



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: Милош Милановић
Број индекса: Е12762

Тема рада: Реализација преноса мултимедијалног садржаја на Андроид базираном дигиталном ТВ пријемнику

Ментор рада: Никола Теслић

Нови Сад, Септембар, 2016.



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:		
Идентификациони број, ИБР:		
Тип документације, ТД:	Монографска документација	
Тип записа, ТЗ:	Текстуални штампани материјал	
Врста рада, ВР:	Завршни (Bachelor) рад	
Аутор, АУ:	Милош Милановић	
Ментор, МН:	Проф. Др Никола Теслић	
Наслов рада, НР:	Реализација преноса мултимедијалног садржаја на Андроид базираном дигиталном ТВ пријемнику	
Језик публикације, ЈП:	Српски / латиница	
Језик извода, ЈИ:	Српски	
Земља публиковања, ЗП:	Република Србија	
Уже географско подручје, УГП:	Војводина	
Година, ГО:	2012	
Издавач, ИЗ:	Ауторски репринт	
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, ФО: (поглавља/страна/цитата/табела/ слика/графика/прилога)	7/37/0/0/11/0/0	
Научна област, НО:	Електротехника и рачунарство	
Научна дисциплина, НД:	Рачунарска техника	
Предметна одредница/Кључне речи, ПО:	Андроид, ДЛНА, УПНП,ДМП, Стб пријемник	
УДК		
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, ВН:		
Извод, ИЗ:	У овом раду приказано је једно решење имплементације ДЛНА протокол стека и реализација ДМП уређаја на Андроид оперативном систему.	
Датум прихватања теме, ДП:		
Датум одбране, ДО:		
Чланови комисије, КО:	Председник: Члан: Члан, ментор:	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO:		
Identification number, INO:		
Document type, DT:	Monographic publication	
Type of record, TR:	Textual printed material	
Contents code, CC:	Bachelor Thesis	
Author, AU:	Miloš Milanović	
Mentor, MN:	PhD Nikola Teslić	
Title, TI:	Solution of media streaming support on Android based STB	
Language of text, LT:	Serbian	
Language of abstract, LA:	Serbian	
Country of publication, CP:	Republic of Serbia	
Locality of publication, LP:	Vojvodina	
Publication year, PY:	2012	
Publisher, PB:	Author's reprint	
Publication place, PP:	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, PD: (chapters/pages/ref./tables/pictures/graphs/appendices)	7/37/0/0/11/0/0	
Scientific field, SF:	Electrical Engineering	
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, S/KW:	Android, DLNA, UPnP, DMP, STB	
UC		
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N:		
Abstract, AB:	This paper presents one solution of implementing DLNA protocol stack and realization of a DMP device on Android operative system.	
Accepted by the Scientific Board on, ASB:		
Defended on, DE:		
Defended Board, DB: President:		
Member:		Menthor's sign
Member, Mentor:		

SADRŽAJ

1.	Uvod.....	1
2.	Teorijske osnove	2
2.1	Linuks.....	2
2.2	Android.....	3
2.2.1	Aplikativni nivo.....	3
2.2.2	Aplikativni međusloj	5
2.2.3	Biblioteke i Android izvršioc	5
2.2.4	Linuks jezgro	5
2.3	Java izvorna sprega	6
2.4	UPnP.....	7
2.5	DLNA.....	8
2.5.1	DMP	9
2.6	Fizička platforma.....	9
2.7	Podatak za prenos.....	9
3.	Koncept rešenja.....	11
3.1	DLNA protokol stek.....	12
3.1.1	UPnP.....	12
3.1.2	FSLib	12
3.2	Java izvorna sprega	13
3.3	Android aplikacija.....	14
4.	Programsko rešenje.....	16
4.1	Priprema okruženja	16
4.1.1	Lanac za razvoj programske podrške	16

4.1.2	Android SDK.....	16
4.1.3	Android NDK	17
4.2	DLNA protokol stek.....	18
4.2.1	UPnP.....	18
4.2.2	FSLib	19
4.2.2.1	Lokalna pretraga.....	20
4.2.2.2	Pretraga u mreži	20
4.3	Java izvorna sprega	21
4.3.1	Deklaracija prototipova funkcija u Javi.....	21
4.3.1.1	Funkcija za prijavljivanje uređaja na mreži	21
4.3.1.2	Funkcija za odjavljivanje uređaja sa mreže	21
4.3.1.3	Funkcija za pristupanje direktorijumu	22
4.3.1.4	Funkcija za prikazivanje trenutnog direktorijuma	22
4.3.1.5	Funkcija za prikazivanje prvog podatka ili direktorijuma po redu	22
4.3.1.6	Funkcija za prikazivanje sledećeg podatka ili direktorijuma po redu.....	22
4.3.1.7	Funkcija za pregled atributa podatka ili direktorijuma	22
4.3.2	Generisanje heder podatka	22
4.3.3	Definicija funkcija u C izvornom kodu.....	23
4.4	Android aplikacija	23
5.	Ispitivanje i verifikacija	26
5.1	Ispitivanje inicijalizacije Android aplikacije.....	26
5.2	Ispitivanje realizacije funkcija FSLib-a	27
5.3	Ispitivanje reprodukcije multimedijalnog sadržaja	27
6.	Zaključak	29
7.	Literatura.....	30

SPISAK SLIKA

Slika 2.1-1 Arhitektura Linuks jezgra.....	3
Slika 2.2-1 Arhitektura Android operativnog sistema	6
Slika 2.4-1 Arhitektura UPnP-a	8
Slika 2.5-1 Prikaz DLNA okruženja	9
Slika 2.7-1 Arhitektura Android opertivnog sistema sa rešenjem DLNA protokol steka .	11
Slika 3.3-1 Koncept rešenja DLNA protocol steka na Android opertivnom sistemu.....	15
Slika 4.1-1 Način povezivanja programskog rešenja izvornog međusloja (Android NDK) i Android aplikacije (Android SDK)	17
Slika 5.1-1 Izvršena uspešna inicijalizacija Android aplikacije i FSLib biblioteke	26
Slika 5.2-1 Rezultat pretrage.....	27
Slika 5.3-1 Reprodukcija audio i video sadržaja.....	28
Slika 5.3-2 Reprodukcija slike	28

SKRAĆENICE

DLNA – *Digital Living Network Aliance*, Protokol za deljenje digitalnog sadržaja između multimedijalnih uređaja

UPnP – *Universal Plug and Play*, Skup mrežnih protokola koji dozvoljavaju multimedijalnim uređajima da ostvare međusobne veze i uspostave razne mrežne usluge

TCP/IP – *Transmission Control Protocol and Internet Protocol*, Skup protokola koji omogućavaju deljenje resursa putem mreže

HTTP – *Hypertext Transfer Protocol*, Mrežni protokol i deo internet protokola namenjen za isporučivanje veb stranica

UDP – *User Datagram Protocol*, Mrežni protokol

XML – *Extensible Markup Language*, Standardni skup pravila za definisanje formata podataka u elektronskoj formi

SOAP – *Simple Object Access Protocol*, Protokola za razmenu strukturiranih infomacija

STB – *Set-top box*, Digitalni TV prijemnik

DMP – *Digital Media Player*, Uredaj za pretragu i reprodukciju multimedijalni sadržaj

API – *Application programming interface*, Programska sprega

NAT – *Network Address Translation*, Preslikavanje mrežne adrese

TV – *Television*, Televizija

USB – *Universal Serial Bus*, Univerzalna serijska magistrala

1. Uvod

Zadatak ovog rada je da predstavi jedno rešenje implementacije DLNA (eng. Digital Living Network Aliance) [3] protokol steka i realizaciju DMP (eng. Digital Media Player) [3] uređaja na Android operativnom sistemu. DMP uređaj je realizovan uz pomoć FSILib [5] biblioteke (koja predstavlja skup funkcija DLNA protokol steka) i skup funkcija UPnP (eng. Universal Plug and Play) [2] protokol steka. Programsко rešenje je implementirano i testirano na Marvel BG2 platformi [4].

Danas najčešći metod praćenja multimedijalnog sadržaja predstavlja gledanje televizora. Iako postoje drugi načini da se multimedijalni sadržaj sa personalnog računara prikaže na TV-u, udaljavanje od TV prijemnika (tj. Odlazak do računara i korišćenje miša i tastature) nije poželjna mogućnost. Zahvaljujući DLNA forumu, definisan je nov i savremen način deljenja , kontrole i reprodukcije multimedijalnog sadržaja u lokalnoj mreži, DLNA protokol stek.

Ovaj rad čine sedam poglavlja:

- U prvom poglavlju predstavljen je kratak opis rešenja
- Drugo poglavlje predstavlje teorijske osnove Android operativnog sistem, DLNA protokola steka, UPnP protokol steka i kratko objašnjenje podatka za prenos
- U trećem poglavlju su opisane ključne komponente ovog rešenja i opis bitnih delova izvornog koda
- Četvrto poglavlje predstavlja detaljan opis realizacije rešenja
- U petom poglavlju su opisani načini ispitivanja i verifikovanja rešenja
- Šesto poglavlje predstavlja karatak rezime svega što je urađeno i mogućnosti za dalji razvoj
- U sedmom poglavlju dat je spisak korišćene literature za izradu rada

2. Teorijske osnove

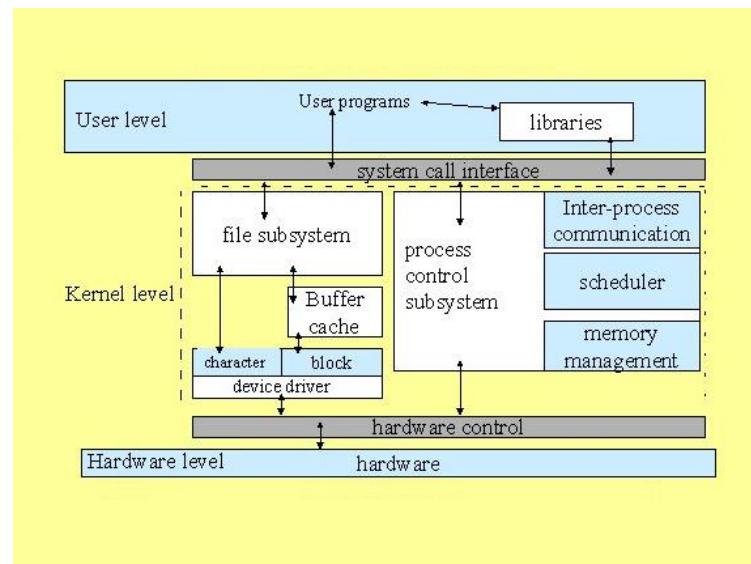
U ovom poglavlju su predstavljene teorijske osnove Android operativnog sistema, DLNA protokola steka, UPnP protokol steka i kratko objašnjenje podatka za prenos

2.1 Linuks

Linuks (eng. Linux) je računarski operativni sistem baziran na juniksu (eng. Unix), pisan je i razvijen u skalu sa principima slobodne programske podrške. Glavni deo linuksa predstavlja njegovo jezgro (takozvani kernel), ili centralni deo operativnog sistema.

Linuks je prvo bitno bio predstavljen kao slobodni operativni sistem za računare zasnovane na 32-bitnoj Intel x86 seriji mikroprocesora. Od tada je prebačen na više računarskih platformi nego i jedan drugi operativni sistem. Linuks je glavni operativni sistem na serverima, međnfrejm računarima i superračunarima. Više od 90% današnjih super računara koristi neku verziju linuksa. Linuks takođe radi na mikrosistemima u kojima je operativni sistem obično ugrađen u uređaj. A to su mobilni telefoni, tablet računari, mrežni usmerivači, televizori, digitalni TV prijemnici i razne konzole za igru. A jedna od bitnijih stvari je i da Android [1] operativni sistem je izgrađen na bazi linuksovog jezgra.

Njegov glavni zadatak je upravljanje i rad sa fizičkom arhitekturom tj. izlazno – ulaznim uređajima, mikroprocesorom i memorijom [6].



Slika 2.1-1 Arhitektura Linuks jezgra

2.2 Android

Android je operativni sistem baziran na linuks jezgru sa međuslojnom programskom podrškom, bibliotekama i API-jem koji su pisani u C programskom jeziku. Aplikativni deo programske podrške naleže na sloj „application framework“ koji uključuje Java kompatibilne biblioteke bazirane na Apache Harmony Java implementaciji. Android koristi Dalvik [1] virtualnu mašinu koja skoro u realnom vremenu izvršava Dalvik dekod kód koji se obično prevodi iz Java bajtkoda.

Glavna platforma korišćena za Android operativne sisteme je bazirana na ARM arhitekturi. Kasnije je napravljena i podrška i za x86 platforme tj. za personalne računare u Android x86 projektu.

Arhitektura Androida može se podeliti u četri nivoa:

- Aplikativni nivo
- Aplikativni međusloj
- Biblioteke i Android izvršioc
- Linuks jezgro

2.2.1 Aplikativni nivo

Aplikativni nivo (eng. Applications) je sloj koji sadrži aplikacije koje korisnik direktno koristi. Razvijane su u Javi sa podrškom biblioteka za Android. Dobra stvar Androida što je razvijan po principima slobodne programske podrške, tako da razvijanje aplikacija je besplatno.

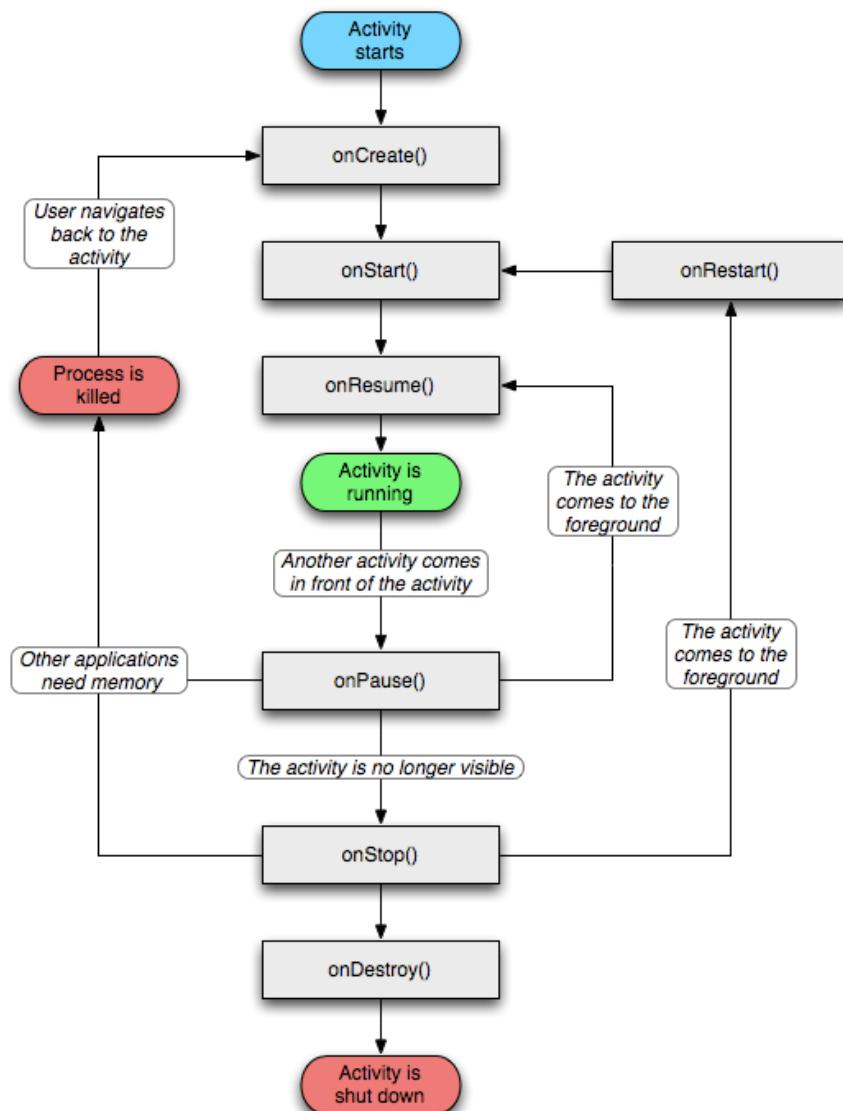
Neke od standardnih aplikacija koje se uvek nalaze na Androidu su: Kontakti, Pretraživač, Telefon, Market, Pretraživač datoteka, Medija plejer, itd.

Android aplikacija može da se sastoji od:

- Aktivnosti (eng. Activity)
- Servisa (eng. Service)

Aktivnost je grafički element koji odgovara jednom ekranu, pretežno je glavni elemenat prilikom razvijanja aplikacije sa grafičkim spregom. Ona ima svoj životni vek (Slika 2.2.1-1).

Servisi su uslužni procesi koji rade u pozadini, pretežno se koriste radi osluškivanja i reakcije na neki događaj koji su izvršeni od strane korisnika ili same logike uređaja.



Slika 2.2.1-1 Životni vek Aktivnosti

2.2.2 Aplikativni međusloj

Aplikativni međusloj (eng. Application Framework) predstavljaju sistemske aplikacije. Na višem su nivou hijerarhije od aplikacija aplikativnog nivoa i rukuju s njima u zavisnosti od događaja. Neki elementi među sloja su:

- Delioc podataka (eng. Content Provider)
- Slušaoc broadcast poruka (eng. Broadcast Receiver)
- Upravljač aktivnostima (eng. Activity Manager)
- Upravljač telefonom (eng. Telephony Manager)
- Upravljač prozorima (eng. Window manager)

Delioc podataka je element koji omogućuje deljenje i razmenu podataka u sistemu.

Slušaoc poruka je element koji prihvata sistemske broadcast poruke. Npr. Prazna baterija, stigao mejl, stigla poruka itd.

Upravljač aktivnostima određuje koja će aktivnost i kada da se izvrši ili zaustavi.

Upravljač telefonom omogućuje korisniku upravljanje dolazećim i odlazećim pozivima.

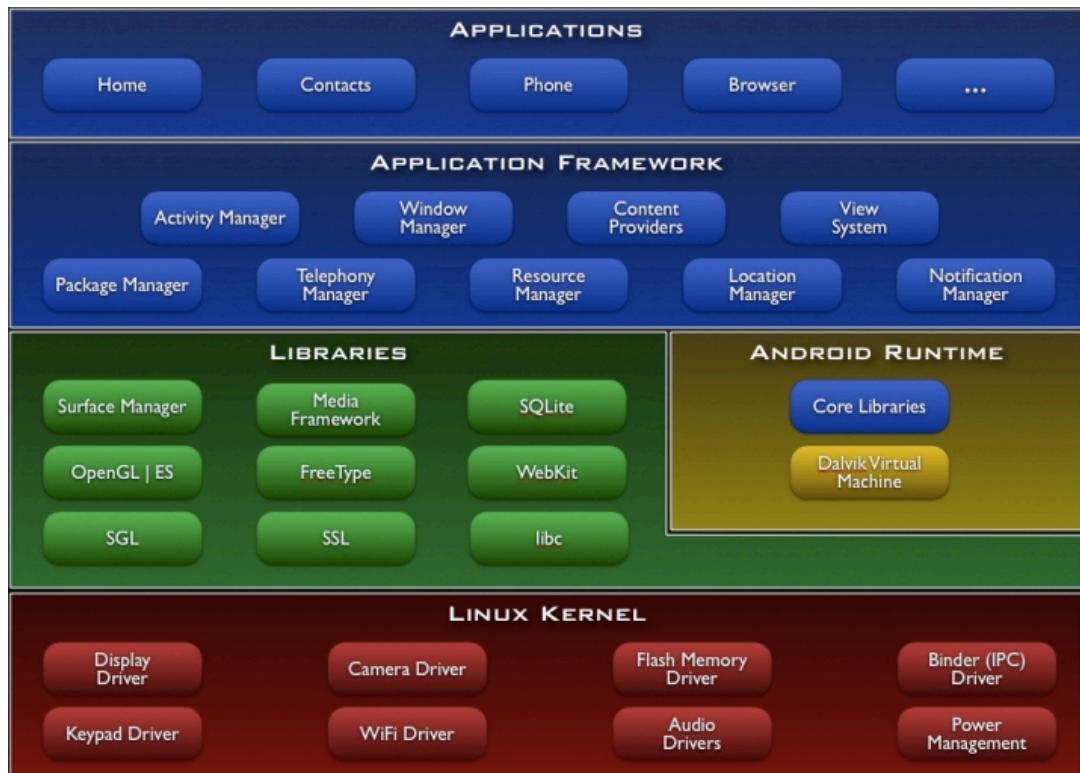
Upravljač prozorima omogućuje kontrolu i upravljanje prozorima u zavisnosti koja se aplikacija (aktivnost) izvršava.

2.2.3 Biblioteke i Android izvršioc

Biblioteke (eng. Libraries) su ustvari most između aplikativnog međusloja i jezgra. Kada Android izvršioc (eng. Android Runtime) koji predstavlja dalvik virtualnu mašinu (eng. Dalvik Virtual Machine) izvršava Java bajtkod tj. Android aplikaciju, on preko biblioteka traži resurse od jezgra.

2.2.4 Linuks jezgro

Linuks jezgro (eng. Linux Kernel) za Android je baziran na standradnom Linuks jezgru ali sa nekim izmenama načinjenim od strane proizvođača. Ne podržava osnovni sistemski programsku podršku i mrežni protokol koji omogućuju osnovni korisnički grafički interfejs (eng. X Window System) niti poseduje celokupan set standardnih GNU biblioteka tako da je veoma teško prenositi postojeće linuks aplikacije na Android.



Slika 2.2-1 Arhitektura Android operativnog sistema

2.3 Java izvorna sprega

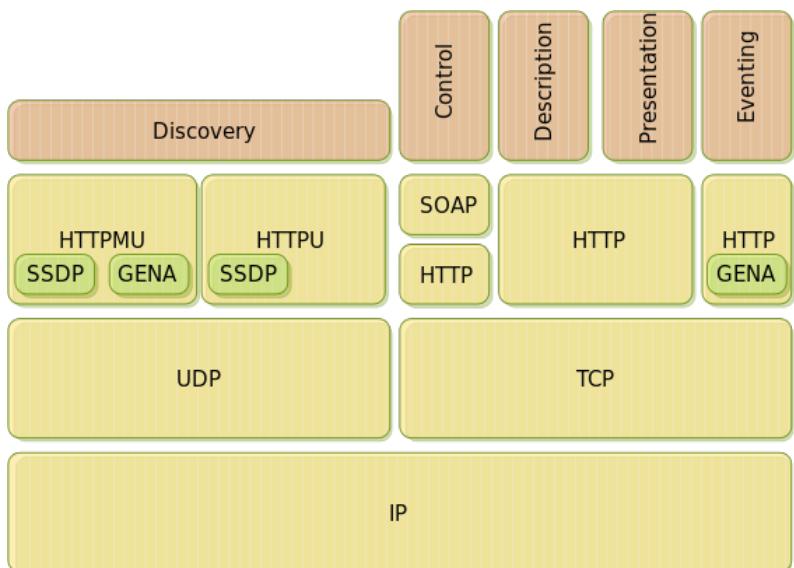
Java izvorna (eng. Native) sprega je programski međusloj koji omogućuje pozivanje (ili da bude pozvana) biblioteka koje su pisane u C i C++ programskom jeziku. Ceo Android operativni sistem se zasniva na ovom konceptu. Java programski jezik je veoma dobar objektno orijentisan jezik, međutim dosta je spor u poređenju sa C programskim jezikom. I zato izvorna sprega ima veliku ulogu u samim performansama izvršnog programa. Jer oni kritični delovi izvršnog programa za koje je potrebna velika brzina, mogu se napisati u C izvornom kodu. Iyovrna sprega pored dobrih stvari ima i par mana:

- Može doći do grešaka koje mogu da destabilizuju virtualnu mašinu
- Gubi se platformska nezavisnost
- Oslobođilac memorije (eng. Garbage Collection) [1] ne radi automatski kao u Javi, već se mora dobro pripaziti na oslobođanje memorije

2.4 UPnP

UPnP standard opisuje grupu protokola za umrežavanje mrežnih uređaja (personalnih računara, štampača, internet kapija, WiFi pristupnih tačaka, mobilnih uređaja). UPnP je namenjen pre svega lokalnim mrežama u stambenim zonama bez stručnih administratora. UPnP omogućava uređajima da jednostavno uočavaju međusobno prisustvo i uspostavljaju funkcionalne mrežne usluge za deljenje podataka, komunikaciju i zabavu. Danas se UPnP protokol najviše koristi za otvaranje NAT (eng. Network Address Translation) priključaka, obezbeđujući igračkim konzolama povezivanje sa internet opslužiocima. Koncept je proizišao iz PnP (eng. Plug and Play) tehnologije koja se koristi za dinamičko priključivanje uređaja za personalni računar, ali se ne odnosi direktno na nadogradnju PnP tehnologije. UPnP uređaji su „plug and play“ pošto po povezivanju na lokalnu mrežu automatski iniciraju i zasnivaju mrežne odnose sa prisutnim kompatibilnim UPnP uređajima.

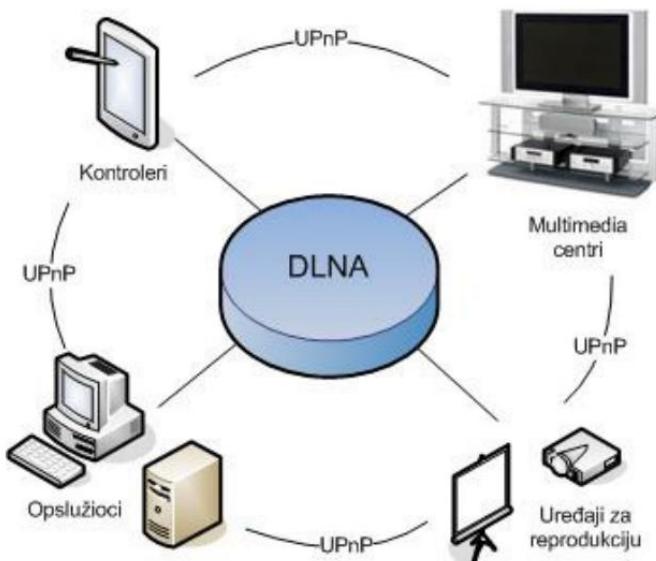
UPnP je otvorena arhitektura mrežnih protokola i omogućava komunikaciju između dva mrežna uređaja pomoću već afirmisanih standarda kao što su TCP/IP (eng. Transmission Control Protocol and Internet Protocol), HTTP(eng. Hypertext Transfer Protocol), XML (eng. Extensible Markup Language) i SOAP (eng. Simple Object Access Protocol). UPnP arhitektura takođe podržava tzv. „zero-configuration networking“ tehnologiju. Uređaj koji poseduje UPnP tehnologiju može dinamički da se prijavi na lokalnu mrežu, preuzme IP adresu, objavi svoje ime, a poseduje mogućnost da na zahtev objavljuje svoje mogućnosti i saznaje podatke o postojanju i mogućnostima takođe prisutnih uređaja. Uređaj takođe može da napusti mrežu bez objavljivanja dodatnih poruka o svom stanju.



Slika 2.4-1 Arhitektura UPnP-a

2.5 DLNA

UPnP A/V standard predstavlja veoma fleksibilan način deljenja multimedijalnog sadržaja u lokalnoj mreži. Prekomerna fleksibilnost koju UPnP A/V standard dozvoljava proizvođačima, pri konfiguraciji svojih uređaja, osnovni je razlog nastanka DLNA [3] interoperabilnih smernica da se pojednostavi i ograniči fleksibilnost UPnP A/V standarda. DLNA protokol stek definiše pet klasa uređaja (Player, Server, Controller, Renderer i Printer) koje su inače podskup UPnP A/V uređaja (Slika 2.4-1). DLNA definiše obavezne podržane A/V formate, vrstu lokalne mreže (Ethernet ili WiFi) i uobičajene uslove korišćenja. DLNA organizacija takođe organizuje proces sertifikacije uređaja, garantujući međusobnu „out-of-the-box“ kompatibilnost uređaja.



Slika 2.5-1 Prikaz DLNA okruženja

2.5.1 DMP

DMP [3] (eng. Digital Media Player) je jedna od klase uređaja. On pretražuje i reproducuje multimedijalni sadržaj koji se nalazi na DMS-u [3] (eng. Digital Media Server). Nije vidljiv u mreži kao DMC [3] (eng. Digital Media Controller). Primeri DMP-a su: televizori, kućni bioskopi, igračke konzole, pametni telefoni, tablet računari itd.

2.6 Fizička platforma

Ciljna fizička platforma predviđena za pokretanje realizovane programske podrške je Set-top boks. Set-top boks predstavlja Marvelova (eng. Marvell [4]) razvojna ploča (Marvel BG2). Ona se sastoji od Marvelovog ARM mikroprocesora od dva jezgra radnog takta od po 1GHz, radne memorije od 512MB, USB magistrale, HDMI priključaka itd. Razvojnu ploču pokreće Android operativni sistem (2.3.x Gingerbread) koji sadrži sve potrebne rukovaće preifernih uređaja za uspešno realizovanje programske podrške.

2.7 Podatak za prenos

Prilikom pisanja programske podrške za namenske računarske sisteme, uzeto je u obzir da platforma i fizička arhitektura mogu biti različiti. Tako da je na neki način programska podrška (eng. Software) pisana za univerzalnu platformu a da se prilikom priprema za upotrebu te programske podrške tačno bira i ona se prenosi na izabranu platformu [8].

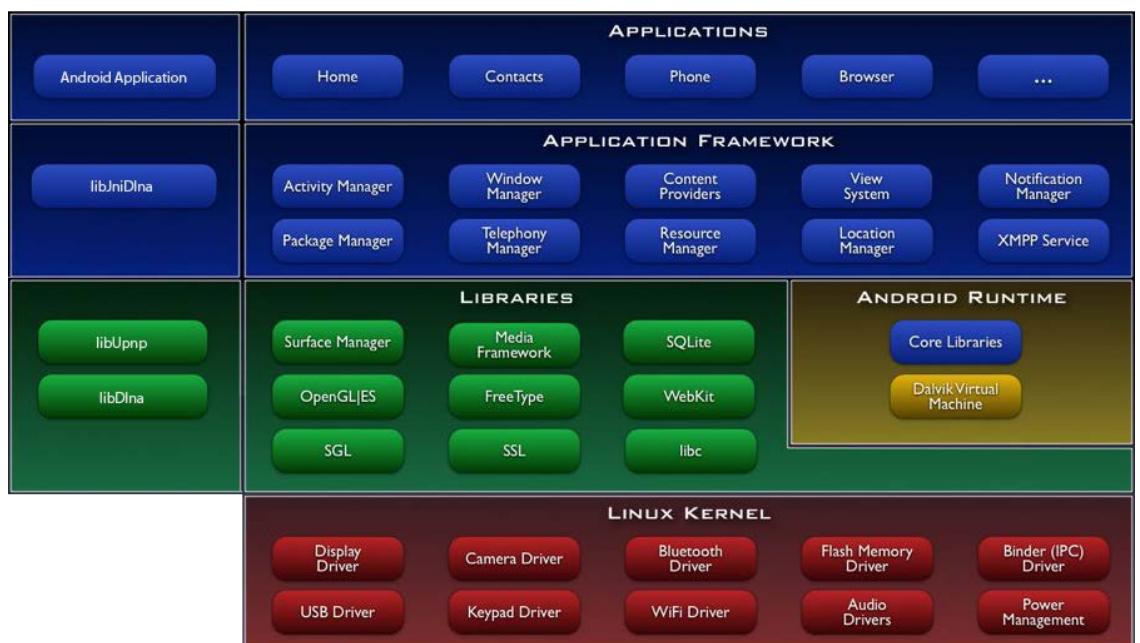
Taj podatak koji sadrži informacije koji će podaci biti uključeni u prenosu koji će kompjajler koristiti itd. zove se „Makefile“. Za Androidove paltorme njegovo ime se razlikuje „Android.mk“ a i sama sintaksa pisanja je za nijansu drugačija.

Standardni linuks programi za kompjajliranje koriste standradni kompjajler „gcc“, ali osnivači Android operativnog sistema odlučili su se za modifikovanu verziju „gcc“ kompjajlera, bajonik (eng. bionic). On je u suštini isti „gcc“ ali postoje neke sitne razlike. Bajonik je nastao zato što je „gcc“ rađen po principu slobodne programske podrške, a njegova licenca ne dozvoljava nikakvo komercijalizovanje, zato je moralo da se nađe prinudno rešenje i napravljen je bajonik.

3. Koncept rešenja

Programska podrška za pretragu i reprodukciju multimedijalnog sadržaja (DMP) realizovana je na osnovu DLNA protokol steka. Zbog pretrage multimedijalnog sadržaja i njegovog korišćenja potreban je UPnP protokol stek koji definiše umrežavanje uređaja u mreži. Tako će DLNA protokol stek koristiti usluge UPnP protokol steka.

Pošto na platformi na kojoj treba da se izvršava programsko rešenje, nalazi se Android operativni sistem koji je baziran na Javi izvornom kodu, a DLNA protokol stek je pisan u C izvornom kodu, potrebno je napisati među sloj takozvana Java izvorna sprega. Uz pomoć Java izvorne sprege Android aplikacija moćiće da koristi funkcije DLNA protokol steka za realizaciju funkcija pretrage i reprodukovanja multimedijalnog sadržaja .



Slika 2.7-1 Arhitektura Android opertivnog sistema sa rešenjem DLNA protokol steka

3.1 DLNA protokol stek

3.1.1 UPnP

UPnP protokol stek je pisan po principima slobodne programske podrške. Tako da se njegov izvorni kod može naći i besplatno preuzeti. Cilj je da se preuzeti izvorni kod prenese na odgovarajuću platformu tj. Android platformu.

On je pisan u izvornom C kodu i namenjen je različitim platformama i arhitekturama mikroprocesora. U njegovim opcijama ne postoji opcija prenosa na Android platformu, ali pošto je on Linuks kompatabilan i ciljna platforma zadovoljava sve preduslove fizičke arhitekture (eng. Hardware), a pošto se zna da se Android bazira na Linuks jezgru, onda se može dodati nova opcija prenosa za Android platformu.

Dovoljno je u korenskim (eng. Root) direktorijuma izvornog koda UPnP protokol steka dodati jedan podatak „Android.mk“, u kome će se opisati svi podaci koji su potrebni za prenošenje. Nakon toga treba se kompajlirati da bi se dobila ciljna deljena biblioteka „libupnp.so“.

3.1.2 FSLib

U FSLib-u [5] je predstavljen DLNA protokol stek. Pisan je u izvornom C kodu. I obuhvata sve funkcije tj. sve celine koje definiše DLNA protokol stek (DMS, DMC, DMP, DMR). Kao i UPnP protokol stek, pisan je za više platformi i treba izvršiti prenos za Android platformu. Celina koja je od značaja je DMP, tako da u podatak za prenos ne treba unositi sve, već samo neophodne delove izvornog koda uz pomoć kojih može se ostvariti tražena funkcionalnost.

DMP definiše reprodukciju i pretragu. Samu reprodukciju će izvršavati ugrađeni deo aplikacije u Android operativnom sistemu, a pretragu i izbor multimedijalnog sadržaja mora se izvršiti uz pomoć funkcija DLNA protokola.

Pretraga može se podeliti na dve celine:

- Lokalna pretraga
- Mrežna pretraga

Za lokalnu pretragu nije potrebna podrška deljene biblioteke „libupnp.so“, zato što se ona ograničava na pretragu samo u lokalnom fajl sistemu.

Dok mrežna pretraga ne može funkcionišati bez deljene biblioteke. Jer DMP nebi znao da se predstavi u mreži i dobije svoju adresu i samim tim „vidi“ druge DMS-ove.

Nakon sređivanja podatka za prenošenje i njegovog kompajliranja, treba da se dobije deljena biblioteka „libdlna.so“

3.2 Java izvorna sprega

FSLib koji predstavlja DLNA protokol stek i samim tim i funkcije, pisan je u C izvornom kodu. Ciljna platforma na kojoj treba da se izvršava aplikacija je Android operativni sistem, na kome se izvrašavaju aplikacije u Java izvornom kodu. Uz pomoć Java izvorne sprege možemo koristiti C-ovske funkcije i koristiti njihove povratne vrednosti u Javi.

Pisanje izvorne sprege se može podeliti u tri celine:

- Deklaracija prototipova funkcija u Javi
- Generisanje heder podatka
- Definicija funkcija u C izvornom kodu

Na osnovu funkcija iz FSLib-a za pretragu i biranje multimedijalnog sadržaja tj. set funkcija koje su bitne da se ostvari željena programska podrška, deklariše se isti set funkcija u Javi ali kao prototipovi.

Na osnovu prototipova iz Java izvornog koda generiše se heder sa istim prototipovima funkcija za C izvorni kod. Nazivi prototip funkcija u hederu nisu klasična već malo drugačija, pored povratne vrednosti sadrže i rezervisanu reč „JNICALL“ a pored naziva i naziv paketa koji je korisnik u Javi odredio.

Nakon generisanja prototipova funkcija u heder podatku, u zasebnom C podatku treba uključiti generisani heder i definisati funkcije. Definicija funkcija se više odnosi na pozivanje odgovarajućih funkcija iz FSLib biblioteke.

Bitno je da funkcije koje su u Javi deklarisane kao prototipovi imaju iste ili slične povratne vrednosti kao funkcije u FSLib-u, jer se samo tako može sa sigurnošću reći da se funkcija uspešno izvršila.

Funkcije koje treba pozivati iz FSLib-a su:

- Funkcija za prijavljivanje uređaja na mreži „registerEntry“
- Funkcija za odjavljivanje uređaja sa mreže „unregisterEntry“
- Funkcija za pristupanje direktorijumu „setCurDir“
- Funkcija za prikazivanje trenutnog direktorijuma „getCurDir“
- Funkcija za prikazivanje prvog podatka ili direktorijuma po redu „getFirst“
- Funkcija za prikazivanje sledećeg podatka ili direktorijuma po redu „getNext“
- Funkcija za pregled atributra podatka ili direktorijuma „getFileProperty“

Nakon definisanja C-ovskih funkcija koje potiču iz Jave, treba definistati podatak za kompajliranje i prenošenje C-ovskog koda, tako da se može koristi iz Jave. Ciljna deljena biblioteka koja treba da se dobije je „libjnidlna.so“.

3.3 Android aplikacija

Nakon generisanja deljene biblioteke „libjnidlna.so“, može se nastaviti dalje sa razvijanjem Android aplikacije. Skoro sve funkcije FSLib-a koje koristi Android aplikacija vraćaju vrednosti i prihvataju kao parametre su tipa string.

Razvijanje aplikacije se takođe može podeliti u nekoliko celina:

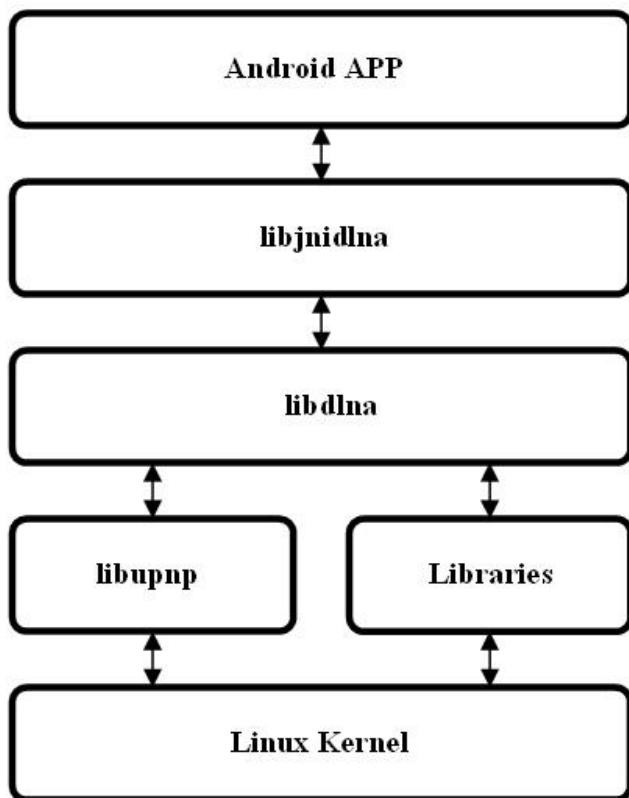
- Definicija funkcije za listanje svih podataka ili direktorijuma
- Definicija Android ugrađene klase ListView [1], radi grafičkog prikazivanja izlistanih podataka ili direktorijuma
- Definicija događaja prilikom interakcije korisnik, aplikacija
- Definicija Android ugrađene klase VideoView [1], radi reprodukovanja multimedijalnih sadržaja, tipa video i audio
- Definicija Android ugrađene klase ImageView [1], radi prikazivanja multimedijalnog sadržaja, tipa slika

Pošto FSLib ne sadrži funkciju koja izlistava sve podatke ili direktorijume u trenutnom direktorijumu, pošto bi to bilo mnogo komplikovanije proslediti nazad Javi, onda se mora definistati funkcija koja to radi. Ona se sastoji od poziva dveju funkcija FSLiba („getFirst“ i „getNext“) i petlje uz pomoć koje prelistava trenutni direktorijum a povratne vrednosti (koje su tipa string) smešta u listu stringova koja ustvari predstavlja sve pronađene direktorijume i podatke.

Nakon generisane liste, ona se može sortirati na osnovu atributa podataka (ili direktorijuma) prikupljenih iz FSLib-a. Atributi se mogu pregledati pozivom funkcije („getFileProperty“) kojoj se prosleđuje string sa nazivom podataka (ili direktorijuma).

Da bi se grafički prikazala generisana lista stringova, mora se iskoristiti već postojeća klasa ListView, kojoj se pri kreiranju objekta, prosleđuje lista stringova kao parametar konstruktora, radi stvaranja grafičkih polja slični tabeli i unos teksta u polja, koja su ustvari nazivi podataka (ili direktorijuma).

Prilikom interakcije korisnika i aplikacije tj. ListView-a, izvršiće se događaj klik. Taj događaj treba da pokrene reprodukciju multimedijalnog sadržaja u zavisnosti od tipa podatka. Ako je tip video ili audio, instanciraće se klasa VideoView, a ako je tip slika onda će se instancirati klasa ImageView.



Slika 3.3-1 Koncept rešenja DLNA protocol steka na Android opertivnom sistemu

4. Programsко rešenje

4.1 Priprema okruženja

Da bi se pisao bilo koji izvorni kod i kasnije dobio izvršni program, bitno je preuzeti od proizvođača, uključiti i podesiti odgovarajuće okruženje za ciljnu platformu.

Za Android operativni sistem potrebno je podesiti tri okruženja. Oni zavise od sloja za koji se piše izvorni kod.

4.1.1 Lanac za razvoj programske podrške

Ovo okruženje se postavlja kada se piše ili prepravlja neka biblioteka u C izvornom kodu. Podešavanja za ovo okruženje dolazi uz izvorni kod Android operativnog sistema i njega pretežno postavlja ili menja osnivač ili vlasnik platforme za koju je operativni sistem namenjen.

Da bi se postavilo okruženje, pretežno se pozicionira u rutu Android operativnog sistema i komandom „source set_mm.sh“ podešavamo konzolu (koja je deo Linuks operativnog sistema za personalne računare) da poziva odgovarajuće biblioteke, odgovarajući kompajler i da kompajlira za odgovarajući mikroprocesor tako da dobijamo željeni izvršni program ili biblioteku.

Nakon pripremanja okruženja dovoljno je pozicionirati se u direktorijumu željenog izvornog koda koji treba da se kompajlira i pozvati komandu „mm“, za ostatak će se pobrinuti unapred objašnjeni događaji, koji će izvršni program smestiti u odgovarajuće izlazne direktorijume.

4.1.2 Android SDK

Android SDK [1] (eng. Software Development Kit), paket programske podrške za razvijanje Android aplikacija.

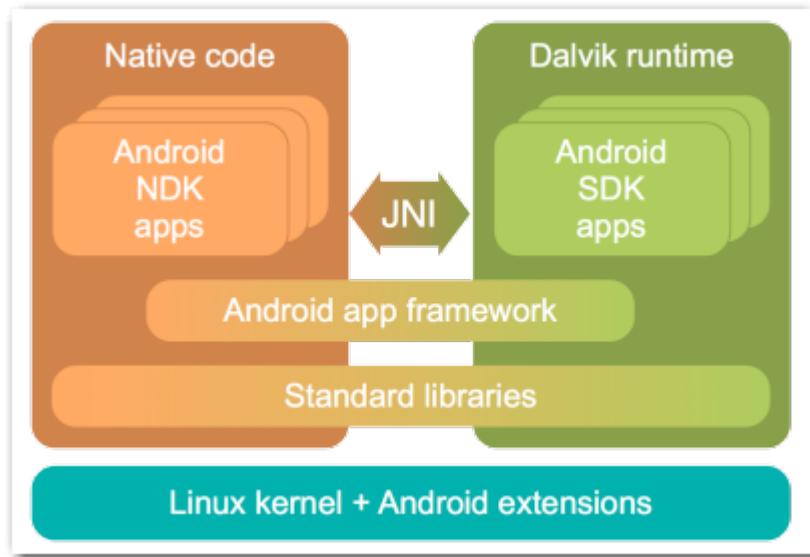
Da bi kodirali, kompajlirali i testirali bilo kakvu vrstu Android aplikacije potrebno je prvo podesiti okruženje. Pošto je Android operativni sistem baziran na principu slobodne programske podrške, osnivači su olakšali korisnicima koji razvijaju aplikacije da u što manje koraka i što prostije podese okruženje.

Sam paket sastoji se od debagera, biblioteka, emulatora, dokumentacije, primera izvornog koda i vodiča. Razvojne platforme koje podržavaju ovaj paket programske podrške su Linuks operativni sistem, MacOs operativni sistem, Windows operativni sistem.

4.1.3 Android NDK

Android NDK [1] (eng. Native Development Kit) paket programske podrške za razvijanje izvornog međusloja Android operativnog sistema.

Paket programske podrške za razvijanje izvornog međusloja je opcionalni sastavni deo paketa programske podrške za razvijanje Android aplikacija. Ne dolazi kao sastavni deo, jer se smatra da nije neophodan korisnicima koji samo žele da razvijaju aplikacije koje su bazirane na Java izvornom kodu. Njegova uloga je velika jer uz pomoć ovog paketa mogu se kompajlirati biblioteke čiji je izvorni kod pisan u C i C++ programskom jeziku.



Slika 4.1-1 Način povezivanja programskog rešenja izvornog međusloja (Android NDK) i Android aplikacije (Android SDK)

4.2 DLNA protokol stek

4.2.1 UPnP

Nakon preuzimanja izvornog koda UPnP protokol steka, potrebno je uvideti značajne delove tj. podatke koji su bitni da UPnP protokol stek funkcioniše bezprekorno i izvrši njihov prenos. Svaki izvorni kod koji je pisan po principu slobodne programske podrške sadrži u sebi neku vrstu dokumentacije. Na osnovu te dokumentacije i podataka za kompajliranje (eng. Makefile) treba se generisati novi podatak za kompajliranje za Android platformu (Android.mk). Sintaksa podatka za kompajliranje za Android platformu je slična onoj koja se dobija uz izvorni kod, ali opet moraju da se naprave par izmena.

Primer podatka za kompajliranje za Android platformu:

`LOCAL_PATH := $(call my-dir)` - Inicijalizacija lokalne promenljive sa vrednošću putanje lokalnog direktorijuma.

`include $(CLEAR_VARS)` - U jednom podatku za kompajliranje mogu se naći lokalne promenljive istih naziva, zato je bitno na početku svim lokalnim promenljivama anulirati vrednost.

`LOCAL_SRC_FILES := \
upnp/src/ssdp/ssdp_device.c` - Dodela naziva i putanja podataka koji su neophodni za rad biblioteke. Ovde naveden samo jedan podatak kao primer.

`LOCAL_C_INCLUDES += $(LOCAL_PATH)/upnp/inc/` - Dodela putanje direktorijuma gde se nalaze heder podaci sa globalnim promenljivama i prototipovima funkcija.

`LOCAL_SHARED_LIBRARIES += libutils libcutils` – Biblioteke koje će se koristiti u radu nove biblioteke tj. nova biblioteka koristiće funkcije koje su definisane u ovim bibliotekama.

`LOCAL_MODULE:= libupnp` – Deklaracija imena nove deljive biblioteke.

`include $(BUILD_SHARED_LIBRARY)` – Ova komanda označava da se treba napraviti deljena biblioteka, u ovom slučaju „libupnp.so“.

Nakon pisanja ovog podatka za kompajliranje, može se izvršiti samo kompajliranje i prenošenje, dovoljno je da se uz pomoć linuks komandne linije (terminala) izvrši pozicioniranje u direktorijum UPnP-a i pozove komandu „mm“.

UPnP biblioteka je izuzetno bitna da bi ostvarili punu funkcionalnost DLNA protokol steka. Bolje reći, bez UPnP biblioteke nemože se ostvariti mrežna komunikacija.

4.2.2 FSLib

FSLib predstavlja DLNA protokol stek koji nije pisan po principu slobodne programske podrške i može se samo koristiti u okruženju koji izriričito definisan licencom.

FSLib je biblioteka koja je namenjena za više platformi, na tom principu je i pisana. Isto kao i UPnP mora da se izvrši prenos i kompajlirati za Android platformu. Na osnovu njenog podataka za kompajliranje, generisće se podatak za kompajliranje za Android platformu. Procedura je ista kao i prilikom kreiranja „libupnp.so“ deljene biblioteke.

Funkcije koje su bitne da se ostvari pretraga, definisane su u okviru jednog tipa „FILE_LIB_ENTRY_t“ i pisane su kao pokazivačke funkcije. Razlog tome je što isti taj set prototipova funkcija može se koristiti za realizaciju drugih klasa uređaja DLNA protokol steka, menjanjem nekih od parametara iste funkcije mogu se koristit u više različitih svrha. Prototipovi funkcija za ovu celinu se nalaze u „filelib.h“ hederu.

Članovi tipa „FILE_LIB_ENTRY_t“ su :

- Promenljiva „name“, na čijoj će se vrednosti održivati definicija funkcija
- Funkcija za inicijalizaciju uređaja na mreži „init“
- Funkcija za deinicijalizaciju uređaja sa mreže „deinit“
- Funkcija za pristupanje direktorijumu „setCurDir“
- Funkcija za prikazivanje trenutnog direktorijuma „getCurDir“
- Funkcija za prikazivanje prvog podatka ili direktorijuma po redu „getFirst“
- Funkcija za prikazivanje sledećeg podatka ili direktorijuma po redu „getNext“
- Funkcija za pregled atributra podatka ili direktorijuma „getFileProperty“

```
typedef struct _file_lib_entry
{
    char* name;
    char* (*setCurDir)(char *URI);
    char* (*getCurDir)(char *URI);
    char* (*getFirst)(char *URI);
    char* (*getNext)(char *URI);
    char* (*getFileProperty)(char *URI, char *ID, char
*retval);
    int (*init)(void);
    int (*deinit)(void);
} FILE_LIB_ENTRY_t;
```

Za realizaciju pretrage na DMP uređaju potrebno je izdvojiti dve celine definicija funkcija. Prva je lokalna pretraga, a druga je pretraga u mreži.

4.2.2.1 Lokalna pretraga

Za realizaciju lokalne pretrage potrebno je definisati promenljivu tipa „FILE_LIB_ENTRY_t“ i dodeliti joj unapred definisane funkcije za lokalnu pretragu koje su pisane maksimalno oslanjajući se na biblioteke linuks jezgra za rad sa direktorijumima i podacima. Promenljiva „name“ sadržaće vrednost „local“ i to će biti glavna naznaka kada se budu koristile funkcije da se radi o lokalnoj pretrazi. Prototipovi funkcija za ovu celinu se nalaze u „file-local.h“ hederu.

Dodata vrednosti promenlivoj tipa „FILE_LIB_ENTRY_t“:

```
static FILE_LIB_ENTRY_t file_local=
{
    "local",
    _setCurDir,
    _getCurDir,
    _getFirst,
    _getNext,
    _getFileProperty,
    _init,
    _deinit
};
```

4.2.2.2 Pretraga u mreži

Za realizaciju pretrage u mreži takođe je potrebno definisati promenljivu tipa „FILE_LIB_ENTRY_t“ i dodeliti joj unapred definisane funkcije, ali za razliku od lokalne pretrage ove funkcije će se oslanjati funkcije deljene biblioteke „libupnp.so“ tj. na UPnP protokol stek. Promenljiva „name“ sadržaće vrednost „dlna“ i to će biti glavna naznaka kada se budu koristile funkcije da se radi o mrežnoj pretrazi. Prototipovi funkcija za ovu celinu se nalaze u „fslib_adapt.h“ hederu.

Dodata vrednosti promenlivoj tipa „FILE_LIB_ENTRY_t“:

```
static FILE_LIB_ENTRY_t file_dlna ={
    "dlna",
    set_cur_dir,
    get_cur_dir,
    get_first,
    get_next,
    get_file_property,
```

```
dlna_init,
dlna_deinit};
```

4.3 Java izvorna sprega

Pisanje izvorne sprege se može podeliti u tri celine:

- Deklaracija prototipova funkcija u Javi
- Generisanje heder podatka
- Definicija funkcija u C izvornom kodu

4.3.1 Deklaracija prototipova funkcija u Javi

Prvi korak pisanja Java izvorne sprege jeste deklarisanje prototipova funkcija u Javi izvornom kodu. Na osnovu tih funkcija kasnije će se izgenerisati heder podatak. Funkcije za izvornu spregu se razlikuju od ostalih po tome što u svom nazivu sadrže rezervisanu reč „native“. Pozivaju se kao sve ostale funkcije u zavisnosti kakva je njihova deklaracija, da li u okviru klase ili globalna.

Funkcije koje treba pozivati iz FSLib-a i za njih napisati prototipove su:

- Funkcija za prijavljivanje uređaja na mreži „registerEntry“
- Funkcija za odjavljivanje uređaja sa mreže „unregisterEntry“
- Funkcija za pristupanje direktorijumu „setCurDir“
- Funkcija za prikazivanje trenutnog direktorijuma „getCurDir“
- Funkcija za prikazivanje prvog podatka ili direktorijuma po redu „getFirst“
- Funkcija za prikazivanje sledećeg podatka ili direktorijuma po redu „getNext“
- Funkcija za pregled atributra podatka ili direktorijuma „getFileProperty“

4.3.1.1 Funkcija za prijavljivanje uređaja na mreži

Funkcija za prijavljivanje uređaja na mreži („registerEntry“) je ustvari funkcija uz pomoć koje se postavlja vrednost promenljive „name“ i inicijalizuje celina koja će se koristiti (lokalna ili mrežna pretraga).

Prototip ove funkcije u Java izvornom kodu:

```
public native int registerEntry();
```

4.3.1.2 Funkcija za odjavljivanje uređaja sa mreže

Na završetku aplikacije potrebno je deinicijalizovati FSLib pozivanje funkcije „unregisterEntry“, samim tim izvršiće se oslobođanje memorije, ulazno-izlaznih uređaja koji su bili korišćeni od strane UPnP-a itd.

Prototip ove funkcije u Java izvornom kodu:

```
public native int unregisterEntry();
```

4.3.1.3 Funkcija za pristupanje direktorijumu

Funkcija za pristupanje direktorijumu „setCurDir“ služi za pozicioniranje u željeni direktorijum na osnovu izabrane putanje koja se prosleđuje parametrom.

Prototip ove funkcije u Java izvornom kodu:

```
public native String setCurDir(String newCurrentDir);
```

4.3.1.4 Funkcija za prikazivanje trenutnog direktorijuma

Funkcija „getCurDir“ vraća putanju trenutnog direktorijuma koji se koristi.

Prototip ove funkcije u Java izvornom kodu:

```
public native String getCurDir();
```

4.3.1.5 Funkcija za prikazivanje prvog podatka ili direktorijuma po redu

Funkcija „getFirst“ vraća naziv prvog podatka ili direktorijuma sa putanje koji je prosleđen kao parametar.

Prototip ove funkcije u Java izvornom kodu:

```
public native String getFirst(String URI);
```

4.3.1.6 Funkcija za prikazivanje sledećek podatka ili direktorijuma po redu

Funkcija „getNext“ vraća naziv sledećek podatka ili direktorijuma sa putanje koji je prosleđen kao parametar.

Prototip ove funkcije u Java izvornom kodu:

```
public native String getNext(String URI);
```

4.3.1.7 Funkcija za pregled atributa podatka ili direktorijuma

Funkcija „getFileProperty“ vraća informaciju o podatku ili direktorijumu. Informacije se odnose na njihve putanje, tipove, vrste itd.

Prototip ove funkcije u Java izvornom kodu:

```
public native String getFileProperty(String URI, int ID);
```

4.3.2 Generisanje heder podatka

Nakon deklaracije prototipova funkcija u Java izvornom kodu, treba generisati heder podatak sa istim tim prototip funkcijama ali za C izvorni kod. Aplikacija „javah“ koja dolazi uz standardni paket Android SDK ima taj zadatak. Pokretanje same aplikacije nije dovoljno već se mora prvo pozicionirati u radni direktorijum Android aplikacije tj. direktorijumu „bin“ pa u

„classes“. Parametar aplikacije mora ime paket i ime same klase gde su prototipovi funkcija deklarisani. I nakon toga može se pokrenuti aplikacija.

Primer pokretanja „javah“ aplikacije:

```
. / javah com.dlna.DlnaNative
```

Sam heder podatak nakon generisanja mora da sadrži sve zadate prototipove funkcija i uključuje heder „jni.h“ koja u sebi sadrži definiciju svih potrebnih funkcija za rad sa izvornom spregom.

4.3.3 Definicija funkcija u C izvornom kodu

Pošto su prototipovi deklarisani sada ih treba definisati i implementirati. Prvo treba napraviti novi podatak, tipa za C izvorni kod. Onda treba uključiti generisani heder iz Jave i ostale hedere koji u sebi sadrže definicije ili prototipove funkcija koje treba koristiti. Znači pored „com_dlna_DlnaNative.h“ treba uneti i „filelib.h“, „file-local.h“, „fslib_adapt.h“. U telima definisanih funkcija iz Jave treba navesti odgovarajuće funkcije iz FSLib-a.

Primer funkcije „registerEntry“:

```
JNIEXPORT jint JNICALL Java_com_dlna_DlnaNative_registerEntry
(JNIEnv *jEnv, jobject jThis, jint fileHandlerType){
    jint returnValue = 0;
    if(0 == fileHandlerType){
        returnValue = registerEntry(getFileLocal(NULL));
    }else if(1 == fileHandlerType){
        returnValue = registerEntry(getFileDlna(NULL));
    }
    return returnValue;
}
```

Nakon definicija svih funkcija potrebno je ovaj C izvorni kod kompajlirati i izvršiti prenos (eng. Porting) za Android platformu. Kao rezultat svega je biblioteka „libjnidlna.so“

4.4 Android aplikacija

Kreiranjem svih potrebnih biblioteka, stečeni su uslovi za kreiranje Android aplikacije. Pošto su prototipovi funkcija ranije deklarisani u okviru klase „DlnaNative“ potrebno je još i uključiti njihove definicije koje se nalaze u deljenoj biblioteci „libjnidlna.so“.

Primer ukjučivanje biblioteke u Java programskom jeziku:

```
static {
    System.loadLibrary("jnidlina");
}
```

Reč „lib“ nije potrebno navoditi jer se podrazumeva. Nakon uključivanja biblioteke funkcije se mogu koristiti.

Pošto FSLib ne sadrži gotovu funkciju za listanje podataka i direktorijuma, onda ona mora de se realizuje u Android aplikaciji. Relizovana je uz pomoć FSLib funkcija „getFirst“ i „getNext“ koje vraćaju vrednosti tipa string. Sve te vrednosti se sakupljaju tj. redaju u listi radi kasnijeg prikazivanja.

Jedan primer realizacije listanja direktorijuma i podataka:

```
public ArrayList<String> listCurrentDirectory() {
    ArrayList<String> returnValue = new
    ArrayList<String>();
    String nextInDir = null;
    String firstInDir = getFirstDlna(" ");
    if (firstInDir != null) {
        returnValue.add(firstInDir);
        do {
            nextInDir = getNextDlna(getCurDirDlna());
            if (nextInDir != null) {
                returnValue.add(nextInDir);
            }
        } while (nextInDir != null);
    }
    if (returnValue.size() == 0) {
        returnValue.add("...");
    }
    return returnValue;
}
```

Bitna stvar pored svega je grafički prikaz nove liste stringova. To se može ostvariti uključivanjem ugrađene klase „ListView“ i prilikom kreiranja njenog objekta, kroz konstruktor se prosleđuje lista stringova koja treba da se prikaže.

Prilikom interakcije korisnika aplikacije i same aplikacije mora da se definišu i događaji na koje će aplikacija izvršavati određene komande tj. reprodukciju izabranog multimedijalnog sadržaja od strane korisnika.

Za prikaz multimedijalnog sadržaja u zavisnosti od tipa, koristiće se dve ugrađene klase. Klasa „VideoView“ za prikaz audio i video multimedijalnog sadržaja i klasa „ImageView“ za prikaz slika.

Definicijom svih neophodnih delova Android aplikacije stekli su se uslovi za definiciju poslednje glavne funkcije. Glavna funkcija mora da sadrži pored svih deklaracija, inicijalizacija polja i kreiranja objekata pravi redosled pozivanja FSLib funkcija.

Da bi se uopšte FSLib mogao koristiti, na samom početku se mora izvršiti inicijalizacija određenih delova samo FSLiba i postavljanje vrste pretrage („local“ ili „dlna“), a na završetku radnog ciklusa aplikacije je obavezna deinicijalizacija.

Primer inicijalizacije:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    .....
    medControl = new DlnaMethods();
    if (medControl.initNative() == 0) {
        Log.i("JAVA", "Initialization FAILED!");
    }
    if (medControl.registerEntry(FILE_LOCAL) == 0) {
        Log.i("JAVA", "Register Entry FAILED!");
    }
    if (medControl.registerEntry(FILE_DLNA) == 0) {
        Log.i("JAVA", "Register Entry FAILED!");
    }
    .....
}
```

Primer deinicijalizacije:

```
@Override
public void onPause(){
    .....
    if(medControl != null)
        medControl.unregisterEntry(FILE_LOCAL);
    if(medControl != null)
        medControl.unregisterEntry(FILE_DLNA);
    .....
}
```

Nakon toga može se pristupati željenim direktorijumima. Za pristup lokalnim podacima i direktorijumima funkcija „setCurDir“ dobiće vrednost parametra „local://“ a za pristup mrežnim „dlna://“. U nastavku, funkcije se mogu bez ikakvih problema koristiti.

Ostatak je na korisniku da kreira redosled događaja tj. vrši željenu interakciju sa aplikacijom.

5. Ispitivanje i verifikacija

Android aplikacija je ispitivana na ciljnoj fizičkoj platformi Marvel BG2. Tip ispitivanja je funkcionaln. Narednim testovima je potvrđena uspešnost rada Android aplikacije a samim tim i izvorne sprege, FSLib biblioteke i UPnP biblioteke. Moduli koji su ispitivani:

- Android aplikacija (Java izvorni kod)
- Android izvorni medusloj (C izvorni kod)
- FSLib biblioteka (C izvorni kod)
- UPnP biblioteka (C izvorni kod)

5.1 Ispitivanje inicijalizacije Android aplikacije

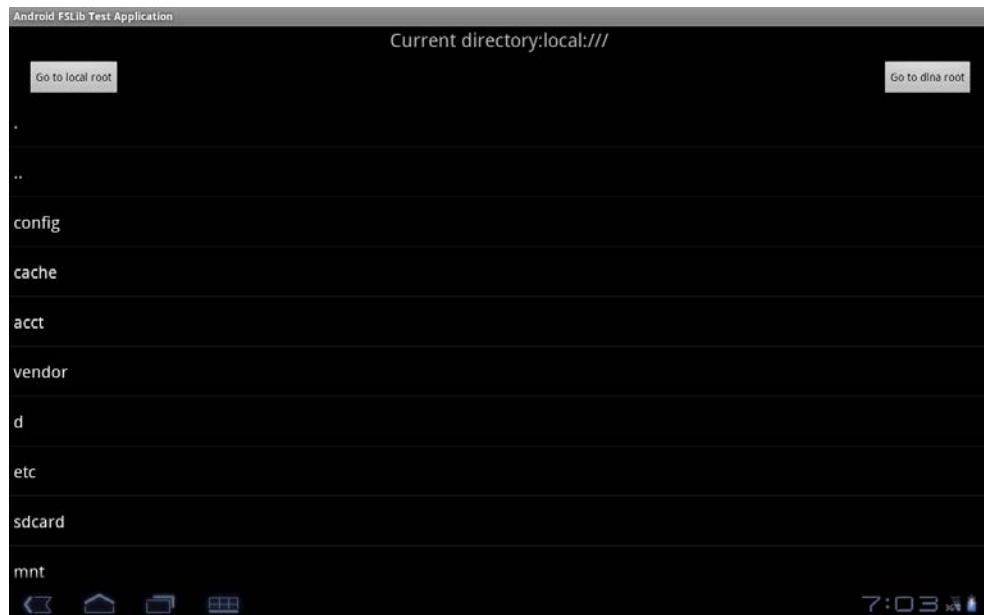
Nakon pokretanja Android aplikacije i inicijalizacije ugrađenih Android klasa sledi uključenje FSLib biblioteke i inicijalizacija modula za rad sa lokalnom i mrežnom pretragom. Ako svaki modul bude uspešno inicijalizovan pojaviće se poruka sa tekstom da je sve uspešno izvršeno (Slika 5.1-1).



Slika 5.1-1 Izvršena uspešna inicijalizacija Android aplikacije i FSLib biblioteke

5.2 Ispitivanje realizacije funkcija FSLib-a

Ugrađenom Android klasom „ListView“ će se grafički prikazati rezultati pretrage FSLib funkcija (Slika 5.2-1 Rezultat pretrage). Ako je pretraga uspešna, pojaviće se lista sa pronađenim direktorijumima i podacima. Ovim se ne potvrđuje samo uspešan rad FSLib funkcija već i uspešan rad Java izvornog sloja preko koga Java poziva FSLib funkcije.



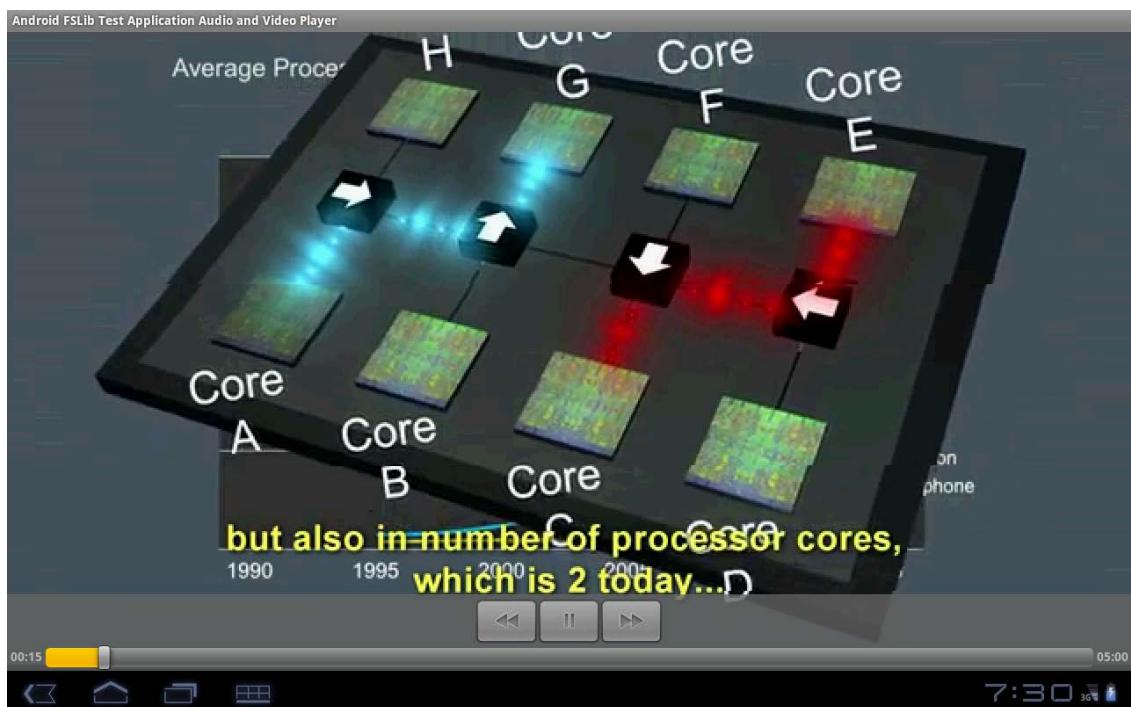
Slika 5.2-1 Rezultat pretrage

5.3 Ispitivanje reprodukcije multimedijalnog sadržaja

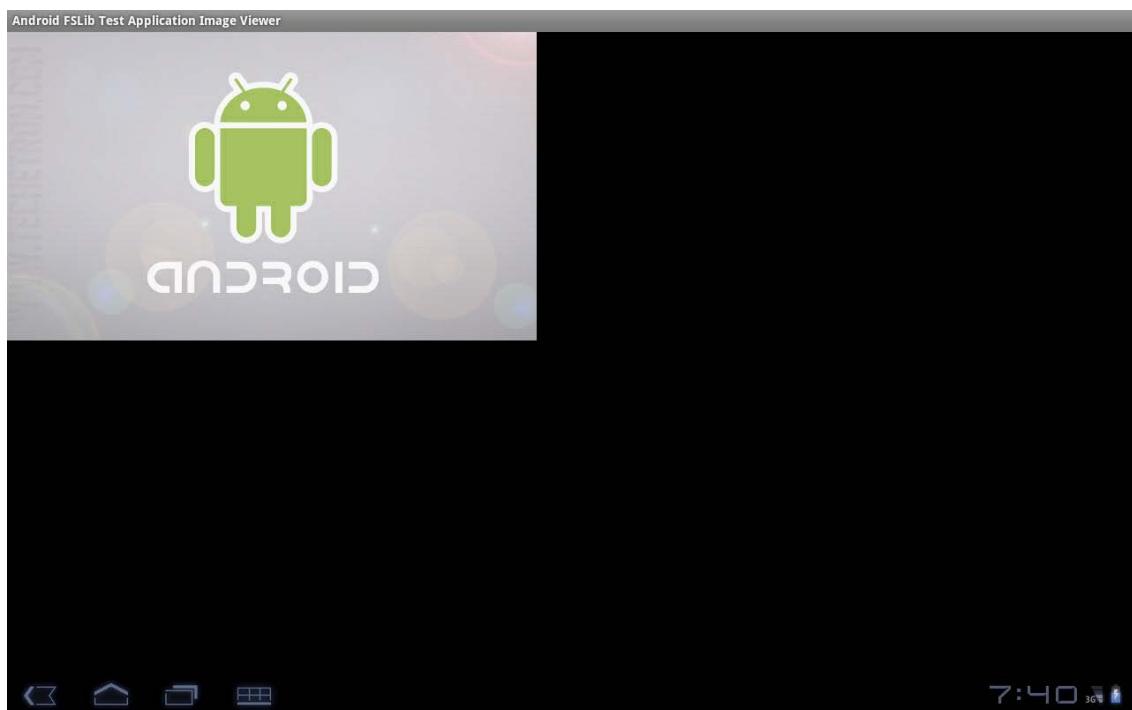
U zavisnosti od odabira multimedijalnog sadržaja tj. tipa podatka (audio, video, slika) pokrenuće se reprodukcija.

Ako je tip audio ili video instanciraće se ugrađena klasa „VideoView“ sa podrškom klase „MediaControl“ i počeće reprodukcija biranog sadržaja (Slika 5.3-1 Reprodukcija audio i video sadržaja).

A ako je tip slika, onda će se instancirati ugrađena klasa „ImageView“ i biće reprodukovani izabrani sadržaj (Slika 5.3-2 Reprodukcija slike).



Slika 5.3-1 Reprodukcija audio i video sadržaja



Slika 5.3-2 Reprodukcija slike

6. Zaključak

U radu je prikazana realizacija DMP uređaja tj. pretraga i reprodukcija multimedijalnog sadržaja sa mreže i lokalnih sistem datodteka na Marvel BG2 platformi. Realizacija je podeljena u tri nivoa, a to je realizacija Android aplikacije kodirane u Java programskom jeziku, realizacija DLNA protokol steka kodiranog u C programskom jeziku i realizacija Java izvorne sprege koja povezuje Android aplikaciju i DLNA, kodirane u C programskom jeziku. Ispitivanje je uspešno izvršeno na ciljnoj platformi.

Uspeh ove realizacije na ciljnoj platformi pokazuje da Android kao operativni sistem pored standardne primene u mobilnim telefonima i tablet uređajima ima veliku mogućnost da postane standardni deo programske podrške prijemnika za digitalnu televiziju [7]. Njegova fleksibilnost je ključna, lako se može nadograditi i izmeniti, a naravno pored fleksibilnosti tu je i stabilnost i preformanse koje dodatno povećavaju negovu primjenjenost i implementaciju. Uz pomoć Androida kao dela programske podrške prijemnika za digitalnu televiziju ne bi dobili samo TV funkcionalnost ili DLNA, već mnogo širi opseg mogućnosti. Pretraga interneta, instaliranje raznih uslužnih aplikacija, igrica, pregled elektronske pošte itd. Zato će u budućnosti primena Android operativnog sistema biti mnogo veća.

7. Literatura

- [1] Android sajt za razvoj, <http://developer.android.com>
- [2] UPnP forum, <http://www.upnp.org>
- [3] DLNA forum, <http://www.dlna.org>
- [4] Marvell, *Hardware Specification*
- [5] iWedia, *FSLib Software API User Manual*
- [6] Predavanja iz predmeta Logičko projektovanje računarskih sistema 2, <http://www.rtrk.uns.ac.rs/studijski-program-2009/v-2009/lprs2>
- [7] M.Vidakovic, N.Teslic, T.Maruna, and V.Mihic: *Android4TV: a proposition for integration of DTV in Android devices*, IEEE 30th International Conference on Consumer Electronics (ICCE), Las Vegas, January 2012, pp. 441-442
- [8] Vladimir Kovačević, Miroslav Popović: *Sistemska programska podrška u realnom vremenu*, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2002