Model programske podrške sa udaljenom korisničkom spregom za ugrađene sisteme

Milan Z. Bjelica, Ištvan Papp, Dragan Samardžija, Zoran Jovanović, *Fakultet tehničkih nauka, Novi Sad*

***Sadržaj* — U svetu ugrađenih uređaja za specifične namene, postizanje prenosivosti programske podrške je izuzetno težak zadatak. Takođe, fizička arhitektura se menja i poboljšava, te prilagođavanje programske podrške uvek predstavlja izazov. Ovaj rad predstavlja model prenosive programske podrške za ugrađene sisteme, sa naglaskom na razdvajanju funkcionalnosti (jezgra) od prezentacije (korisničke sprege). Model omogućuje jednostavno izmeštanje korisničke sprege, kao dela programske podrške koji je veoma teško učiniti prenosivim na udaljene uređaje različitih arhitektura uz zadržavanje funkcionalnosti. U okviru rada predstavljen je odgovarajući komunikacioni protokol ka udaljenoj korisničkoj sprezi i prikazan jedan primer realizacije sistema koji koristi predloženi model.**

***Ključne reči* — prenosivost, skalabilnost, terminal, ugrađeni sistem, korisnička sprega.**

I. UVOD

JEDAN od glavnih problema programske podrške za ugrađene sisteme je mogućnost njenog izvršavanja isključivo na uređajima kojima je namenjena. Prenosivost programske podrške, odnosno njena ponovna iskoristivost (*reusability*) i nezavisnost od fizičke arhitekture predstavljaju nove zahteve koje nije jednostavno zadovoljiti. Ovi zahtevi nastaju kao posledica veoma kratkog vremena između dve verzije uređaja koji se pojavljuju na tržištu (šest meseci do godinu dana), što je neophodno zbog održavanja konkurentne prednosti. Dalje, eksponencijalni tehnološki napredak omogućava različita poboljšanja fizičke arhitekture, što se danas najčešće odnosi na smanjenje potrošnje, manje dimenzije uređaja i poboljšanje performansi.

Vreme u okviru razvoja uređaja koje se dodeljuje razvoju programske podrške ima mali udeo u odnosu na ukupno vreme razvoja. Veliki deo vremena se dodeljuje razvoju prototipa sa nepotpunom funkcionalnošću (*mockup*) kako bi se ispitala upotrebljivost i potvrdio koncept u odnosu na krajnje korisnike. Ispitivanje na nivou sistema, korisničko ispitivanje i ispitivanje konkretnih komponenti fizičke arhitekture (npr. u cilju smanjivanja potrošnje energije), pa čak i izrada kućišta uređaja predstavljaju aktivnosti kojima se posvećuje veća pažnja, jer su presudne za subjektivnu ocenu kvaliteta proizvoda i povećanje prodaje. Tako se od programske podrške očekuje da bude prilagodljiva i skalabilna u svetlu izmena koje donose neprestane iteracije u okviru korisničkih testova. Iznenadna izmena delova fizičke arhitekture zbog potrebe za smanjenjem troškova izrade takođe zahteva programsku podršku koja će se tome lako prilagoditi. Kratko vreme razvoja nameće i zahtev da se programska podrška razvija paralelno sa razvojem fizičke arhitekture, te da je vreme potrebno za oživljavanje (*bringup*) sistema kada se izrade prvi primerci fizičke arhitekture minimalno, ako ne i nulto! Zbog toga se prenosivost već podrazumeva kao implicitna osobina programske podrške.

U ovom radu su identifikovane osnovne prepreke zahtevima prenosivosti i ponovne iskoristivosti programske podrške za ugrađene sisteme. Predstavljen je model programske podrške zasnovan na *POSIX* standardu [1], koji predlaže mehanizam odvajanja funkcionalnosti od korisničke sprege i jednostavnu zamenu blokova programske podrške (*plug-in*). Konačno, dat je primer upotrebe modela u okviru realizacije kontrolera za automatizaciju domaćinstva.

II. OSNOVNE PRETPOSTAVKE

Za razliku od aplikacija za opštenamenske sisteme, aplikacije za ugrađene sistemime se izvršavaju u „negostoljubivoj“ sredini određenoj fizičkom arhitekturom. Osnovne prepreke razvoju prenosivih i ponovno iskoristivih aplikacija na ugrađenim sistemima su (1) široka paleta procesora za specifične namene (razlike u arhitekturi, preciznosti, broju bita, alatima za razvoj), (2) različiti blokovi fizičke arhitekture za specifične namene (grafički podsistem, podsistem za obradu slike i zvuka, prihvatna i predajna logika) kojima se upravlja na nestandardan način.

Gornje prepreke izazvane su (1) zahtevima rada u realnom vremenu što uzrokuje oskudan broj međuslojeva programske podrške (kao što je operativni sistem) koji sakrivaju instrukcije samog procesora od aplikacije, a uzrokuje i izmeštanje obrade na posebne blokove fizičke arhitekture, po potrebi; (2) zahtevom što niže cene konačnog uređaja, što dovodi do korišćenja fizičke arhitekture slabijih performansi; (3) zahtevom korišćenja arhitekture koja ne obavlja suvišne funkcije (što znači i veću potrošnju energije) koje bi eventualno doprinele uopštavanju okruženja i njegovoj prilagodljivosti prenosivoj aplikaciji.

Ovaj rad je delimično finansiran od Ministarstva za nauku Republike Srbije, projekat 11005, od 2008. god.

Milan Z. Bjelica, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Trg Dositeja Obradovića 6, 21000 Novi Sad, Srbija (telefon: 381-21-4801139; faks: 381-21-450721; e-mail: milan.bjelica@rt-rk.com).

Ištvan Papp, e-mail: istvan.papp@rt-rk.com

Dragan Samardžija, e-mail: dragan.samardzija@rt-rk.com

Zoran Jovanović, e-mail: zoran.jovanovic@rt-rk.com

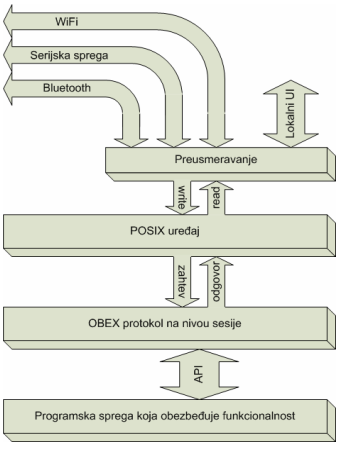
Većina uređaja potrošačke elektronike uključuje neku vrstu korisničke sprege (tasteri, različiti ulazni uređaji, različiti displeji za prikaz stanja ili rezultata rada). S obzirom na raznolikost korisničkih sprega, uzrokovanih pre svega faktorima kao što su namena uređaja, dimenzije, cena i sl, ove korisničke sprege su prva velika prepreka prenosivosti aplikacije. Funkcionalnost uređaja koja se ne odnosi na korisničku spregu, kao što je prijem i obrada signala, komunikacioni protokol, reprodukcija zvuka i sl. najčešće se postiže tako što se ovi zadaci povere ugrađenom procesoru. Razlike u arhitekturama procesora predstavljaju drugu prepreku. Međutim, primetno je da u većini slučajeva postoji prevodilac za programski jezik C.

U nastavku rada, izložen je model koji dve navedene prepreke tretira odvojeno i uvodi podelu između dela programske podrške koja obezbeđuje funkcionalnost i dela koji obezbeđuje korisničku spregu. Korisnička sprega se jednostavno priključuje i menja, u zavisnosti od konačne arhitekture.

III. MODEL

Model programske podrške koji se predlaže u ovom radu se sastoji od dva dela: **jezgro programske podrške** (*core*) koje obezbeđuje funkcionalnost, i **korisnička sprega** (*UI*) za prihvat korisničkog unosa i prikaz trenutnog stanja rada.

Jezgro programske podrške se sastoji od sledećih gradivnih blokova: (1) programska sprega (*Application Programming Interface – API*) koja obezbeđuje konkretnu funkcionalnost; (2) protokol na ISO OSI nivou sesije kao usluga za prijem zahteva od strane *UI*; (3) Virtuelni uređaj za komunikaciju sa *UI* (zasnovan na POSIX standardu). Blok dijagram jezgra je prikazan na Slici 1.



Slika 1 – Blok dijagram jezgra programske podrške

Programski kod jezgra treba da zadovoljava sledeće preporuke: (1) prenosiv programski jezik (programski jezik C je dobar izbor za većinu ugrađenih uređaja); (2) izbegavanje upotrebe biblioteka koje su specifične za razvojno okruženje; (3) statičko zauzimanje memorije; (4) izbegavanje operacija koje koriste sistemski stek, kao što su npr. rekurzivni pozivi; (5) izbegavanje konkurentnog programiranja, ili, ukoliko je takav vid programiranja neophodan, korišćenje standardne *POSIX* biblioteke (*pthread*); (6) korišćenje standardnih operacija nad datotekama i sistemskim spregama, ukoliko je moguće (takođe *POSIX* standard). Tako napisan kod biće lakše preneti na različite ciljne platforme.

Programska sprega jezgra treba da realizuje i objavi u obliku *.h* biblioteke, spisak funkcija (*API*) koje obezbeđuju celokupnu funkcionalnost jezgra. Osnovne funkcije su, npr: (1) funkcija za inicijalizaciju (*init*), (2) funkcija za deinicijalizaciju (*exit*), (3) funkcija za kreiranje nove instance, odnosno sesije (*open*), (4) funkcija za zatvaranje instance, odnosno sesije (*close*), (5) funkcije za obavljanje konkretnih operacija nad određenom instancom, odnosno u okviru otvorene sesije (*ioctl*).

Protokol na ISO OSI nivou sesije služi za prijem zahteva i slanje odgovora od strane *UI,* koji zahteva uslugu od jezgra. Protokol se zasniva na sledećim osnovnim primitivama: (1) uspostavljanje i raskidanje veze (na ovaj način je moguće da jezgro opslužuje više *UI* aplikacija u okviru zasebnih sesija); (2) prijem zahteva; (3) poziv odgovarajuće *API* funkcije jezgra, koja odgovara primljenom zahtevu; (4) slanje odgovora. Obrađivač zahteva treba da štedi sistemske resurse, pa se preporučuje korišćenje binarnog protokola. Predlaže se korišćenje protokola *OBEX* [2], s obzirom da omogućava izuzetno jednostavno parsiranje, uz mogućnost proizvoljnog broja podataka različitih tipova u okviru istog paketa. Dobra osobina *OBEX* je i laka nadogradnja, uz zadržavanje kompatibilnosti sa ranijim verzijama.

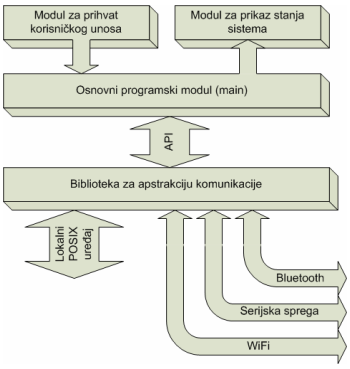
Virtuelni uređaj za komunikaciju sa *UI* omogućava predstavljanje kompletnog jezgra u obliku standarnog *POSIX* uređaja, sa primitivama *open, close, read, write.* Ovaj uređaj može da bude samostalan u okviru programske podrške ugrađenog sistema, ili se može lako integrisati u okviru sistema datoteka OS *Linux* koji se sve češće izvršava na velikom broju ugrađenih procesora. *OBEX* zahtevi koji se žele uputiti jezgru, prosleđuju se u obliku niza bajtova korišćenjem primitive *write.* Nakon toga, poziva se primitiva *read,* koja se blokira i konačno isporučuje niz bajtova koji sadrže *OBEX* odgovor sa rezultatom obrađenog zahteva.

Važna osobina ovako koncipirane programske podrške jezgra je što omogućava laku izmenu korisničke sprege koja se koristi, po potrebi. Takođe, moguće je korišćenje više programskih sprega iz više različitih *UI* aplikacija, zahvaljujući postojanju sesija. Konačno, moguće je izmestiti *UI* aplikaciju na jedan ili više udaljenih uređaja.

Omogućavanje rada sistema sa *UI* aplikacijom na udaljenom uređaju nije komplikovano zahvaljujući modelu jezgra. U zavisnosti od sprege kojom je udaljena aplikacija povezana sa sistemom na kojem se jezgro izvršava (serijska sprega, *WiFi, Bluetooth* i sl), potrebno je otvoriti datu spregu, čitati podatke sa nje i prosleđivati korišćenjem primitive *write* na *POSIX* uređaj jezgra. Na isti način, podatke dobijene pozivom *read* na istom *POSIX* uređaju, treba u istom obliku proslediti na ciljnu spregu. Ovaj način prosleđivanja je jednostavan i može se podržati za više različitih sistemskih sprega istovremeno (Slika 2).

Korisnička sprega (*UI*) se sastoji od (1) biblioteke za apstrakciju komunikacije sa jezgrom sistema; (2) modula

za prihvat korisničkog unosa; (3) modula za prikaz stanja sistema (ukoliko se ono prikazuje), npr. u grafičkom obliku na displeju; (4) osnovnog programskog modula (*main*).



Slika 2 – Blok dijagram korisničke sprege (*UI*)

Biblioteka za apstrakciju komunikacije sa jezgrom sistema obezbeđuje *API* sa funkcijama koje se direktno preslikavaju na moguće zahteve jezgru, odnosno *API* programske sprege jezgra. *UI* aplikacija koristi te *API* pozive da ostvari željenu funkcionalnost, u zavisnosti od zahteva korisnika. Biblioteka sakriva detalje komunikacionog protokola od *UI* aplikacije i čini funkcije jezgra dostupnim, kao da su realizovane u samoj aplikaciji. Programski kod biblioteke treba da bude napisan u skladu sa preporukama koje su navedene za programski kod jezgra, kako bi se ona što lakše prenela na odredišnu platformu.

Modul za prihvat korisničkog unosa sadrži potrebne rukovaoce ulaznim uređajima (npr. tasteri na kućištu uređaja, tastatura, daljinski upravljač i sl). On čeka događaje koji ovi rukovaoci objavljuju, prevodi ih u zahteve razumljive *UI* aplikaciji i prosleđuje ih.

Modul za prikaz stanja sistema sadrži sve potrebne rukovaoce fizičkom arhitekturom i sadrži potrebne biblioteke za prikaz grafičkih elemenata, kako bi se omogućio prikaz trenutnog stanja sistema. Ovaj modul obezbeđuje *API* pozive za izmenu stanja sistema (prikaz okvira, ispis teksta, kontrola led dioda i sl).

Osnovni programski modul održava automat stanja sistema, dok su prelazi stanja vođeni događajima koje objavljuje modul za prihvat korisničkog unosa. U skladu sa prelazima stanja, pozivaju se *API* funkcije za komunikaciju sa jezgrom, čime se postiže željena funkcionalnost. U zavisnosti od povratne vrednosti ovih funkcija, odlučuje se o izmeni prikaza stanja sistema, te se izmena obavi odgovarajućim *API* pozivom modula za prikaz stanja sistema.

Grafička *UI* aplikacija je slabo prenosiva, ali je zato lako izmenljiva, dok nije potrebno iznova verifikovati sistem u celosti, pošto funkcionalnost ne može biti dovedena u pitanje zahvaljujući podeljenom modelu programske podrške.

IV. PRIMER REALIZACIJE

Predloženi model programske podrške primenjen je u okviru realizacije sistema za automatizaciju domaćinstva [3]. Funkcionalnost sistema za automatizaciju obavlja se korišćenjem MIPS procesora u okviru integrisanog kola za primene u digitalnoj televiziji kompanije *Trident Microsystems* [4]. Samo integrisano kolo je sastavni deo standardne TV platforme sa LCD panelom za prikaz slike, i sadrži potrebne blokove za dekodovanje audio/video tokova, grafičku obradu (*Graphic Accelerator - GA*) i sprege na panel (*High Definition Multimedia Interface - HDMI*). Korisnici sistema imaju mogućnost kontrole uređaja i svetla u domaćinstvu, kao i postavljanja određenih scenarija u zavisnosti od prilike. Sprega sa korisnicima ostvarena je na nekoliko načina: (1) na samom TV prijemniku korišćenjem daljinskog upravljača i prikaza na ekranu; (2) konzolnom aplikacijom na laptop računaru koji je povezan sa TV serijskom vezom; (3) mobilnim telefonom koji je povezan sa TV *Bluetooth* vezom preko *Bluegiga WT11* [5] modula (Slika 3).

Slika 3. Uređaji koji čine sistem: jezgro i *UI* aplikacije

MIPS procesor na korišćenoj TV platformi izvršava *Linux OS*. Programska podrška jezgra smeštena je unutar *Linux* kernela, u obliku kernel modula (*.ko*). Jezgro se predstavlja korisničkom prostoru (*user space*) u okviru *Linux OS* u obliku virtuelnog *POSIX* uređaja u sistemu datoteka (*/dev/shdev*).

Realizacija sesionog protokola bazirana je na *OBEX*. Sesioni protokol prima zahteve, poziva odgovarajuću *API* funkciju koja se odnosi na automatizaciju domaćinstva, i isporučuje odgovor onome koji je ispostavio zahtev, u zavisnosti od povratne vrednosti pozvane *API* funkcije. Npr. korisnik želi da dobavi podatke o jednoj lampi u domaćinstvu. On formira bafer za slanje u skladu sa *OBEX* objektnim modelom, kreiranjem zahteva `GET` sa zaglavljima `NAME`, koje označava zahtev, `OBEX_HDR_IDENTIFY_FLAG` u kome se navodi da li se traže podaci o prvoj lampi u bazi, ili o sledećoj ukoliko je zahtev iznova ispostavljen, te `OBEX_HDR_CONNECTION_ID` za navođenje identifikatora veze koji određuje na koju sesiju se zahtev odnosi. Po prijemu, zahtev se parsira, proveri se njegova validnost i poziva se *API* funkcija baze podataka kojom se dobavljaju podaci o lampama domaćinstvu. Tada se šalje *OBEX* odgovor `OK`, uz zaglavlja `OBEX_HDR_TYPE`, gde se naglašava tip elementa kao niz karaktera „sh-lamp“, zatim `OBEX_HDR_OBJECT_ID`, koje sadrži identifikacioni broj lampe, `OBEX_HDR_ASCII_NAME`, koje sadrži simbolički naziv lampe, kao i `OBEX_HDR_CONTENT_COUNT` koje sadrži broj funkcija koje lampa podržava. U slučaju da podaci ne mogu biti isporučeni, biće prosleđen npr. odgovor greške `OBEX_NOT_FOUND`, ukoliko nema više lampi u domaćinstvu.

UI aplikacija za TV platformu koristi razvijenu *ShLib* biblioteku u okviru *Linux* korisničkog prostora, uključenu u obliku deljene biblioteke (*.so*). Biblioteka otvara uređaj */dev/shdev* i šalje *OBEX* zahteve korišćenjem poziva *write* tog uređaja, dok prijem odgovora obavlja korišćenjem poziva *read.* Deo *API* funkcija ove biblioteke prikazan je na Slici 4. Ostatak *UI* aplikacije realizovan je korišćenjem okruženja *DirectFB* [6] i *GTK* [7] za *Linux OS,* čime je omogućen grafički prikaz na ekranu. Sprega sa daljinskim upravljačem, kojim se obavlja korisnički unos, ostvaren je pisanjem dodatnog *DirectFB* rukovaoca.

```
void shAPIInit(char *posix_dev)
void shAPIInitFuncs(int (*send)(...), int (*receive)(...))
int shAPIConnect( int *conn_id )
int shAPIDisconnect( int conn_id )
int shAPIDbEnterHome(const char *name, int obj_id, int *id, int conn_id)
int shAPIDbEnterHomeByName(const char *name, int *id, int conn_id)
int shAPIDbEnterHomeById(int obj_id, int *id, int conn_id)
int shAPIDbEnterRoom(const char *name, int obj_id, int *id, int conn_id)
int shAPIPlayMacroWithName(char *name, int conn_id)
int shAPIPlayMacroWithId(int obj_id, int conn_id)
int shAPIStopMacroWithName(char *name, int conn_id)
int shAPIStopMacroWithId(int obj_id, int conn_id)
int shAPIPauseMacroWithName(char *name, int conn_id)
int shAPIPauseMacroWithId(int obj_id, int conn_id)
int shAPIResumeMacroWithName(char *name, int conn_id)
int shAPIResumeMacroWithId(int obj_id, int conn_id)
int shAPIControlModule(int id, char *name, unsigned char req,
                       unsigned char intensity, unsigned char req_reply,
                       int conn_id)
int shAPIControlModuleByName(char *name, unsigned char req,
                             unsigned char intensity,
                             unsigned char req_reply, int conn_id)
int shAPIControlModuleById(int id, unsigned char req,
                           unsigned char intensity, unsigned char req_reply,
                           int conn_id)
int shAPIPollModule(int id, char *name, unsigned char req,
                    unsigned char intensity, int timeout, int conn_id)
int shAPIPollModuleByName(char *name, unsigned char req,
                          unsigned char intensity, int timeout, int conn_id)
int shAPIPollModuleById(int id, unsigned char req, unsigned char intensity,
                        int timeout, int conn_id)
```

Slika 4 – Deo API funkcija ShLib biblioteke

UI aplikacija za laptop računar koristi prenesenu verziju *ShLib* biblioteke, uz minimalne izmene u njenom kodu. Umesto */dev/shdev* uređaja, ova biblioteka upućuje zahteve na *COM* prolaz laptop računara, sa kojeg prima i odgovore. Na strani TV platforme, u okviru *Linux OS* realizovan je veoma jednostavan pozadinski proces (*daemon*) koji otvara serijski uređaj */dev/ttyS1,* konstantno čita podatke korišćenjem *read* poziva, i bez izmene iste prosleđuje pozivom *write* na */dev/shdev* odmah po prijemu. Na isti način, jedna nit u okviru *daemon* aplikacije koristi poziv *read* da čita podatke sa */dev/shdev,* koje odmah prosleđuje pozivom *write* na */dev/ttyS1*. Korisnička sprega ove *UI* aplikacije se zasniva na unošenju tekstualnih naredbi, nakon kojih se poziva odgovarajuća *ShLib API* funkcija i njen rezultat ispisuje u konzolnom prozoru.

UI aplikacija za mobilni telefon koristi *ShLib* integrisan kao *java* programski kod unutar finalne MIDP aplikacije. Ova aplikacija komunicira sa TV platformom korišćenjem *Bluetooth* veze na RFCOMM nivou. Na strani TV, koristi se *WT11* modul koji je povezan sa TV korišćenjem serijske *UART (Universal Asynchronous Receiver - Transmitter)* sprege. *Daemon* aplikacija na *Linux OS* koristi */dev/wt11* za preusmeravanje podataka.

V. RANIJA ISTRAŽIVANJA

Problematika pisanja prenosive i skalabilne programske podrške predstavlja temu koja je aktuelna duže vreme, i predmet je mnogih istraživanja. Radovi na ovu temu rešavaju probleme abstrahovanja fizičke arhitekture, prenosivosti konkurentnih programa i upotrebe standardnih biblioteka [8], [9], [10]. Udaljene korisničke sprege i aspekti prenosivosti takođe su navedeni u nekoliko radova. Matinlassi tako u svom radu ispituje prenosivost i skalabilnost sistema koji koristi terminale, koristeći QADA metod [11]. Nichols et. al u nekoliko radova predlažu način za generisanje udaljene korisničke sprege koju nazivaju *Personal Universal Controller* [12].

Model izložen u ovom radu okrenut je ka praktičnoj realizaciji sistema i lakom održavanju sa naglaskom na ugrađene sisteme zasnovane na *Linux* OS. Izložene ideje su se potvrdile kao veoma efikasne u praktičnoj realizaciji nekoliko sistema koji su razvijeni na Institutu za računarsku tehniku u Novom Sadu.

VI. ZAKLJUČAK

U radu je izložen model programske podrške koji doprinosi skalabilnosti, prenosivosti i lakom održavanju ugrađenih sistema. Upotreba modela predstavlja posebnu pogodnost na sistemima koji koriste jednu ili više udaljenih korisničkih aplikacija (terminala) i koji su zasnovani na *Linux* OS. Date su konkretne smernice, standard i protokol, što čini model veoma primenljivim u praktičnim sistemima.

LITERATURA

[1] Furr, S, *Using POSIX for Embedded Development,* in proceedings of SDR'04, Arizona, USA

[2] Megowan, P, Suvak, D, Kogan, D, *IrDA Object Exchange Protocol OBEX™*, Infrared Data Association, Version 1.2, 1999

[3] Bjelica, M. Z, *Realizacija sistema za automatizaciju domaćinstva sa grafičkom korisničkom spregom na TV prijemniku,* master rad, bibliotka Fakulteta Tehničkih Nauka, Novi Sad, 2008

[4] *Trident Microsystems,* http://www.tridentmicro.com

[5] *Bluegiga WT11 module,* http://www.bluegiga.com/ /WT11_Class_1_Bluetooth_Module

[6] *DirectFB*, www.directfb.org

[7] *GTK+,* www.gtk.org

[8] Vuletic, M, Pozzi, L, Ienne, P, *Programming Transparency and Portable Hardware Interfacing: Towards General-Purpose Reconfigurable Computing,* in proceedings of the IEEE ASAP'04

[9] Murthi, V, Levine, D, Marquis, J, Shirazi, B, *Creating Portable and Automatically Scalable Parallel Software Using the PARSA Programming Methodology,* in proceedings of IEEE ICA3PP'02

[10] Shinjo, Y, Pu, C, *Achieving Efficiency and Portability in Systems Software: A Case Study of POSIX-Compliant Multithreaded Programs,* IEEE Transactions on Software Engineering, Vol. 31, No. 9, 2005

[11] Matinlassi, M, *Evaluating the Portability and Maintainability of Software Product Family Architecture: Terminal Software Case Study,* in proceedings of IEEE/IFIP WICSA'04

[12] Nichols, J, Myers, B, Higgins, M, Hughes, J, Harris, T. K, Rosenfeld, R, Pignol, M, *Generating Remote Control Interfaces for Complex Appliances,* in proceedings of the 15th annual ACM symposium on User interface software and technology, 2002

ABSTRACT

In this paper we present a software model for the embedded systems, that separates functionality (core) from presentation (user interface). The model provides means to easily swap user interfaces, as well as their remote execution. We propose standards, protocols and guidelines for creating portable and scalable embedded software.

A SOFTWARE MODEL WITH REMOTE USER INTERFACE FOR EMBEDDED SYSTEMS

M. Z. Bjelica, I. Papp, D. Samardžija, Z. Jovanović