikazuje pogovorno okno ob uspešnem zaključku prevajanja. Slika 6.3 pa prikazuje pogovorno okno o napaki med prevajanjem.



Slika 6.1: Grafični vmesnik XML prevajalnika



Slika 6.2: Pogovorno okno ob uspešnem končanju prevajanja



Slika 6.3: Pogovorno okno o napaki med prevajanjem

Poglavje 7

Primeri XML opisa grafičnega vmesnika

Kot za primer si oglejmo eno enostavnejšo aplikacijo z enim zaslonom. Na tem zaslonu bo viden gumb za zaprtje aplikacije ter seznam, napolnjen s testnimi elementi. Ko uporabnik izbere želeni element na seznamu, se odpre pogovorno okno s sporočilom o izbranem elementu.

7.1 XML izvorna koda primera

XML izvorna koda AGUI datoteke:

```
<?xml version='1.0' encoding='UTF-8'?>

<HMIDescription name="MainScreen"
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:noNamespaceSchemaLocation="aguiSchema.xsd" >

  <GUIElement name="exitBut">
    <category>button</category>
    <img>exit</img>
    <events>
      <event type="clicked">
        <action name="goToExitScreen">
          <type>screenChg</type>
          <serviceClass>GUI</serviceClass>
          <serviceName>GUI</serviceName>
          <param>EXIT</param>
        </action>
```

```
      </event>
    </events>
  </GUIElement>

  <GUIElement name="scrollList">
    <category>scrollList</category>
    <testItems>30</testItems>
    <events>
      <event type="selectedItem" param1="int32">
        <action name = "showDialogBox">
        <type>spawn</type>
        <serviceClass>GUI</serviceClass>
        <serviceName>GUI</serviceName>
        <dialogBox>
          <type>information</type>
          <title>Processing</title>
          <textId>11</textId>
          <display>param1</display>
           <!-- to add more buttons write OK|CANCEL|....  -->
          <buttons>OK</buttons>
        </dialogBox>
        </action>
      </event>
    </events>
  </GUIElement>

</HMIDescription>
```

XML izvorna koda RULE datoteke:

```
<?xml version='1.0' encoding='UTF-8'?>

<rule name="MainScreen"
   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xsi:noNamespaceSchemaLocation="ruleSchema.xsd" >

  <onGUIElement name="exitBut">
    <type>button</type>
    <pos>
      <xCor>270</xCor>
      <yCor>10</yCor>
    </pos>
    <size>
      <height>30</height>
      <width>30</width>
    </size>
    <imgProperty>
```

```
      <pos>
        <xCor>5</xCor>
        <yCor>5</yCor>
      </pos>
      <size>
        <height>20</height>
        <width>20</width>
      </size>
    </imgProperty>
  </onGUIElement>

  <onGUIElement name="scrollList">
    <type>scrollList</type>
    <pos>
      <xCor>30</xCor>
      <yCor>5</yCor>
    </pos>
    <size>
      <width>260</width>
      <height>190</height>
    </size>
  </onGUIElement>

</rule>
```

7.2 C++ izvorna koda prevedenih XML datotek

main datoteka:

```
/*****************************************************************************
*
* This file is generated with HXParser. Any changes made in this
    file
* will be lost upon next parsing with HXParses.
*
*****************************************************************************/

#include "BaselibFunctions.h"
#include "HGuiDefines.h"
#include "HGuiQT.h"
#include "HControlPanelQT.h"
#include "Delegate.h"
#include "HQueue.h"
#include "MainScreen.h"
```

```
static bool end = false;

/*************************************************************************
*
* Working thread for processing the messages from services
*
*************************************************************************/
THREADFUN ProcessMsg (void * )
{
  Delegate *tmpDg;
  HQueue *msgQueue;

  msgQueue = getHandleToMsgQueue();

  while(!end)
  {

    while(  (tmpDg = (Delegate *)msgQueue->Pop()) != NULL )
    {
      tmpDg->Invoke();
    }
    SLEEP(10);
  }

  return 0;
}
/*************************************************************************
*
*       MAIN FUINCTION
*
*************************************************************************/
int32 main(int32 argc, bbyte *argv[])
{
  HString nameScr;
  /*************************************************************************
  *
  *       GUI INIT
  *
  *************************************************************************/
  HGuiQT gui(argc, argv);

  /*************************************************************************
  *
  *       SERVICES INIT
  *
  *************************************************************************/
  initBaselib();
```

```
/*************************************************************************
*
*        GUI CONSTRUCTION
*
*************************************************************************/
HControlPanelQT controlPanel;

MainScreen MainScr;

HSize size(328, 238);
controlPanel.setSize(size);

nameScr = "MainScreen";
controlPanel.addItem(&MainScr, nameScr);

nameScr = "MainScreen";
controlPanel.setMainItem(nameScr);

HCONNECT(&MainScr, SIGNAL(goToScreen(HAnimateScreen_t)),
    &controlPanel, SLOT(changeScreen(HAnimateScreen_t)));

controlPanel.show();

/*************************************************************************
*
* Spawning thread for processing the messages
*
*************************************************************************/
P_HANDLE thread = NULL;
THREADIDTYP threadID = 0;
thread = (P_HANDLE)SPAWNTHREAD(NULL,0,ProcessMsg,NULL,0,
    threadID);

/*************************************************************************
*
*        GET IN TO GUI LOOP THREAD
*
*************************************************************************/
gui.exec();

/*************************************************************************
*
*        CLOSING PROGRAM
*
*************************************************************************/
end = true;
```

```
  WAITFORTHREAD(thread,INFINITE);
  CLOSETHREAD(thread);

  closeBaselib();

  return 0;

}
```

MainScreen.h datoteka:

```
/*********************************************************************
*
* This file is generated with HXParser. Any changes made in this
    file
* will be lost upon next parsing with HXParses.
*
*********************************************************************/


#ifndef GEN_MAINSCREEN_H
#define GEN_MAINSCREEN_H

#include "HScreenQT.h"
#include "iMainService.h"
#include "HButtonQT.h"
#include "HLabelQT.h"
#include "HScrollListQT.h"
#include "HDialogBoxQT.h"

class MainScreen : public HScreenQT
{
  Q_OBJECT
public:
  MainScreen(void *container = 0);
  ~MainScreen(void);

private slots:
  void goToExitScreenexitBut();
  void showDialogBoxscrollList(int32 param1);

private:


  HButtonQT *exitBut;
  HLabelQT *exit;
```

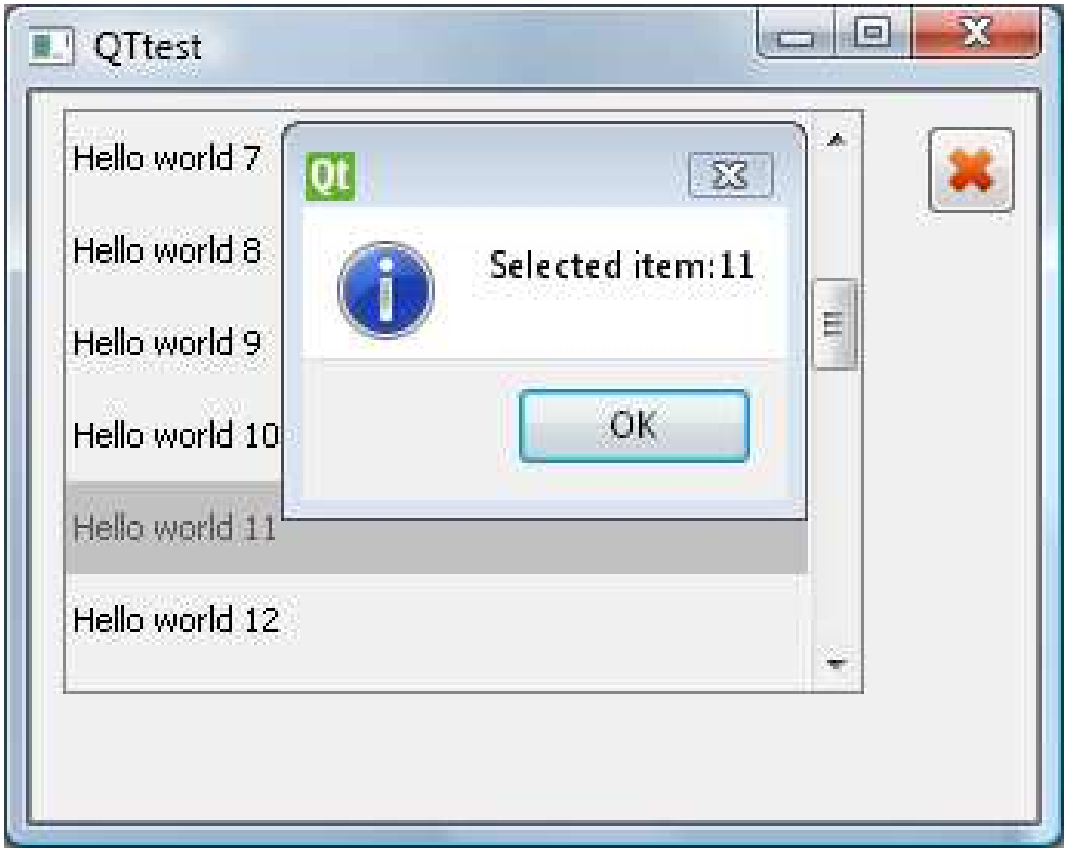
```
  HScrollListQT *scrollList;

  HDialogBoxQT dialogBox;
};

#endif
```

MainScreen.cpp datoteka:

```
/*****************************************************************************
*
* This file is generated with HXParser. Any changes made int this
    file
* will be lost upon next parsing with HXParses.
*
*****************************************************************************/

#include "MainScreen.h"

#include "HGuiDefines.h"
#include "MsgQueueDelegate.h"
#include "BaselibFunctions.h"
#include "HControlPanelQT.h"
#include "HSmartPointer.h"

MainScreen::MainScreen(void *container)
: HScreenQT(container)
{
  exitBut = new HButtonQT(this);
  scrollList = new HScrollListQT(this, 30);
  exit = new HLabelQT(exitBut);

  HPoint pos0(290, 10);
  exitBut->setPosition(pos0);
  HSize size0(30, 30);
  exitBut->setSize(size0);
  HString imgPath0exit("../images/exit.png");
  exit->showPicture(imgPath0exit);
  HPoint imgPos0exit(5, 5);
  exit->setPosition(imgPos0exit);
  HSize imgSiz0exit(20, 20);
  exit->setSize(imgSiz0exit);
  HCONNECT( exitBut, HSIGNAL(clicked()), this,
      HSLOT(goToExitScreenexitBut()) );

  HPoint pos1(10, 5);
  scrollList->setPosition(pos1);
```

```
  HSize size1(260, 190);
  scrollList->setSize(size1);
  HCONNECT( scrollList, HSIGNAL(selectedItem(int32)), this,
      HSLOT(showDialogBoxscrollList(int32)) );

}

void MainScreen::goToExitScreenexitBut( )
{
  HAnimateScreen_t animScreen;
  animScreen.aniType = MOVE;
  animScreen.direction = LEFT;
  animScreen.screenName = "EXIT";

  emit goToScreen(animScreen);
}

void MainScreen::showDialogBoxscrollList(int32 param1 )
{
  HString dgBtitStshowDialogBox = "Processing";
  HString dgBtxtpStshowDialogBox = CONTROLPANEL->getTextById(11);
  dgBtxtpStshowDialogBox.append(param1);
  dialogBox.information(this, dgBtitStshowDialogBox,
      dgBtxtpStshowDialogBox);
}

MainScreen::~MainScreen(void)
{
}
```

7.3 Posnetek aplikacije



Slika 7.1: Posnetek aplikacije opisane z XML jezikom

Poglavje 8

Zaključek

V okviru diplomskega dela je realiziran sistem za generiranje enotnega uporabniškega vmesnika za podporo različnih naprav. Ta sistem omogoča enostavno dodajanje novih naprav. Ob tem pa so potrebne minimalne spremembe v obstoječem uporabniškem vmesniku. Za enostavnejše in razumljivejše kreiranje uporabniškega vmesnika je bil razvit XML jezik. S tem XML jezikom enostavno opišemo uporabniški vmesnik ter vse akcije, ki se dogajajo na tem uporabniškem vmesniku. Ob tem je bil še razvit prevajalnik, ki ta XML jezik prevede v C++ izvorno kodo. Prevedena koda je dokaj optimalna, saj so ti uporabniški vmesniki v večini namenjeni za vgradne sisteme. Kot vemo, pa vgradni sistemi nimajo veliko razpoložljivih resursov.

Možnost optimizacije diplomske naloge leži predvsem v prevajalniku, ki prevaja XML jezik v C++ izvorno kodo, saj je trenutno narejen tako, da gre večkrat čez XML kodo, saj je tako bilo lažje kreirati berljivo C++ kodo in ob tem obdržati optimalnost. Razširiti je možno tudi XML jezik, saj trenutno podpira le osnovne grafične gradnike ter osnovne nastavitve lastnosti teh grafičnih elementov.

Kljub odprtim poglavjem, ki zadevajo nadaljnji razvoj diplomske naloge, sistem deluje zadovoljivo, saj prikaže, da zamišljen koncept podpore različnim napravam deluje v praksi.

Dodatek A

XML shema

A.1 XML shema za AGUI datoteko

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

  <xs:element name="HMIDescription">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="variable" type="variableType"
            minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="GUIElement" type="GUIElementType"
            minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:complexType name="GUIElementType">
    <xs:sequence>
      <xs:element name="category" type="xs:string" />
      <xs:element name="testItems" type="xs:integer"
          minOccurs="0" maxOccurs="1" />
      <xs:element name="textId" type="xs:integer" minOccurs="0"
          maxOccurs="1" />
      <xs:element name="text" type="xs:string" minOccurs="0"
          maxOccurs="1" />
      <xs:element name="img" type="xs:string" minOccurs="0"
          maxOccurs="1" />
      <xs:element name="register" type="registerType"
          minOccurs="0" maxOccurs="unbounded" />
```

```
    <xs:element name="events" type="eventsType" minOccurs="0"
        maxOccurs="1"/>
    <xs:element name="showFunReturn" type="showFunReturnType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="contains" type="containsType"
        minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="showFunReturnType">
  <xs:sequence>
    <xs:element name="showVariable" type="showVariableType"
        minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="showVariableType">
  <xs:sequence>
    <xs:element name="if" type="ifType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:element name="else" type="elseType" minOccurs="0"
        maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="containsType">
  <xs:sequence>
    <xs:element name="GUIElement" type="GUIElementType"
        minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="registerType">
  <xs:sequence>
    <xs:element name="onEvent" type="onEventType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="onEventType">
  <xs:sequence>
    <xs:element name="serviceClass" type="xs:string"/>
    <xs:element name="serviceName" type="xs:string" />
    <xs:element name="data" type="dataRegEvType" minOccurs="0"
        maxOccurs="1"/>
    <xs:element name="returnValType" type="xs:string"
```

```
        minOccurs="0" maxOccurs="1" />
      <xs:element name="invokeFun" type="xs:string" minOccurs="0"
        maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required" />
  </xs:complexType>

  <xs:complexType name="dataRegEvType">
    <xs:sequence>
      <xs:element name="type" type="xs:string" minOccurs="1"
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute processContents="lax" />
  </xs:complexType>

  <xs:complexType name="eventsType">
    <xs:sequence>
      <xs:element name="event" type="eventType" minOccurs="1"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="eventType">
    <xs:sequence>
      <xs:element name="action" type="actionType" minOccurs="1"
        maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" use="required" />
    <xs:anyAttribute processContents="lax" />
  </xs:complexType>

  <xs:complexType name="actionType">
    <xs:sequence>
      <xs:element name="type" type="xs:string" />
      <xs:element name="serviceClass" type="xs:string" />
      <xs:element name="serviceName" type="xs:string" />
      <xs:element name="returnVar" type="xs:string" minOccurs="0"
        maxOccurs="1" />
      <xs:element name="param" type="xs:string" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element name="dialogBox" type="dialogBoxType"
        minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="if" type="ifType" minOccurs="0"
        maxOccurs="unbounded" />
      <xs:element name="else" type="elseType" minOccurs="0"
        maxOccurs="1" />
    </xs:sequence>
```

```
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:complexType name="dialogBoxType">
    <xs:sequence>
      <xs:element name="type" type="xs:string"/>
      <xs:element name="title" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="textId" type="xs:integer" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="text" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="display" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="buttons" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ifType">
    <xs:sequence>
      <xs:element name="text" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="param" type="xs:string" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="elseType">
    <xs:sequence>
      <xs:element name="text" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="param" type="xs:string" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="variableType">
    <xs:sequence>
      <xs:element name="type" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>

</xs:schema>
```

A.2 XML shema za RULE datoteko

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

  <xs:element name="rule">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="onType" type="onTypeType" minOccurs="0"
            maxOccurs="unbounded"/>
          <xs:element name="onGUIElement" type="guiElementType"
              minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="onTypeType">
    <xs:sequence>
      <xs:element name="pos" type="posType" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="size" type="sizeType" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="imgProperty" type="imgPropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="textProperty" type="textPropertyType"
          minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:complexType name="guiElementType">
    <xs:sequence>
      <xs:element name="type" type="xs:string" />
      <xs:element name="pos" type="posType" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="size" type="sizeType" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="imgProperty" type="imgPropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="textProperty" type="textPropertyType"
          minOccurs="0" maxOccurs="1"/>
      <xs:element name="style" type="styleType" minOccurs="0"
          maxOccurs="1"/>
    </xs:sequence>
```

```
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:complexType name="sizeType">
    <xs:all>
      <xs:element name="width" type="xs:integer" />
      <xs:element name="height" type="xs:integer" />
    </xs:all>
  </xs:complexType>

  <xs:complexType name="styleType">
    <xs:all>
      <xs:element name="transparency" type="xs:integer" />
    </xs:all>
  </xs:complexType>

  <xs:complexType name="posType">
    <xs:all>
      <xs:element name="xCor" type="xs:integer" />
      <xs:element name="yCor" type="xs:integer" />
    </xs:all>
  </xs:complexType>

  <xs:complexType name="imgPropertyType">
    <xs:sequence>
      <xs:element name="pos" type="posType" minOccurs="0"
          maxOccurs="1"/>
      <xs:element name="size" type="sizeType" minOccurs="0"
          maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" /> <!-- name
        attribute is optional -->
  </xs:complexType>

  <xs:complexType name="textPropertyType">
    <xs:sequence>
      <xs:element name="size" type="xs:integer" minOccurs="0"
          maxOccurs="1" />
      <xs:element name="type" type="xs:string" minOccurs="0"
          maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

Slike

1.1 Slika celotnega sistema 5

2.1 Grafični uporabniški vmesnik GNOME okolja 7
2.2 Sistem Xerox Alto 8
2.3 Sistem Xerox STAR 8
2.4 Sistem Xerox Machintosh 9

3.1 Enotni vmesnik servisov 11

4.1 Zgradba enostavnega prevajalnika 21
4.2 Potek prevajanja XML datotek 23

6.1 Grafični vmesnik XML prevajalnika 39
6.2 Pogovorno okno ob uspešnem končanju prevajanja 40
6.3 Pogovorno okno o napaki med prevajanjem 40

7.1 Posnetek aplikacije opisane z XML jezikom 49

Tabele

3.1 Tabela modelov podatkovnih tipov prevajalnikov. Velikosti posameznih podatkovnih tipov so podani v bit-ih. 17

4.1 Tabela podprtih operacijskih sistemov in pripadajočih prevajalnikov knjižnice Xerces 26

Literatura

[1] Christoph Ainhauser, Reinhard Stolle, Jürgen Steurer "Enabling semantic interoperation of dynamic plug-in services," München, Germany.

[2] Andreas Hildisch, Jürgen Steurer, Reinhard Stolle "HMI generation for plug-in services from semantic descriptions" *Workshop on Software Engineering for Automotive Systems*, Minneapolis, MN, maj 2007.

[3] (2009) GUI Definition. Dostopno na: http://www.linfo.org/gui.html

[4] (2009) The Art of Unix Usability. Dostopno na: http://www.catb.org/ esr/writings/taouu/html/ch02s05.html

[5] Wilbert O. Galitz, "The Essential Guide to User Interface Design," *Wiley*, Third Edition, chap. 1, str. 7–9, 2007.

[6] Andrew W. Appel, Jens Palsberg, "Modern Compiler Implementation in Java," *Cambridge University Press*, Second Edition, 2002.

[7] (2009) Apache: Xerces-C++. Dostopno na: http://xerces.apache.org/xerces-c/

[8] (2009) W3C: Extensible Markup Language. Dostopno na: http://www.w3.org/XML/

[9] (2009) 64-Bit Programming Models: Why LP64? Dostopno na: http://www.unix.org/version2/whatsnew/lp64_wp.html