



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: Вељко Илкић

Број индекса: Е12694

Тема рада: Реализација подсистема за репродукцију аудио и видео садржаја за Андроид базиране пријемнике дигиталног телевизијског сигнала

Ментор рада: Никола Теслић

Нови Сад, јун, 2012.



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:	Вељко Илкић	
Ментор, МН:	Проф. др Никола Теслић	
Наслов рада, НР:	Реализација подсистема за репродукцију аудио и видео садржаја за Андроид базиране пријемнике дигиталног телевизијског сигнала	
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 репримт	
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/45/0/2/22/0/0	
Научна област, НО:	Електротехника и рачунарство	
Научна дисциплина, НД:	Рачунарска техника	
Предметна одредница/Кључне речи, ПО:	Репродукција видео и аудио садржаја на Андроид пријемнику дигиталног телевизијског пријемника	
УДК		
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, ВН:		
Извод, ИЗ:	Приказано је једно решење подсистема и програмске подршке за репродукцију аудио и видео садржаја за Андроид базиране пријемнике дигиталног телевизијског садржаја. Описана је уgraђена програмска подршка Андроид оперативног система и измене које су неопходне како би се перформансе подсистема за репродукцију побољшале и прилагоделе циљној платформи.	
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:	Veljko Ilkić	
Mentor, MN:	Prof. dr Nikola Teslić	
Title, TI:	Realisation of subsystem for multimedia playback for Android STB devices	
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/applications)	7/45/0/2/22/0/0	
Scientific field, SF:	Electrical Engineering	
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, S/KW:	Video and audio playback on Android STB device	
UC		
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N:		
Abstract, AB:	This paper showed one solution of subsystem for multimedia playback for Android STB devices. There is description of embedded multimedia software support in Android operating system, and improvements that are needed to be done for better performance of Android STB device.	
Accepted by the Scientific Board on, ASB:		
Defended on, DE:		
Defended Board, DB:	President: Member: Member, Mentor:	Mentor's sign

SADRŽAJ

1.	Uvod.....	1
1.1	Postavka problema	1
1.2	Analiza problema	1
2.	Teorijske osnove.....	4
2.1	Aplikativni sloj.....	4
2.1.1	<i>View.java</i>	4
2.1.2	<i>SurfaceView.java</i>	5
2.1.3	<i>VideoView.java</i>	5
2.2	Aplikativno radno okruženje i <i>JNI</i> sloj.....	5
2.2.1	<i>Media Player</i> radno okruženje	5
2.2.2	<i>MediaPlayer.java</i>	7
2.2.3	<i>android_mediaplayer.cpp</i>	8
2.2.4	<i>mediaplayer.cpp</i>	9
2.2.5	<i>IMediaPlayer.cpp</i>	9
2.2.6	<i>MediaPlayerService.cpp</i>	9
2.3	Sloj niskog nivoa.....	10
2.3.1	<i>main_mediaserver.cpp</i>	10
2.3.2	<i>LayerBuffer.cpp (SurfaceFlinger)</i>	16
2.3.3	<i>AudioFlinger.cpp</i>	17
3.	Analiza rešenja	20
3.1	Opis ciljne platforme	20
3.2	Koncept rešenja.....	21
3.3	Aplikativni sloj.....	22

3.3.1	Detalji realizacije	23
3.4	Aplikativno radno okruženje <i>JNI</i> sloj.....	26
3.5	Sloj niskog nivoa.....	29
4.	Ispitivanje i verifikacija	32
5.	Zaključak.....	35
6.	Literatura	37

SPISAK SLIKA

Slika 2.1 <i>Media Player</i> radno okruženje.....	6
Slika 2.2 Komunikacija modula programske podrške	6
Slika 2.3 Dijagram stanja <i>MediaPlayer.java</i> modula	7
Slika 2.4 Arhitektura programske podrške za reprodukciju multimedije	10
Slika 2.5 Organizacija <i>MediaServer</i> procesa	10
Slika 2.6 Arhitektura <i>OpenCORE</i> modula	11
Slika 2.7 Tok podataka unutar <i>OpenCORE</i> modula	13
Slika 2.8 Arhitektura programske podrške sa <i>StageFright</i> modulom	13
Slika 2.9 Arhitektura <i>StageFright</i> modula	14
Slika 2.10 Komunikacija <i>JAVA</i> aplikacije sa modulima nižeg nivoa.....	15
Slika 2.11 Komunikacija <i>JAVA</i> aplikacija i <i>Surface Flinger</i> modula	16
Slika 2.12 Komunikacija <i>JAVA</i> aplikacije sa <i>FrameBuffer</i> modulom	17
Slika 2.13 Arhitektura programske podrške za reprodukciju zvuka	17
Slika 2.14 Arhitektura programske podrške za reprodukciju zvučnog sadržaja	18
Slika 3.1 <i>Marvell BG2 SOC</i> platforma	20
Slika 3.2 Arhitektura programske podrške i redosled poziva metoda	28
Slika 3.3 Komunikacija <i>JAVA</i> aplikacije i modula za reprodukciju niskog nivoa	30
Slika 3.4 Arhitektura <i>Presentation Engine</i> modula.....	31
Slika 4.1 Prozor sa opcijama za reprodukciju	33
Slika 4.2 Reprodukcija multimedijalnog sadržaja sa masovne memorije	33
Slika 4.3 Reprodukcija multimedijalnog sadržaja sa posluživača unutar kućne mreže	33
Slika 4.4 Reprodukcija multimedijalnog sadržaja sa posluživača u globalnoj mreži	34
Slika 4.5 Izgled kontrole za reprodukciju multimedije.....	34

SPISAK TABELA

Tabela 2.1 Podržani multimedija kontejneri unutar <i>OpenCORE</i> modula.....	12
Tabela 2.2 Podržani kodeci unutar <i>OpenCORE</i> modula	12

SKRAĆENICE

STB - *Set-Top-Box*, Prijemnik digitalnog televizijskog signala

DLNA - *Digital Living Network Alliance*, Noprofitabilna organizacija, koja omogućuje razmenu multimedijalnih podataka

HTTP - *Hypertext Transfer Protocol*, Aplikativni protokol za komunikaciju

JNI - *Java Native Interface*, Programska sprega u okviru *Android* operativnog sistema

JVM - *Java Virtual Machine*, Virtualna mašina koja izvršava *JAVA* program

DVM - *Dalvik Virtual Machine*, Virtualna mašina unutar *Android* operativnog sistema

IPC - *Inter - Process Communication*, Mehanizam za komunikaciju između procesa u *Android* operativnom sistemu

MIDI - *Musical Instrument Digital Interface*, Protokol za kontrolu električnih instrumenata

RTSP - *Real Time Streaming Protokol*, Mrežni protokol

DMS - *Digital Media Server*, Usluživač u *DLNA* mreži

DMP - *Digital Media Player*, Modul za reprodukciju u *DLNA* mreži

DMC - *Digital Media Controller*, Modul za kontrolu u *DLNA* mreži

PE - *Presentation Engine*, Modul za kontrolu i reprodukciju multimedijalnog sadržaja

SIP - *Session Initiation Protocol*, Mrežni protokol

VoIP - *Voice over IP*, Komunikacioni protokol

FTP - *File Transfer Protocol*, Komunikacioni protokol

1. Uvod

U ovom radu je realizovan podsistem za reprodukciju audio i video sadržaja, prilagođen *Android* prijemnicima digitalnog televizijskog signala. Izvršiti ispitivanje i verifikaciju programske podrške pomoću namenske *Android* aplikacije, koja poseduje mogućnost demonstriranja reprodukcije različitih formata video i audio sadržaja. Prilikom realizacije podsistema za reprodukciju korištena je *Marvell BG2 SOC* platforma.

1.1 Postavka problema

Cilj zadatka je omogućiti višestruku programsku podršku, koja obezbeđuje reprodukciju različitog video i audio sadržaja. Akcenat se stavlja na reprodukciju video i audio sadržaja visokog kvaliteta i rezolucije[8].

Podsistem treba da poseduje mogućnost reprodukovanja lokalnog video i audio sadržaja sa masovne memorije, zatim neophodna je mogućnost reprodukovanja sadržaja sa računara, koji se nalaze unutar kućne mreže (eng. *DLNA*). Realizovani podsistem treba da sadrži i mogućnost prikaza video i audio sadržaja koji pristiže putem toka podataka koristeći *HTTP* komunikacioni protokol.

Realizaciju podsistema za reprodukciju video i audio sadržaja za *Android* bazirane prijemnike digitalnog televizijskog signala potrebno je ispitati i verifikovati pomoću namenske *Android* aplikacije, koja putem iste korisničke sprege može prikazati različite aspekte reprodukcije audio i video sadržaja na *Android* baziranom TV prijemniku [3].

1.2 Analiza problema

Prilikom realizacije potrebno je iskoristiti ugrađenu korisničku spregu koju nudi *Android* operativni sistem i imajući u vidu ciljnu platformu za koju se ovakav podsistem razvija prilagoditi i izmeniti određene funkcionalnosti[2].

Voditi računa o količini sistemskih promena, kako bi se korišćenje programske podrške omogućilo što većem broju različitih digitalnih televizijskih platformi.

Digitalni televizijski prijemnik (eng. *STB*), je uređaj koji obično sadrži tuner i povezuje se na televizijski set. Vrši obradu nad signalom i prikazuje ga na televizijskom ekranu. Koristi se u kablovskim i satelitskim sistemima, kako bi pretvorio signal iz kablovskog ili satelitskog oblika u oblik pogodan za prikaz na ekranu i određenom obradom poboljšao kvaliteta signala [8].

Treba uočiti da se programska podrška za reprodukciju multimedijalnog sadržaja na *Android* operativnom sistemu prostire preko više slojeva arhitekture *Android* operativnog sistema.

Android operativni sistem možemo posmatrati kao programski stek [1]:

- Aplikativni sloj:

Po svojoj prirodi objektno orijentisan i napisan u programskom jeziku *JAVA*.

Aplikacija koja će se koristiti sa ispitivanje i verifikaciju programske podrške će biti smeštena u ovom sloju. Implementacija ostalih slojevi je sakrivena od krajnjeg korisnika i korisnik nema uvid u način njihovog funkcionisanja [3].

- Aplikativno radno okruženje i *JNI* (eng. *Java Native Interface*) sloj:

Objektno orijentisan, pisan u *JAVA* i *C/C++* programskom jeziku i omogućava lak pristup razvojnog okruženju i razvojnoj podršci, koju omogućava *Android* kao operativni sistem. Predstavlja prirodan način za povezivanje sistemskih i aplikativnih komponenti u *Android* operativnom sistemu.

- Sloj niskog nivoa:

Sadrži biblioteke zavisne od platforme, koje omogućavaju funkcionisanje velikog broja različitih sistemskih procesa. Pristupa mu se isključivo putem *JNI* sloja.

U toku realizacije programske podrške za multimedijalnu reprodukciju neophodno je da sloj, koji zavisi od platforme, pretrpi najveće izmene, koji je svakako neosetan ukoliko se posmatra čitava izvorna verzija programske podrške za reprodukciju multimedijalnog sadržaja, jer digitalna televizijska platforma ima svoje specifične detalje o kojima *Android* kao operativni sistem ne vodi računa, na taj način je zadovoljen zahtev o podršci širokog spektra uređaja [9].

Za realizaciju, ranije navedenih, različitih aspekata reprodukcije multimedijalnog sadržaja korišćene su različite tehnike prilagođavanja ciljnoj platformi i to će kasnije biti detaljno opisano.

Neophodno je opisati kako ugrađena programska podrška koja se nalazi u okviru operativnog sistema funkcioniše, kakve formate digitalnog sadržaja podržava u svom izvornom

obliku, koji delovi su suvišni, kako određene stvari utiču na performanse, razlike i sličnosti pri rukovanju video i audio podacima. Detaljnije u sledećem poglavlju.

Nakon razumevanja osnovnih principa na kojima se zasnima ugrađena programska podrška u *Android* operativnom sistemu potrebno je opisati specifične detalje koji su vezani za *Marvell BG2 SOC* ciljnu platformu i način realizacije programskog rešenja, a nakog toga potrebno je opisati način ispitivanja i verifikacije i dobijene rezultate

Ovaj rad sačinjen je od šest poglavlja.

Prvo poglavlje daje kratak opis postavljenih zahteva, kao i kratak opis bitnih pojmoveva za realizaciju rada.

Drugo poglavlje prikazuje arhitekturu *Android* operativnog sistema, module koji pripadaju programskoj podršci za reprodukciju audio i video sadržaja, kao i način komunikacije među njima.

Treće poglavlje prikazuje ciljnu platformu, koncept programskog rešenja i detalje vezane za samo programsko rešenje na ciljnoj platformi.

Četvrto poglavlje prikazuje način ispitivanja i verifikacije programskog rešenja, kao i prikaz dobijenih rezultata.

Peto poglavlje daje kratak rezime o tome šta je urađeno u ovom radu i koje su dalje mogućnosti za napredak.

Šesto poglavlje daje spisak korišćene literature prilikom pisanja rada.

.

2. Teorijske osnove

U ovom poglavlju je dat opis *Android* operativnog sistema, kao i modula koji sačinjavaju podsistem i programsku podršku za reprodukciju audio i video sadržaja na *Android* baziranim prijemnicima digitalnog televizijskog signala.

Operativni sistem *Android* sastoji se iz više programskih slojeva (*software stack*), isto tako programska podrška koja omogućuje reprodukciju multimedijalnog sadržaja podeljena je u više celina [1]. Polazeći od najvišeg sloja, svaki sledeći ima niži nivo apstrakcije, ali isto tako i veću kontrolu nad fizičkim modulima ciljne platforme.

Detaljniji opis ugrađene podrške za reprodukciju multimedije u *Android* operativnom sistemu sledi u daljem tekstu.

2.1 Aplikativni sloj

Aplikativni sloj programske podrške u najvećem delu se oslanja na programski jezik *JAVA* i odgovarajuće korisničke sprege koje olakašavaju komunikaciju *Android* aplikacije sa nižim slojevima u kojima se rukuje hardverskim delom ciljne platforme.

Aplikativni sloj programske podrške, kao takav, takođe poseduje odgovarajuću hijerarhiju, koja se može posmatrati iz dva dela. Jedan koji se tiče grafičkog prikaza i drugi koji se tiče komunikacije sa nižim slojevima programskog steka. U daljem tekstu biće opisani moduli hijerarhije, koji se su od značaja za reprodukciju multimedije.

2.1.1 *View.java*

Osnovni gradivni blok svake *Android* aplikacije je svakako klasa *View.java*, koja je zadužena za prikazivanje grafičkih elemenata na ekranu. Uvek zauzima pravougaoni deo ekrana, određenih dimenzija zadatih parametrima, na željenoj poziciji [10].

2.1.2 *SurfaceView.java*

Klasa od velikog značaja u okviru podrške za reprodukciju multimedijalnog sadržaja, a koja nasleđuje metode klase *View.java*, je upravo *SurfaceView.java*.

SurfaceView modul obezbeđuje pristup grafičkoj ravni, na kojoj je moguće iscrtavati željeni sadržaj. Takođe *SurfaceView* obezbeđuje pozicioniranje u Z ravni (*Z ordering*) [10].

2.1.3 *VideoView.java*

Predstavlja veoma važan deo hijerarhije jer je prvi modul koji je usko povezan sa reprodukcijom multimedijalnog sadržaja i predstavlja gafičku spregu prema krajnjem korisniku. Nasleđuje klasu *SurfaceView.java*, a osnovna namena mu je reprodukovanje video sadržaja, ali se može koristiti i prilikom reprodukovanja audio sadržaja. Putem *MediaPlayerControl.java* sprege implementira različite metode za kontrolu reprodukcije multimedijalnog sadržaja, a najvažnije su:

- *public void setVideoPath(String path)*
- *public void start()*
- *public void stopPlayback()*

Klasa *VideoView* osim što omogućava prikaz video sadržaja predstavlja i modul na vrhu hijerarhije, koja se tiče komunikacije aplikativnog i nižih slojeva, sve do hardverskih modula za reprodukciju multimedijalnog sadržaja.

Komunikacija sa nižim slojevima, što se tiče reprodukcije video i audio sadržaja, počinje pozivom *setVideoPath(String path)* metode, u kojoj se kao parametar prosleđuje putanja do datoteke čiji sadržaj treba reprodukovati na ekranu.

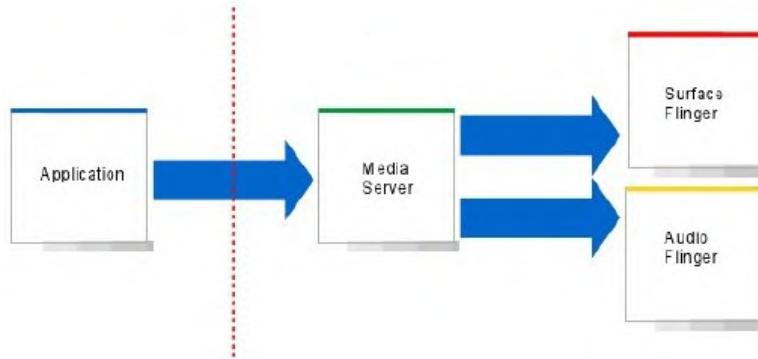
Potom se vrši priprema za reprodukciju. Instancira se objekat klase *MediaPlayer.java* i vrši se njegova priprema i inicijalizacije. Kao parametri mu se prosleđuju putanja do multimedijalne datoteke i referenca na grafičku ravan na kojoj multimedijalni sadržaj treba da bude iscrtan putem *setDataSource()* i *setDisplay()* metoda.

2.2 Aplikativno radno okruženje i *JNI* sloj

2.2.1 *Media Player* radno okruženje

Obuhvata više modula koji zajedno čine najveći deo programske prodrške podsistema za reprodukciju video i audio sadržaja.

Na slici 2.1. prikazana je organizacija *Media Player* radnog okruženja, na kojoj se vidi da se ona zasniva na komunikaciji *Android* aplikacije i posluživača [9].



Slika 2.1 *Media Player* radno okruženje

Aplikacija se izvršava u svom virtualnom prostoru (koji predstavlja zaseban proces u *JVM*), dok se posluživač izvršava u drugom procesu operativnog sistema. Posluživač se inicijalno pokreće prilikom pokretanja operativnog sistema.

Sledi opis komunikacije između aplikacije i posluživača i modula koji se nalaze između njih u hijerarhiji.



Slika 2.2 Komunikacija modula programske podrške

Na prethodnoj slici grafički je predstavljen tok komunikacije prilikom iniciranja reprodukcije multimedijalnog sadržaja, od aplikativnog do najnižeg nivoa (eng. *Function Call Stack*) [5].

Moduli prikazani na grafu su:

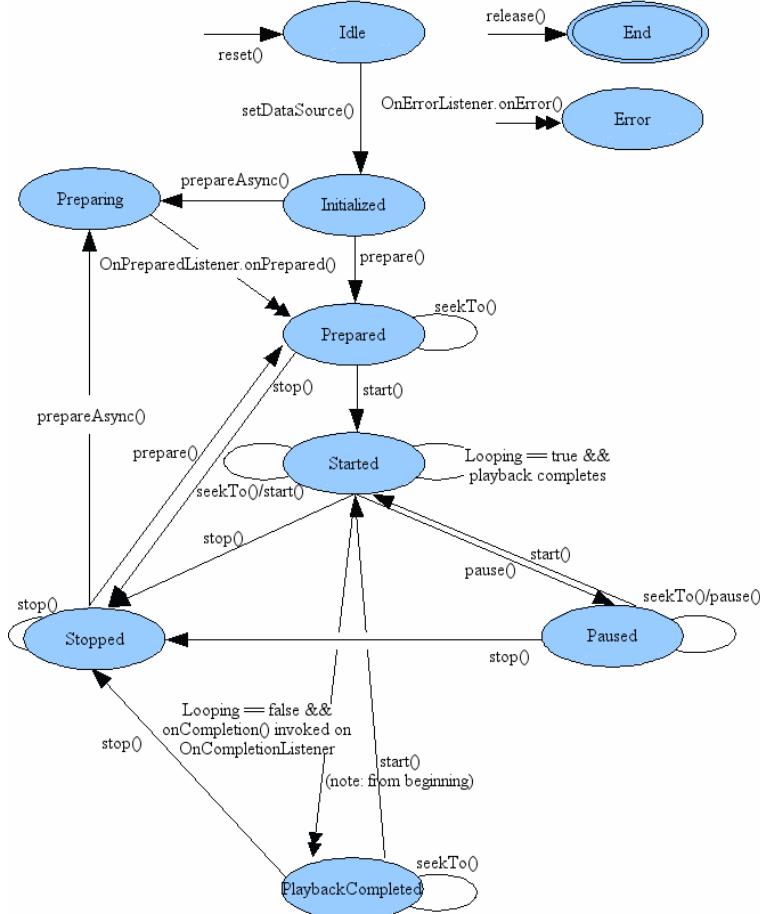
- DVM Proxy (*Dalvik Virtual Machine Proxy*) – *MediaPlayer.java*
- JNI (*Java Native Interface*) – *android_mediaplayer.cpp*
- Native Proxy – *mediaplayer.cpp*
- Binder proxy i Binder Native – *IMediaPlayer.cpp*
- Native Implementation – *MediaPlayerService.cpp*

2.2.2 *MediaPlayer.java*

Klasa *MediaPlayer.java* za cilj ima kontrolu reprodukcije video i audio datoteka, kao i multimedijalnog toka podataka sa mreže.

Kontrola multimedijalnih podataka je realizovana kao automat sa konačnim brojem stanja.

Sledeći dijagram prikazuje životni vek i stanja u kojima se može naći objekat klase *MediaPlayer.java* [10].



Slika 2.3 Dijagram stanja *MediaPlayer.java* modula

Elipsama su prikazana stanja, dok su prelazi prikazani strelicama.

Postoje dve vrste prelaza. Prelazi označeni strelicom na jednom kraju predstavljaju sinhronne događaje, dok prelazi sa strelicama na oba kraja predstavljaju asinhronne događaje.

Sa dijagrama možemo uočiti da *MediaPlayer* modul ima sledeća stanja:

Nakon instanciranja ili nakon dovođenja u početno stanje, objekat se nalazi u inicijalnom stanju (eng. *Idle*), dok se nakon pozivanja *release()* metode objekat dovodi u završno stanje (eng. *End*). Objekat *MediaPlayer* klase se smatra “živim” dok god se nalazi u nekom između prethodno dva navedena stanja.

U nekim slučajevima moguće je da *MediaPlayer* ne uspe da reprodukuje multimedijalni sadržaj iz različitih razloga: korišćeni algoritam za kodovanje nije podržan, željena rezolucija nije podržana, isteklo vreme za sinhronizaciju (eng. *timeout*) kod mrežnog toka podataka, itd.

Pozivanjem *setDataSource()* metode *MediaPlayer* prelazi iz inicijalnog stanja (eng. *Idle*) u inicijalizovano stanje (eng. *Initialized*).

MediaPlayer se mora postaviti u stanje “Spreman” (eng. *Prepared*) pre nego što je moguće započeti reprodukciju video ili audio sadržaja.

Postoje dva načina da se zadovolji ovakav uslov: pozivanjem sinhronne metode *prepare()*, koja dovodi objekat u željeno stanje tek kada se metoda u potpunosti završi, ili pozivanjem asinhronne metode *asyncPrepare()*, koja dovodi objekat u željeno stanje gotovo istovremeno, dok unutrašnja nit radi na preostaloj pripremi sve dok se ona u potpunosti ne završi.

Ukoliko se objekat nalazi u stanju “Spreman”, pozivanjem metode *start()* započeće reprodukcija željenog medija sadržaja, a objekat će se postaviti u stanje “Startovan” (eng. *Started*).

Reprodukcijski se u svakom trenutku može pauzirati ili zaustaviti pozivanjem metode *pause()*, odnosno *stop()*. Instanca *MediaPlayer* klase se u tom slučaju postavlja u stanje “Pauziran” (eng. *Paused*) ili “Zaustavljen”(eng. *Stopped*), respektivno.

U slučaju kada se objekat nalazi u stanju “Zaustavljen”, reprodukciju nije moguće započeti dok god se ponovo ne pozove metoda *prepare()* ili *asyncPrepare()*.

Na kraju, nakon završetka reprodukcije, tj. kada multimedijalni sadržaj dosegne kraj, objekat se postavlja u stanje “ReprodukcijskaZavršena” (eng. *PlaybackCompleted*).

2.2.3 android_mediaplayer.cpp

Napisan u C/C++ programskom jeziku. Omogućava da JAVA kod, koji se izvršava u *JVM* (eng. *Java Virtual Machine*), komunicira sa aplikacijama zavisnim od platforme u nižim slojevima hijerarhije.

2.2.4 mediaplayer.cpp

Predstavlja C++ objekat koji se prosleđuje ka nižim slojevima kroz *Binder Interface* modul. *mediaplayer.cpp* omogućava postojanje više instanci objekta za reprodukciju multimedijalnog sadržaja. Kada se u aplikativnom sloju instancira objekat za reprodukciju video ili audio sadržaja, preko *JNI* sloja, instancira se i C++ objekat. *JAVA* objekat u sebi sadrži “plitku” referencu (eng. *weak reference*) C++ objekta, kako bi olakšao postupak recikliranja (eng. *garbage collection*).

2.2.5 IMediaPlayer.cpp

Binder predstavlja apstrakciju *IPC*-a (eng. *Inter-Process Communication*). Omogućava kontrolu i deljenje objekata između procesa. Procesima je omogućen pristup deljenoj memoriji, što je važno zbog deljenja objekata između aplikacije i samog modula nikog nivoa za reprodukciju multimedije, jer se aplikacija i medija posluživač nalaze u različitim procesima operativnog sistema.

Najveću primenu nalazi pri rukovanju *SurfaceFlinger* i *AudioFlinger* modulima (unutar *MediaServer* procesa).

SurfaceFlinger i *AudioFlinger* moduli predstavljaju apstrakcije za video, odnosno audio podršku.

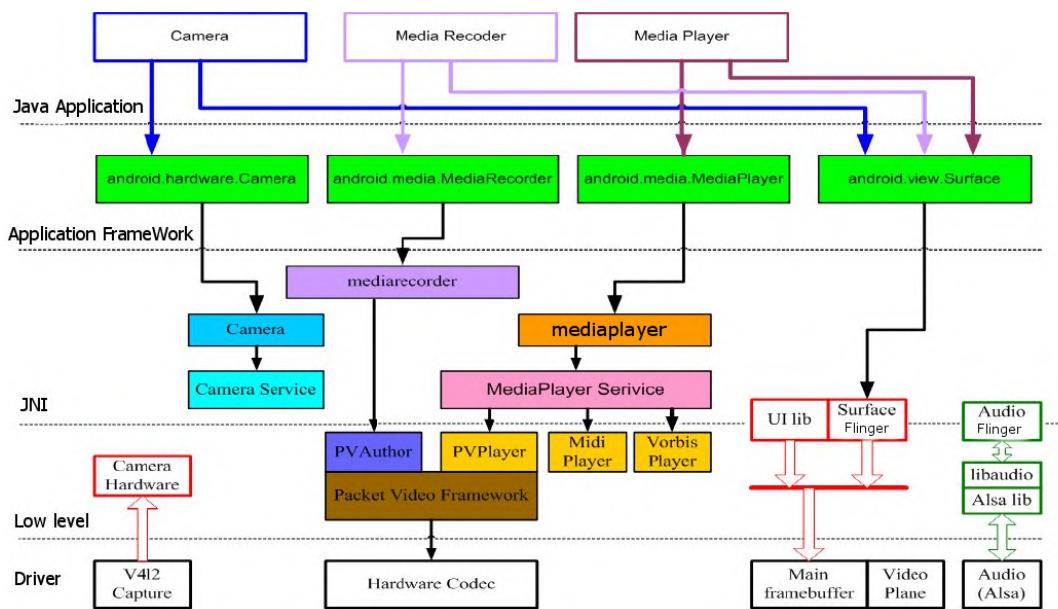
Binder proxy modul se koristi za kontrolu funkcionalnosti sa strane aplikativnog sloja, dok se *Binder Native* izvršava sa strane medija posluživača.

2.2.6 MediaPlayerService.cpp

Na osnovu formata datoteke, koju treba reprodukovati, određuje se koji modul za reprodukciju treba instancirati. Instancirani modul je zapravo odgovoran za reprodukciju audio i video sadržaja i predstavlja “srce” multimedijalne podrške, preko sistemskog programa (eng. *driver*) komunicira sa fizičkim komponentama same ciljne platforme i vrši obradu video i/ili audio sadržaja.

Slika 2.4 prikazuje detaljnu organizaciju programske podrške za reprodukciju multimedijalnog sadržaja [2].

Lako se može primetiti da u hijerarhiji poziva funkcija u programskoj podršci za reprodukciju multimedijalnog sadržaja postoji veliki broj “viškova” i redundancije u algoritmu (eng. *overheads*). To se može tolerisati iz razloga što se ova procedura ne izvršava često. Zbog toga potrebno je uložiti dodatni trud, kako bi se iskorišćenost objekata za reprodukciju i mogućnost korišćenja već postojećih objekata u memoriji što više povećala.



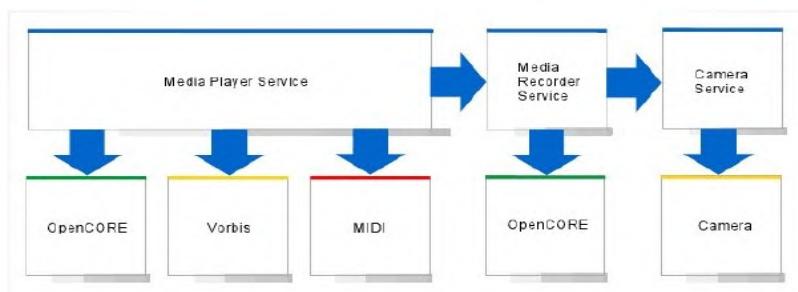
Slika 2.4 Arhitektura programske podrške za reprodukciju multimedije

2.3 Sloj niskog nivoa

2.3.1 *main_mediaserver.cpp*

Medija poslužilac (*MediaServer* modul) je proces koji se pokreće prilikom uključivanja uređaja, tj. prilikom "podizanja" operativnog sistema. Njegova funkcija jeste da inicijalizuje i instancira odgovarajuće servise, koji na odgovarajući način rukuju multimedijalnim podacima. Jedan od servisa, koji je od velikog značaja za programsku podršku za reprodukovanje multimedijalnog sadržaja, a koje se instancira i pokreće unutar *MediaServer* procesa, jeste *MediaPlayerService* [5].

Sledeća slika prikazuje *MediaServer* proces i njegove gradivne blokove.

Slika 2.5 Organizacija *MediaServer* procesa

Uočava se da u sklopu programske podrške za reprodukciju multimedijalnog sadržaja, može da postoji više različitih modula, koji su sposobni da reprodukuju multimedijalni sadržaj. A *MediaPlayerService* može da instancira više od jednog modula [6].

Svaki od modula za reprodukciju audio ili video sadržaja sadrži odgovarajući kodek.

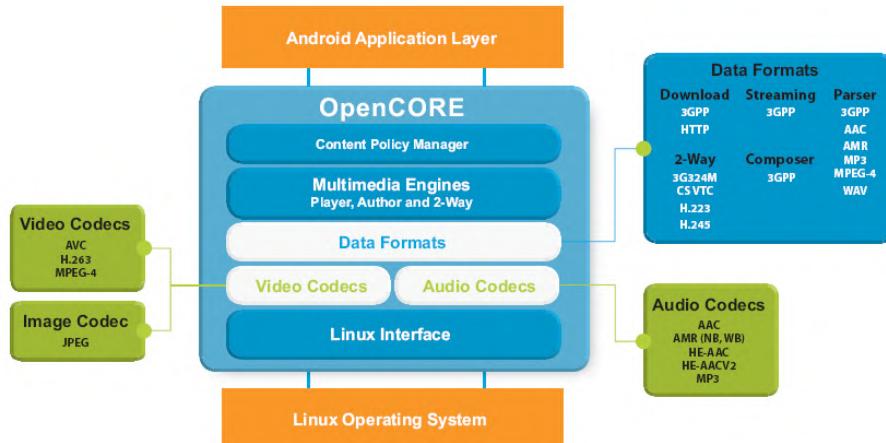
Koder vrši kodovanje toka multimedijalnih podataka i sprema ih za slanje ili smeštanje na masovnu memoriju.

Dekoder vrši dekodovanje multimedijalnih podataka i priprema ih za reprodukovanje [11].

U daljem tekstu sledi kratak opis nekih od ugrađenih modula za reprodukciju multimedijalnog sadržaja u *Android* operativnom sistemu:

- *Vorbis Player* – open source codec, jednostavan i zauzima malo memorijskih resursa. Oslanja se psihoh-akustički modela ljudskog uha, što znači da se prilikom kodovanja u velikoj meri oslanja na anatomiju ljudskog uha. Prepoznaje frekvencije koje su od manjeg značaja (van ospega čujnosti) ili maskirane delove i izostavlja ih iz procesa kodovanja. Na taj način se dobija velika ušteda u pogledu memorijskih resursa i brzine protoka. Obično se koristi za reprodukovanje sistemskih zvuka.
- *MIDI Player* – Poseduje mogućnost reprodukcije detatoka *MIDI* (eng. *Musical Instrument Digital Interface*) formata.
- *OpenCORE* – Veoma važan deo *MediaServer* procesa, jer poseduje mogućnost da se instancira prilikom reprodukcije svih drugih multimedijalnih formata osim dva prethodno navedena.

Poseduje podršku za najvažnije multimedijalne formate kao i algoritme za kodovanje i dekodovanje.



Slika 2.6 Arhitektura *OpenCORE* modula

Na prethodnoj slici prikazana je organizacija i arhitektura *OpenCORE* modula za reprodukciju multimedije.

Sledi tabelarni prikaz podržanih algoritama za kodovanje i dekodovanje:

<i>Containter format</i>	<i>Parser</i>	<i>Composer</i>
<i>MPEG4</i>	<i>YES</i>	<i>YES</i>
<i>3GPP</i>	<i>YES</i>	<i>YES</i>
<i>3GPP2</i>	<i>YES</i>	<i>NO</i>
<i>MP3</i>	<i>YES</i>	<i>NO</i>
<i>AAC</i>	<i>YES</i>	<i>NO</i>
<i>AMR</i>	<i>YES</i>	<i>YES</i>
<i>WAV</i>	<i>YES</i>	<i>NO</i>

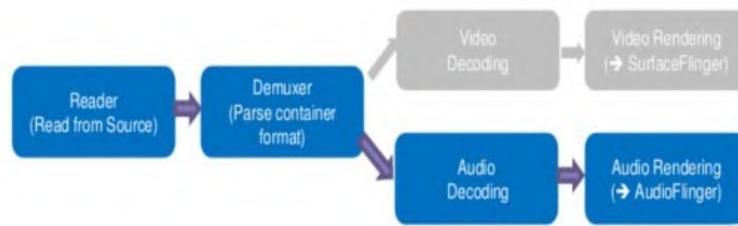
Tabela 2.1 Podržani multimedija kontejneri unutar *OpenCORE* modula

<i>Type</i>	<i>Codec</i>	<i>Decoder</i>	<i>Encoder</i>
<i>Video</i>	<i>H.263</i>	<i>YES</i>	<i>YES</i>
	<i>MPEG-4</i>	<i>YES</i>	<i>YES</i>
	<i>AVC/H.264</i>	<i>YES</i>	<i>YES</i>
<i>Audio</i>	<i>MP3</i>	<i>YES</i>	<i>NO</i>
	<i>AAC</i>	<i>YES</i>	<i>NO</i>
	<i>AMR-NB</i>	<i>YES</i>	<i>YES</i>
	<i>AMR-WB</i>	<i>YES</i>	<i>NO</i>

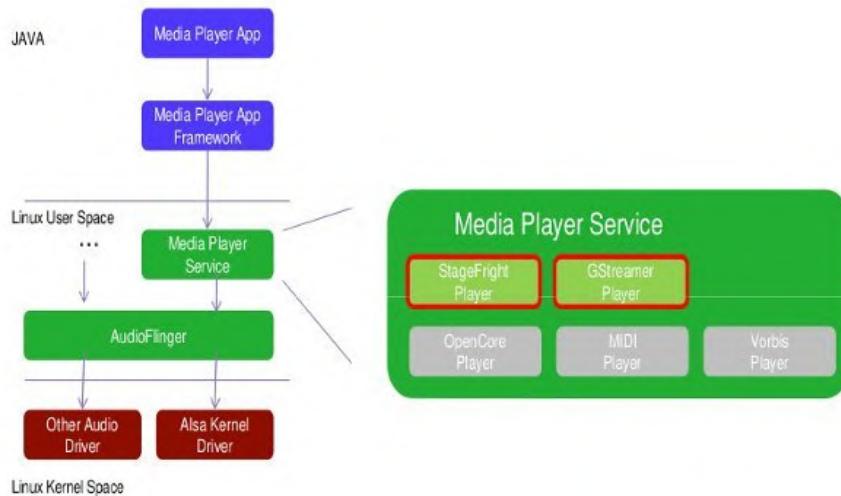
Tabela 2.2 Podržani kodeci unutar *OpenCORE* modula

Na slici 2.7 prikazan je tok multimedijalnih podataka kroz *OpenCORE* modul.

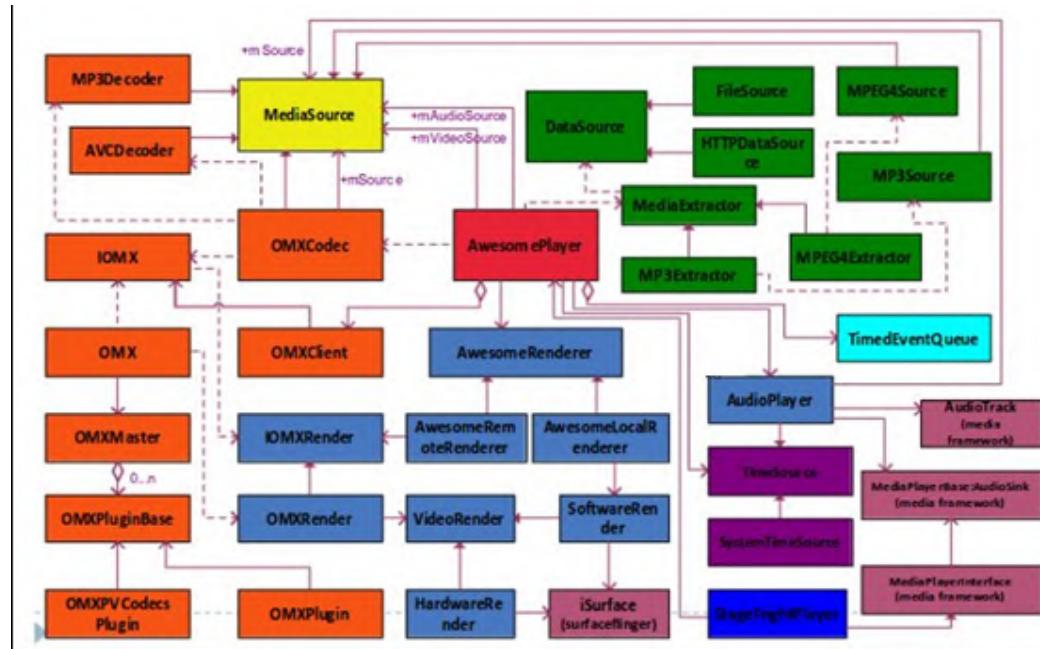
Reader preuzima podatke sa izvora i prosluđuje ih *Demuxer* modulu, koji na osnovu tipa i formata podaka, dalje usmerava tok video, odnosno audio podataka, dekoderu. a ovaj dalje modulima za reprodukciju.

Slika 2.7 Tok podataka unutar *OpenCORE* modula

- Kao dodatak osnovnom *MediaServer* modulu, odnosno *MediaPlayerService* modulu dodaje se još jedan modul za reprodukciju multimedijalnog sadržaja. *StageFright* modul poseduje mogućnost reprodukovanja različitih video i audio formata slično kao i *OpenCORE* modul, ali predstavlja mnogo jednostavnije i bolje programsko rešenje, takođe poseduje mogućnost da na osnovu formata datoteke instancira modul za dekodovanje i modul za usmeravanje (eng. *demux*). Na sledećoj slici prikazan je izgled hijerarhije nakon dodavanja *StageFright* modula u podsistem za reprodukovanje multimedijalog sadržaja.

Slika 2.8 Arhitektura programske podrške sa *StageFright* modulom

Na slici 2.9. prikazana je organizacija modula i arhitektura *StageFright* ugrađenog modula za reprodukciju multimedijalnog sadržaja.



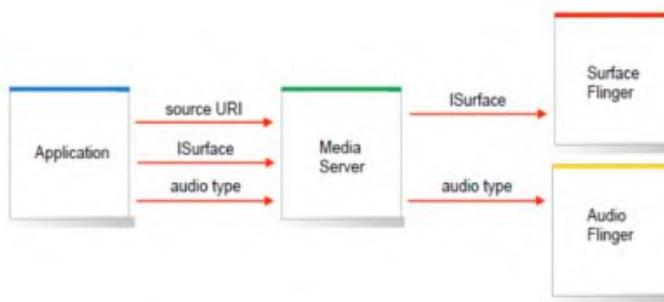
Slika 2.9 Arhitektura *StageFright* modula

- *DataSource* – Sadrži putanju do toka podataka koje treba reprodukovati. Predstavlja završnu tačku u hijarhiji poziva funkcija, prilikom iniciranja reprodukcije multimedijalnog sadržaja, opisanoj na slici 2.2.
- *MediaExtractor* – Prikuplja podatke iz toka podataka, kao i podatke koji bliže opisuju trenutni tok podataka
- *OMXPlugin* – Postoje dva *OMXPlugin* dodatka, gde *OMXPV Codecs Plugin* omogućava programsko dekodovanje, a *OMXPlugin* omogućava pristup fizičkim modulima za dekodovanje.
- *AudioPlayer* – Modul odgovoran za reprodukovanje zvuka. Kada je zvuk prisutan, određuje vremenu rezoluciju neophodnu za sinhronizaciju zvuka i videa.
U zavisnosti od vrste odabranog algoritma za dekodovanje videa, odgovarajući modul za reprodukciju će biti instanciran. Ukoliko je u multimedijalnom toku podataka prisutan samo video signal, sistemsko vreme uređaja se koristi za sinhronizaciju video paketa.
- *AwesomePlayer* – Rukuje gore navedenim modulima u okviru *StageFright* modula. Obezbeđuje reprodukciju multimedijalnog sadržaja sa masovne memorije, *HTTP* i

RTSP tokove podataka. Povezan je sa *MediaServer* modulom preko sprege koju obezbeđuje *StageFright*.

Nakon detaljnog opisa *MediaServer* procesa moguće je nastaviti opis programske podrške podistema za reprodukciju multimedijalnog sadržaja na višem nivou apstrakcije.

Sledeća slika prikazuje način na koji *JAVA* aplikacija uz pomoć *MediaServer* procesa komunicira sa modulima niskog nivoa, kada je prethodno izvršena inicijalizacija svih neophodnih modula unutar podistema za reprodukciju audio i video sadržaja.



Slika 2.10 Komunikacija *JAVA* aplikacije sa modulima nižeg nivoa

Aplikacija prosleđuje tri vrste podataka ka nižim slojevima.

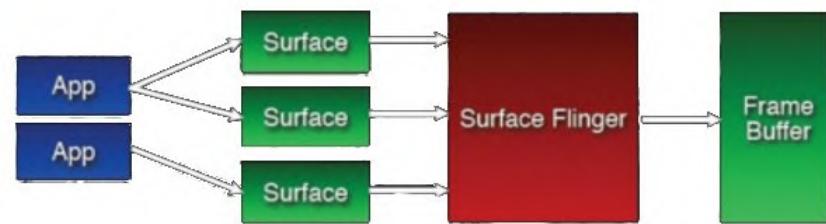
- *Source URI* – Putanja do datoteke na spoljnoj memoriji, putanja do datoteke u resursima aplikacije ili do toka podataka na mreži.
- *Surface* – Apstrakcija grafičkog prikaza
- *Audio type* – Na osnovu ovog podatka *MediaPlayerService* može pravilno da usmeri pakete ka nekom od modula koji je sposoban da reprodukuje audio sadržaj zadatog formata.

Raspakovani video i audio paketi unutar *MediaServer* procesa se prosleđuju *AudioFlinger*, odnosno *SurfaceFlinger* modulima.

Ovakva komunikacija za razliku od iniciranja programske podrške ne sadrži znatnu količinu “viškova” i redundancije (eng. *overheads*), jer se podaci ne posleđuju nazad do *JAVA* aplikacije, što predstavlja značajnu uštedu.

2.3.2 *LayerBuffer.cpp* (*SurfaceFlinger*)

Surface predstavlja objekat koji sadrži informaciju o pikselima koji treba da se prikažu na ekranu. *Surface* objekti se posleđuju u obliku memoriske strukture kroz *Binder IPC* korisničku spregu.

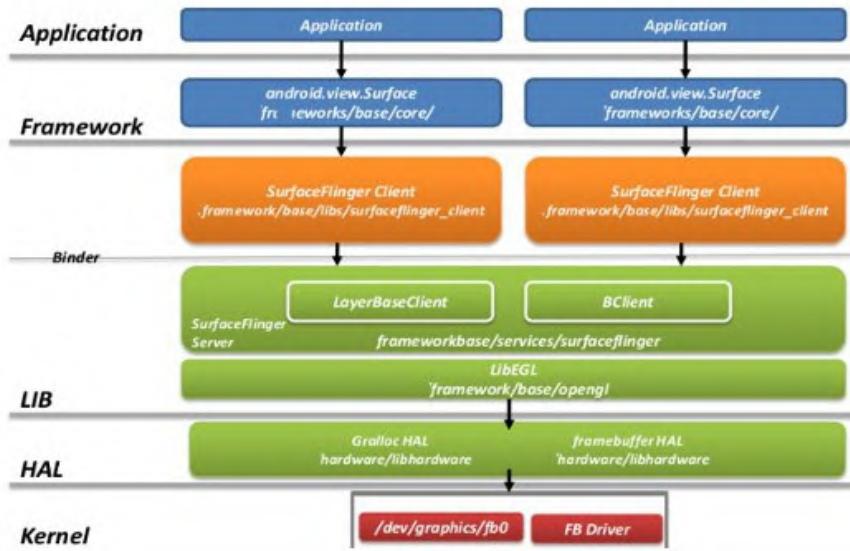


Slika 2.11 Komunikacija *JAVA* aplikacija i *Surface Flinger* modula

SurfaceFlinger predstavlja modul koji rukuje prozorima unutar *Android* operativnog sistema. Svaki prozor (eng. *dialog*, eng. *status bar*) se predstavlja kao *OpenGL* struktura i sadrži informacije o delu ekrana na kom se iscrtava, takođe predstavlja i zaseban grafički sloj (eng. *layer*), koji je na određeni način raspoređen u Z ravni (eng. *Z ordering*).

SurfaceFlinger kombinuje različite *2D* i *3D* *Surface* objekte i spaja ih u jednu *OpenGL* strukturu oslanjajući se na metode definisane *OpenGL ES 1.1* standardom i hardversku *2D* akceleraciju. Sadrži dve memoriske strukture kako bi omogućio *double buffering* režim rada. Na taj način aplikacija može da osvežava svoju grafiku, u trenutku dok se na ekran iscrtava sadržaj definisan u prethodnoj memorijskoj strukturi, a takođe se smanjuje i se verovatnoća pojavljivanja različitih artifaka u slici, npr. “preklapanje” (eng. *aliasing*).

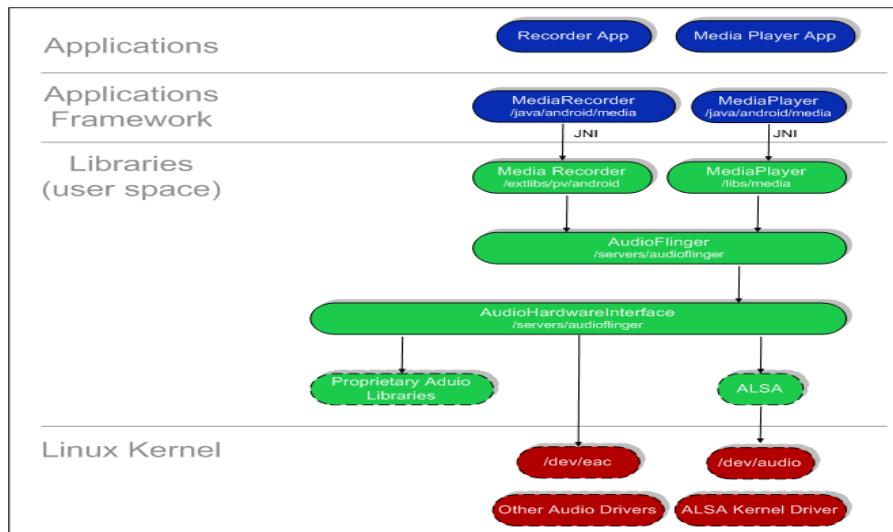
Na slici 2.11. i 2.12. prikazan je način na koji je *SurfaceFlinger* modul povezan sa *JAVA* aplikacijama i *FrameBuffer* modulom, koji se nalazi u *Linux Kernel* prostoru operativnog sistema i na koji se način *Surface* objekat posleđuje od aplikativnog do sloja niskog nivoa [11].



Slika 2.12 Komunikacija JAVA aplikacije sa *FrameBuffer* modulom

2.3.3 *AudioFlinger.cpp*

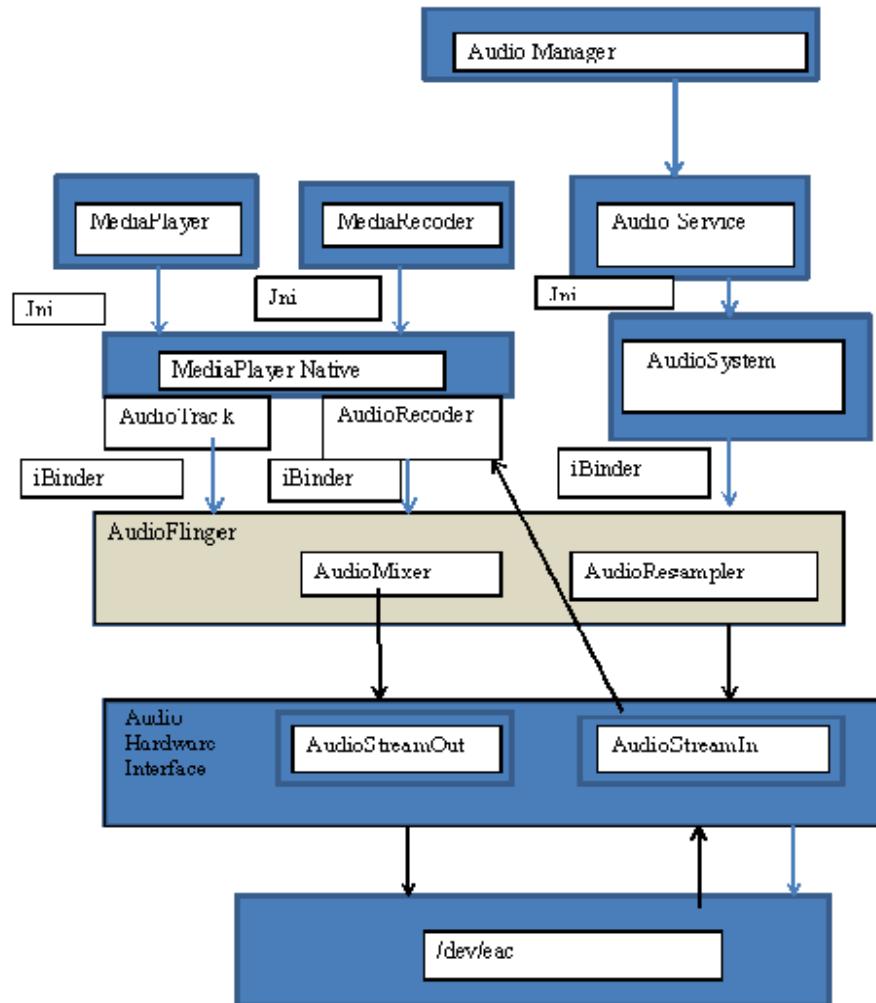
Mehanizam koji je ugrađen u *Android* operativni sistem baziran je na korisnik-usluživač arhitekturi [9]. Gde korisnika predstavlja JAVA korisnička aplikacija, a sa strane usluživača se nalazi *AudioFlinger* modul, koji se pokreće unutar *MediaService* procesa prilikom iniciranja programske podrške za reprodukciju multimedijalnog sadržaja. Realizacija *AudioFlinger* modula je sakrivena od korsničkog dela, dok je komunikacija obezbeđena kroz odgovarajuću korisničku spregu.



Slika 2.13 Arhitektura programske podrške za reprodukciju zvuka

Na prethodnoj slici prikazana je arhitektura ugrađene programske podrške *Android* operativnog sistema za reprodukciju zvučnog sadržaja.

Najvažniju ulogu u ovako organizovanoj arhitekturi igraju *AudioFlinger* i *AudioHardwareInterface* moduli.



Slika 2.14 Arhitektura programske podrške za reprodukciju zvučnog sadržaja

AudioFlinger predstavlja deo programske podrške koji obezbeđuje osnovne kontrole nad reprodukcijom zvučnog sadržaja. Zadužen je za rukvanje i kombinovanje tokova podataka sa zvučnim sadržajem i prosleđuje ih *AudioHardwareInterface* modulu. Tok podataka koji sadrži zvuk sadrži i informacije o formatu i tipu toka podaka, na osnovu kojih se kontrolišu podešavanja u pogledu nivoa reprodukcije i usmeravanja ka modulima nižeg nivoa. *AudioHardwareInterface* predstavlja apstrakciju koja sakriva detalje specifične za platformu i komunikaciju sa sistemskim programima od krajnjeg korisnika.

Na slici 2.14 je prikazan detaljniji izled arhitekture programske podrške za reprodukciju zvučnog sadržaja.

Android kao operativni sistem obezbeđuje različite module koji su sposobni da reprodukuju tok audio podaka[10].

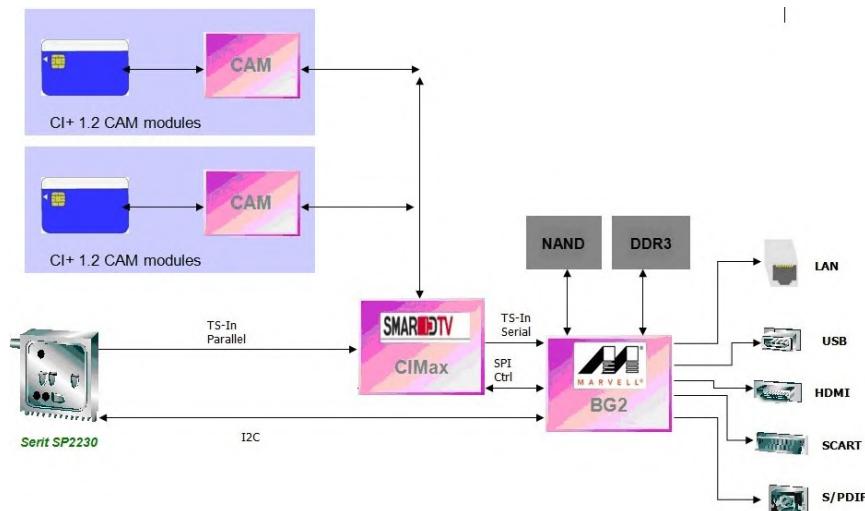
- *AudioTrack* – poseduje mogućnost reprodukcije “sirovog” toka podatka (eng. *raw data stream*), ali samo .*wav* formata.
- *MediaPlayer* – detaljno opisan u prethodnim poglavljima. Najveći problem predstavlja brzina obrade podataka i nemogućnost reprodukovanja “sirovih” podataka.
- *SoundPool* – Veoma jednostavan i brz. Nije u stanju da reprodukuje tok podataka velike veličine, jer pre početka reprodukcije podatke prevodi u “sirov” oblik. Veliki nedostatak predstavlja manjak kontrole nad reprodukcijom toka podataka.
- *AsyncPlayer* – Poseduje mogućnost reprodukcije više tokova podataka istovremeno. Svu pripremu i obradu podataka izvršava u zasebnoj niti, na taj način, ako dođe do nekog zagušenja unutar sistema, to neće imati posledice na reprodukciju audio sadržaja u pogledu različitih neprijatnih artifakata. Poseduje veoma jednostavnu kontrolu za reprodukciju (samo medote za startovanje i zaustavljanje).
- *Jet Player* – Poseduje mogućnost reprodukovanja samo *JET* datoteka.

3. Analiza rešenja

U ovom poglavlju opisana ciljna platforma i pristup koji je upotrebljen prilikom realizacije programskog rešenja, kao i detalji vezani za samu implementaciju.

3.1 Opis ciljne platforme

Sledi opis ciljne platforme korišćene prilikom realizacije podsistema za reprodukciju audio i video sadržaja kao i njeni najbitiniji moduli.



Slika 3.1 Marvell BG2 SOC platforma

Ključne komponente:

- Satelitski birač kanala tipa *Serit SP2230* – koristi se za preuzimanje signala sa satelita
- *CIMax* čip proizvođača *SmarDTV* – koristi se za usmeravanje prenosnog toka
- Dva modula uslovnog pristupa – koriste se za dešifrovanje zaštićenog prenosnog toka

- Procesor *ARM* familije, *Marvell BG2 88DE3100* [12]
- *DDR3* operativna memorija *Micron MT4IJ2G4THA* – memorija koja se stavlja na raspolaganje sistemu
- *NAND* memorija *Micron MT29F16G08* – memorija koja se koristi za smeštanje aplikacije na ciljnoj platformi

3.2 Koncept rešenja

Prilikom realizacije podistema za reprodukciju audio i video sadržaja na *Android* baziranim prijemnicima digitalnog televizijskog signala, uporedo sa implementacijom, treba voditi računa i o načinu ispitivanja i verifikovanja funkcionalnosti čitave programske podrške.

Shodno tome najbolji pristup realizaciji ovakvog podistema bi bio simultano razvijanje namenske aplikacije, koja će služiti za ispitivanje i verifikaciju, i programske podrške podistema za reprodukciju multimedijalnog sadržaja.

Aplikaciju za ispitivanje i verifikaciju je potrebno razviti u programskom jeziku *JAVA*, razviti odgovarajuće grafičko okruženje koje će služiti kao sprega prema krajnjem korisniku.

Aplikacija treba da ima mogućnost odabira vrste multimedijalnog sadržaja i izvora sa kojeg ona potiče (lokalna datoteka, datoteka koja se nalazi na nekom drugom računaru unutar mreže, tok podataka sa globalne mreže) i mogućnost interakcije sa krajnjim korisnikom putem daljinskog upravljača.

Nakon odabira želenog sadržaja aplikacija treba da demonstrira funkcionalnost programske podrške tako što će reprodukovati željeni sadržaj i obezbediti osnovne kontrole za reprodukciju.

Pošto se *JAVA* aplikacija izvršava na ciljnoj platformi digitalnog televizijskog prijemnika, za koju se programska podrška razvija, treba odmah imati u vidu sve specifične detalje i ograničenja koja ona nosi.

Na ovaj način ispitivanje će se umnogome olakšati, jer će postojati mogućnost da se u svakom trenutku proveri stanje programske podrške, jednostavnim pokušajem reprodukovanja nekog od ponuđenih multimedijalnih sadržaja.

Pošto ugrađena programska podrška u *Android* operativnom sistemu nije u mogućnosti da reprodukuje neke od formata video sadržaja, koji su bitni za ciljnu platformu jer poseduju veliku rezoluciju i određene dodatne sadržaje pored video podataka (eng, *television stream*), ili ne pruža zadovoljavajuće performanse i vreme odziva, potrebno je izmeniti deo originalne programske podrške koji zavisi od platforme i dodati odgovarajuća poboljšanja. To se odnosi na vremensko skraćivanje određenih procedura, ili čak izbacivanje nepotrebnih delova programske podrške, koji negativno utiču na performanse celog podistema. Takođe, potrebno je dodati module koji

podržavaju nepostojeće algoritme dekodovanja u originalnoj programskoj podršci i omogućiti podsistemu za reprodukciju da komunicira s njima.

Pored modula koji poseduju mogućnost reprodukovanja multimedijalnog sadržaja, opisanih u prethodnom poglavlju, potrebno je dodati još jedan zaseban modul, koji je sposoban da rukuje različitim tipovima multimedijalnih datoteka i zadovoljava detalje koji su specifični za ciljnu platformu.

Nakon toga neophodno je izmeniti ostale slojeve programskog steka, kako bi omogućili komunikaciju izmedju modula originalne, već postojeće programske podrške, i modula specifičnih za ciljnu platformu.

Na taj način je omogućeno da aplikativni sloj i sama *JAVA* aplikacija za verifikaciju i ispitivanje komunicira sa specijalnim modulima za reprodukciju i iskoristi njihove mogućnosti prilikom reprodukcije različitog multimedijalnog sadržaja na zahtev krajnjeg korisnika.

Kako je već ranije zaključeno, programska podrška za reprodukciju multimedijalnog sadržaja sastoji iz više različitih slojeva i modula, prilikom njenog implementiranja svaki od njih se nazavisno razvija. Uvek je potrebno imati na umu kako su ovi slojevi međusobno povezani u ugrađenoj programskoj podršci, kako bi se što je više moguće zadržao isti koncept i način funkcionisanja i smanjilo broj detalja koji zavise od ciljne platforme, tj. zadržati transparentnost programskog rešenja.

3.3 Aplikativni sloj

Android aplikacija je neophodna za ispitivanje i verifikovanje kvaliteta i funkcionalnosti podistema, jer predstavlja najbolji indikator stanja programske podrške. Razvija se na najvišem nivou apstrakciju u poređenju sa ostalim slojevima unutar programske podrške i predstavlja direktnu vezu sa krajnjim korisnikom i pri tome sakriva od njega sve detalje implementacije.

Android aplikacija ima vrlo jednostavan i intuitivnu grafičku spregu ka korisniku. Poseduje mogućnost odabira ispitinih datoteka, koje su unapred definisane prilikom razvoja aplikacije.

Iz namenskog menija korisnik može da odabere vrstu multimedijalne datoteke koju želi da reprodukuje: video ili audio.

Nakon toga može da odabere neku od pomuđenih opcija za reprodukovanje:

- *File from SD card* – reprodukovanje ispitne sekvence koja se nalazi na unapred definisanoj lokaciji na spoljnoj memoriji ciljne platforme.
 - *File from DLNA server (DMS)* – reprodukovanje ispitne sekvence koja je smeštena na unapred definisanoj lokaciji na posluživaču unutar kućne mreže (eng. *DMS*).
- Dodatkom ovakve mogućnosti podsistemu za reprodukciju, ciljna platforma se može posmatrati kao *Digital Media Player* (eng. *DMP*) [4].

DMP poseduju mogućnost prolaženja i pristupa multimedijalnim datoteka u okviru *DLNA* mreže, i transparentni su u odnosu na kontrolere *DLNA* mreže (eng. *DMC*).

- *Internet stream* – reprodukovanje toka podataka koji dolazi sa posluživača iz globalne mreže.

Nakon odabira neke od ponuđenih opcija započinje reprodukovanje multimedijalnog sadržaja, uz osnovne kontrole za reprodukciju: *START*, *PAUSE*, *FORWARD*, *REWIND* i *SEEK*.

Pošto se aplikacija izvršava na digitalnom televizijskom prijemniku, interakcija sa aplikativnom grafičkom spregom se odvija putem daljinskog upravljača, što predstavlja prirodni način komunikacije za svaki televizijski prijemnik.

3.3.1 Detalji realizacije

Pošto se aplikacija koristi isključivo u svrhe ispitivanja i verifikacije i kao krajnja demonstracija mogućnosti podsistema za reprodukciju multimedijalnog sadržaja, nije neophodno da poseduje složenu grafičku spregu.

Aplikacija se sastoji iz jedne aktivnosti (eng. *Activity*), što znači da ne postoji mogućnost prelaska na druge „ekrane“, jer to nije neophodan detalj.

Osnovni element grafičke sprege čini element *VideoView*, koji je opisan u poglavljju 2.1.3.

Aplikacija u svojoj *onCreate()* metodi, koja se poziva prilikom pokretanja, instancira objekat klase *VideoView* i postavlja ga na ekran. Dimenzije *VideoView* objekta podešene su tako da zauzima prostor čitavog ekrana. Na taj način se krajnjem korisniku omogućuje utisak korišćenja klasičnog televizijskog prijemnika.

```
//Create object of video
VideoView video = new VideoView(getApplicationContext());

//Set video as a screen content
setContentView(video);
```

Nakon instanciranja modula koji će biti zadužen za reprodukciju multimedijalnog sadržaja, neophodno je omogući osnovne kontrole za reprodukciju. Za to je zadužen objekat klase *MediaController*. Funkcionalnost *MediaController* objekta se povezuje sa *VideoView* modulom, nakog čega oni čine jednu celinu, koja obezbeđuje i reprodukciju audio i video sadržaja i kontrolu nad njim.

```
//Create object of media controller
MediaController controller = new MediaController(getApplicationContext());
```

```
//Set video view object as a parent for controller
controller.setAnchorView(video);
controller.setMediaPlayer(video);

//Attach controller to view view object
video.setMediaController(controller);
```

Naravno potrebno je omogućiti aplikaciji da prepoznaže kada korisnik koristi daljinski upravljač. U tu svrhu neophodno je iskoristiti privatnu klasu *MultimediaKeyListener*, koja implementira metode *OnKeyListener* korisničke sprege. Funkcionalnost određenih tastera na daljinskom upravljaču u implementiranim metodama treba prilagoditi ciljnoj platformi.

Korisniku treba omogućiti da pritiskom na definisani taster prikaže ponuđene opcije za reprodukciju i da nakon odabira odgovarajuće opcije, reprodukcija multimedijalnog sadržaja automatski započne.

```
public boolean onKey(View v, int keyCode, KeyEvent event){

    //Check if user pressed button on remote
    if(event.getAction() == KeyEvent.ACTION_DOWN){

        //Check which button is pushed
        switch(keyCode){
            case KeyEvent.KEYCODE_MENU:

                //On Menu button show options for playback
                optionsDialog.show();

                return true;

            }

        }

        case KeyEvent.KEYCODE_BACK:

            //Close application
            System.runFinalizersOnExit(true);
            android.os.Process.killProcess(android.os.Process.myPid());

            return true;

        }

    }

    default:
        //Do nothing
        return false;
}
```

```

        }

    }

    return true;
}
}

```

Zatim *MultimediaKeyListener* objekat sa *VideoView* objektom, kako bi aplikacija reagovala na pritiske tastera daljinskog upravljača.

```
//Attach keyListener on video view
video.setOnKeyListener(new MultimediaOnKeyListener());
```

Prozor sa opcijama za reprodukovanje je realizovan kao modifikovan objekat klase *Dialog*. Njegov izgled implementiran je u okviru *xml* datoteke, koji predstavlja prirodno okruženje za implementaciju grafičke sprege u okviru *Android* razvojnog okruženja.

Pomoću *RadioButton* elemenata, moguće je izabrati vrstu multimedijalnog sadržaja.

Nakon toga moguće je odabratiti izvor sa kog potiče multimedijalna datoteka. Odabirom neke od ponuđenih opcija, prozor sa opcijama se gasi i počinje reprodukovanje multimedijalnog sadržaja. U slučaju audio datoteke, ekran imati crnu pozadinu za vreme trajanja reprodukcije, a u slučaju video datoteke, na ekranu će se iscrtavati njen sadržaj.

Elemente klase *Button* koji predstavljaju moguće opcije za odabir izvora multimedijalne datoteke, treba povezati sa *onClickListener* spregom, kako bi se uspostvaila interakcija sa korisnikom.

```
//Attach onClickListener
buttonFromFileFromSDCard.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        //Set video path and start
        video.setVideoPath(localPath);
        video.start();

        //Show video path
        multimediaPath.setText(localPath);

        //Hide options dialog
        optionsDialog.cancel();
    }
});
```

3.4 Aplikativno radno okruženje *JNI* sloj

JNI sloj programske podrške predstavlja vezu modula koji se nalaze u aplikativnom sloju i modula koji se nalaze u sloju niskog nivoa.

Prilikom realizacije podsistema za reprodukciju audio i video sadržaja, treba imati u vidu arhitekturu ciljne platforme, i izmene koje je neophodno izvršiti u sloju niskog nivoa, kako bi se performanse i vreme odziva poboljšalo. Shodno tome *MediaPlayerService* modul u okviru *JNI* sloja treba da pretrpi najveće izmene.

Početak komunikacije između aplikativnog sloja i sloja niskog nivoa dešava se u trenutku kada *JAVA* aplikacija želi da započne reprodukciju multimedijalnog sadržaja pozivom *setVideoPath* metode nad objektom *VideoView.java* klase.

Komunikacija prema nižim slojevima unutar *VideoView* klase teče na sledeći način:

- *public void setVideoPath(String path)*
- *public void setVideoURI(Uri uri, Map<String, String> headers)*
- *private void openVideo()*

U okviru *openVideo()* metode vrši se instanciranje objekta *MediaPlayer.java* klase i njegova priprema za reprodukciju. Prosleđuje mu se referenca na grafičku ravan na kojoj multimedijalni sadržaj treba da bude prikazan, pozivom *setDisplay()* metode, kao i putanja do multimedijalne datoteke čiji sadržaj treba reprodukovati pozivom *setDataSource()* metode nad objektom *MediaPlayer.java* klase.

Unutar *setDataSource()* metode u okviru *MediaPlayer.java* klase podaci se prosleđuju ka *JNI* sloju putem *native _setDataSource()* metode, čija se realizacija nalazi u *android_media_MediaPlayer.cpp* datoteci. *_setDataSource()* metoda zavisna od platforme, zatim poziva statičku metodu *android_media_MediaPlayer_setDataSourceAndHeaders()* u kojoj se instancira *C++* objekat za reprodukciju multimedijalnog sadržaja i prosledjuje mu se podatak o putanji multimedijalne datoteke koju treba reprodukovati.

```
android_media_MediaPlayer_setDataSourceAndHeaders(env, thiz, path, NULL, NULL) {

    sp<MediaPlayer> mp = getMediaPlayer(env, thiz);
    mp->setDataSource(pathStr,headersVector.size() > 0? &headersVector : NULL);
}
```

C++ objekat za reprodukciju multimedijalnog sadržaja definisan je u *mediaplayer.cpp* datoteci. Unutar njegove *setDataSource()* metode, preko *IMediaDeathNotifier.cpp* sprege poziva se *MediaPlayerService*, koji se pokreće unutar *main_mediaserver.cpp* modula, koji na osnovu

putanje multimedijalne datoteke instancira neki od realizovanih modula za reprodukciju multimedijalnog sadržaja niskog nivoa. Zatim se putanja do multimedijalne datoteke prosleđuje *IMediaPlayer.cpp* modulu i vrši se njegova inijalizacija.

```
setDataSource() {
    status_t err = UNKNOWN_ERROR;
    const sp<IMediaPlayerService>& service(getMediaPlayerService());
    if (service != 0) {
        sp<IMediaPlayer> player(service->create(getpid(), this,
                                                    mAUDIO_SESSION_ID));
        if (NO_ERROR != player->setDataSource(source)) {
            player.clear();
        }
        err = attachNewPlayer(player);
    }

    //Init player
    attachNewPlayer();

    return err;
}
```

Nakon toga metoda *status_t setDataSource(const char* url, const KeyedVector<String8, String8*> headers)*, koja je definisana u *IMediaPlayer.cpp* modulu, prosleđuje putanju i zaglavlje multimedijalne datoteke preko *IBinder* i *IMediaPlayerService* sprege *MediaPlayerService* modulu, gde se vrši određivanje odgovarajućeg modula za reprodukciju.

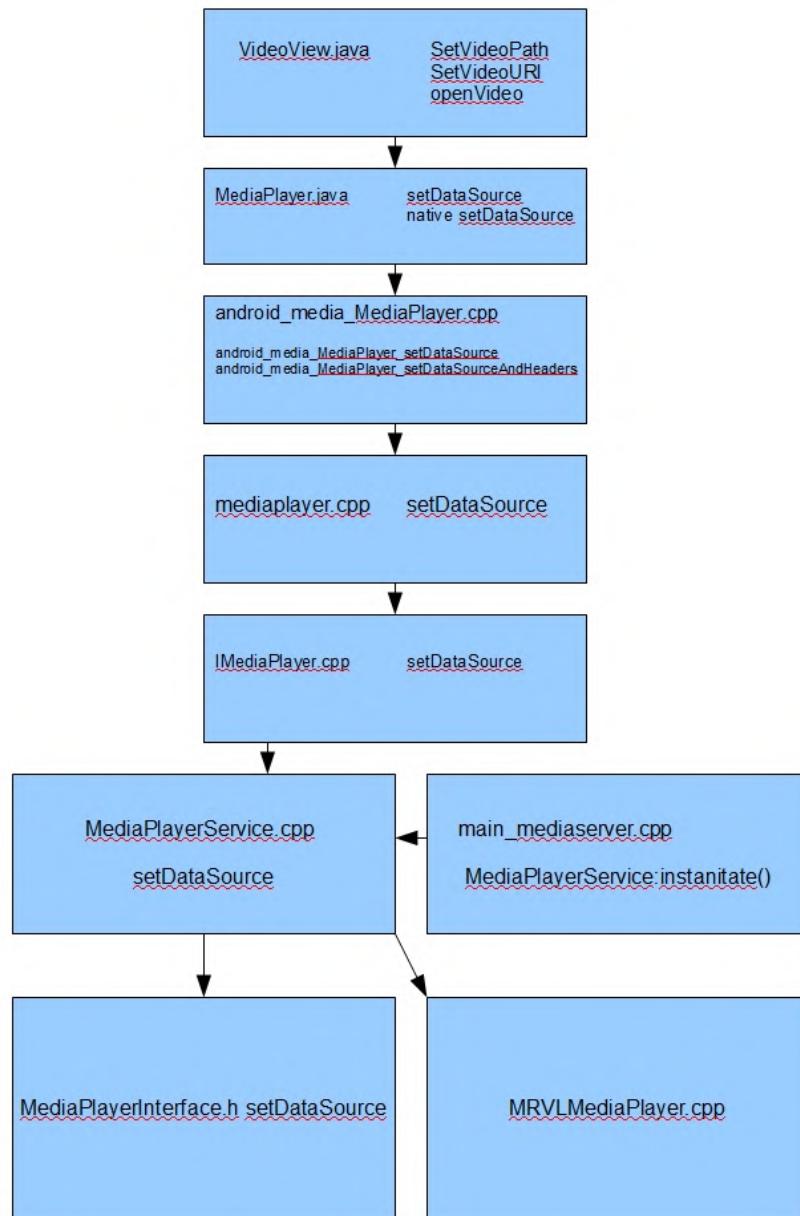
MediaPlayerService.cpp modul, kao što je već ranije rečeno, ispituje na osnovu putanje multimedijalne datoteke, koji od postojećih modula za reprodukciju multimedijalnog sadržaja najviše odgovara, zatim ga instancira, i preko *MediaPlayerInterface.h* modula prosleđuje neophodne podatke najnižem sloju programske podrške, gde se korišćenjem fizičkih modula za reprodukciju audio, odnosno video podataka, prikazuju na ekranu ciljne platforme.

U cilju poboljšanja performansi i vremena odziva podsistema za reprodukciju multimedijalnog sadržaja na *Android* baziranom prijemniku digitalnog televizijskog signala, realizovan je modul niskog nivoa za reprodukciju video i audio sadržaja, koji vodi računa o specifičnim detaljima ciljne platforme.

Iz tog razloga *MediaPlayerService* modul je izmenjen, tako što je izbačen deo za ispitivanje formata multimedijalne datoteke, čime je značajno dobijeno na brzini odziva. Umesto da se prilikom svakog iniciranja reprodukcije ispituje koji od modula najviše odgovara zadatom

formatu, *MediaPlayerService* modul programskoj podršci uvek prosleđuje *MRVLMediaPlayer.cpp* modul, koji razvijen na osnovnu specifičnih detalja ciljne platforme.

Na sledećoj slici prikazana je arhitektura programske podrške za reprodukciju multimedijalnog sadržaja, koja je razijena za ciljnu platformu, i redosled poziva funkcija od aplikativnog do nižih slojeva programske podrške opisan u prethodnom pasusu.



Slika 3.2 Arhitektura programske podrške i redosled poziva metoda

3.5 Sloj niskog nivoa

U okviru sloja niskog nivoa nalazi se realizacija modula za reprodukciju multimedijalnog sadržaja, koji je razvijan shodno detaljima ciljne platforme podistema za reprodukciju.

Realizacija modula za reprodukciju niskog nivoa, koji se u najvećoj meri koristi u ovako realizovanom podsistemu, nalazi se unutar *MRVLMediaPlayer.cpp* modulu.

MRVLMediaPlayer modul u sebi objedinjuje više manjih modula za reprodukciju, koji su nameski razvijeni za neke od specifičnih formata multimedijalog sadržaja.

Elementi koji su objedinjeni *MRVLMediaPlayer* modulom su:

- *MRLV_GSTPlayer*
- *MRVL_FFPlayer*
- *MRVL_DIVXPlayer*
- *MRVL_MCPlayer*
- *MRVL_DummyPlayer*
- *TS_Player*
- *CorePlayer*

MRVLMediaPlayer modul je usko povezan sa različitim bibliotekama koje poseduju mogućnost rukovanja različitim algoritmima i metodama za kodovanje i dekodovanje multimedijalnog sadržaja, između ostalog i sa *FFMPEG* bibliotekom, pa samim tim pruža široki opseg podržanih multimedijalnih formata što u velikoj meri povećava kvalitet programske podrške podistema za reprodukciju, u odnosu na originalnu ugrađenu programsku podršku.

FFMPEG biblioteka je besplatan programski paket, koji sadrži podršku za rukovanje multimedijalnim tokovima podataka [11]. Sastoji se od više modula, koji imaju svoju specifičnu namenu. Najvažniji moduli, koji su iskorišćeni prilikom realizacije podistema za reprodukciju multimedije su:

- *ffprobe* - koji je zadužen za prikupljanje informacija o toku multimedijalnih podataka
- *libavcodec* - sadrži podršku za veliki broj algoritama za kodovanje i dekovovanje multimedijalnog sadržaja. Algoritmi su definisani kao veoma brzi i veoma dobrog kvaliteta.

Bitan detalj za programsko rešenje podistema za reprodukciju je taj da se *Android* grafičko okruženje sastoji iz više grafičkih ravni, od kojih najniži prioritet ima grafička ravan na kojoj se iscrta video sadržaj. Pošto *MRVLMediaPlayer* poseduje mogućnost da istovremeno

instancira dva podmodula za reprodukciju, iskorišten je sledeći pristup za rešenje ovog problema.

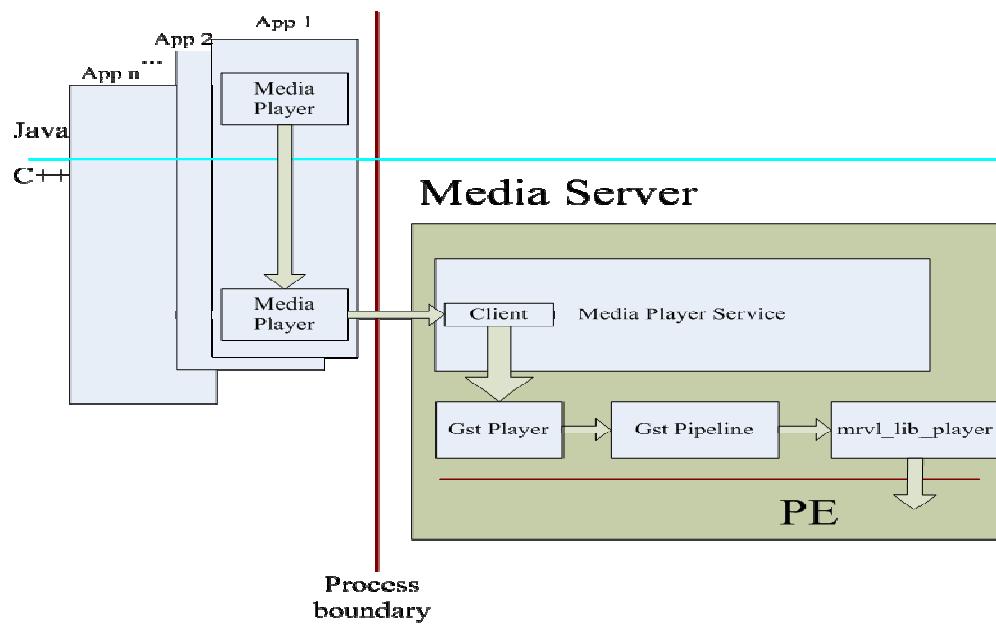
Unutar `setDataSource()` metode u *MRVLMediaPlayer* modulu instancira se *MRVL_DummyPlayer* modul, koji nema dodirnih tačaka za reprodukcijom zadatog multimedijanog sadržaja. Njegova uloga je da postavi nivo transparencije grafičkog sloja, koji je po prioritetu iznad grafičkog sloja na kom se iscrtava video sadržaj, na maksimum. Na taj način je obezbeđena vidljivost grafičkog sloja sa video sadržajem unutar *JAVA* aplikacije za ispitivanje i verifikaciju[6].

Nakon instanciranja *MRLV_DummyPlayer* modula i njegove inicijalizacije, potrebno je instancirati jedan podmodul, koji će reprodukovati zadati multimedijalni sadržaj.

Određivanje koji je podmodul najpogodniji za tu svrhu, vrši se uz pomoć *FFMPEG* biblioteke. Unutar *ffprobe* modula vrši se detaljno ispitivanje toka multimedijalnih podataka, određuje se format toka podataka, parsiraju se *PID* tabele, različiti dodatni podaci i sl.

Nakon što ovaj modul sakupi dovoljno informacija o toku podataka, *MRVLMediaPlayer* instancira neki od ranije navedenih modula za reprodukciju. Treba napomenuti da je vreme ove procedure prilikom realizacije dodatno smanjeno, čime se još više dobija na brzini odziva pod sistema za reprodukciju multimedije.

Na sledećoj slici prikazan je način na koji *JAVA* aplikacija komunicira sa *MRVLMediaPlayer* modulom za reprodukciju, koji je prilikom realizacije ovog pod sistema za digitalnu televizijsku platformu, zamenio ostale ugrađene module za reprodukciju, koji su opisani u ranijim poglavljima.

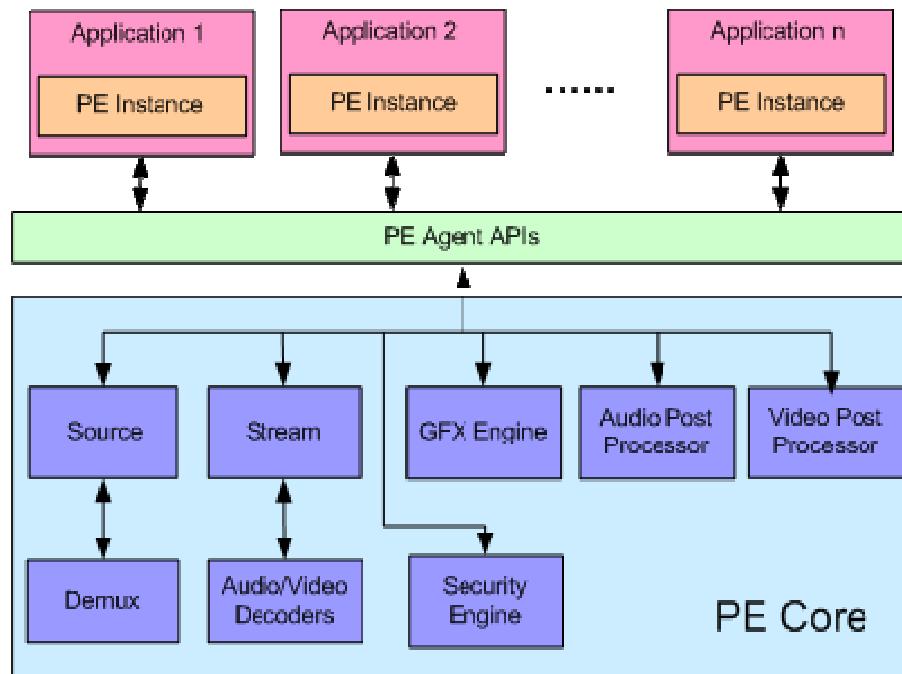


Slika 3.3 Komunikacija *JAVA* aplikacije i modula za reprodukciju niskog nivoa

Sa slike 3.4 se može uočiti da modul za reprodukciju niskog nivoa vrši komunikaciju sa podsistom za reprodukciju niskog nivoa (eng. *Presentation Engine*).

Presentation Engine modul obezbeđuje reprodukovanje multimedijalnog toka podataka, komunicirajući sa hardverskim modulima. Obezbeđuje dekodovanje i obradu audio podataka, dekovanje i obradu video podataka velike rezolucije, simultano reprodukovanje više od jednog toka video podataka, usmeravanje toka podataka, kontrolu grafičkog procesora, itd.

Sledeća slika prikazuje arhitekturu *Presentation Engine* modula.



Slika 3.4 Arhitektura *Presentation Engine* modula

Presentation Engine modul predstavlja najniži nivo programske podrške podistema za reprodukciju multimedijalnog sadržaja i samim tim kvalitet njegovog rada u velikoj meri utiče na ispravnost čitavog podistema.

4. Ispitivanje i verifikacija

U ovom poglavlju je dat je opis načina ispitivanja i verifikacije programskog rešenja, kao i rezultata koji su prikupljeni prilikom demonstracije funkcionalnosti.

Nakon završetka implementacije podsistema za reprodukciju multimedijalnog sadržaja potrebno je izvršiti njegovo ispitivanje, verifikaciju i demonstraciju funkcionalnosti.

Prilikom ispitivanja kvaliteta programske podrške poredene su performance originalne programske podrške i programske podrške prilagođene ciljnoj platformi.

Prilikom ispitivanja korištene su ispitne sekвенце video formata, jer se na taj način iskorištava veća količina memorijskih i procesorskih resursa, u odnosu na audio sadržaj.

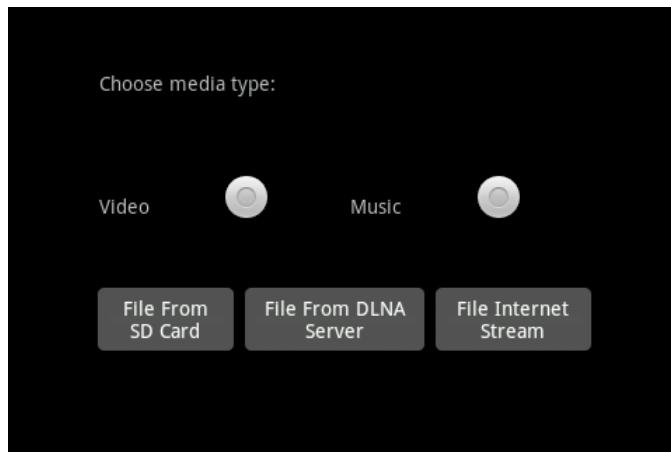
Programska podrška prilagođena ciljnoj platformi prikazala je mnogo brži odziv prilikom reprodukcije audio i video sadržaja, što rezultuje da krajnji korisnik mnogo bolje doživljava programsko rešenje (eng. *user experience*).

Takođe primetno je da podsistem podržava neke od formata multimedijalnog sadržaja, koje do sada nije podržavao, i da je njihova reprodukcija na zavidnom nivou, npr. multimedijalne datoteke *.TS* formata (*transport stream*) visoke rezolucije, koji predstavlja veoma zastupljen format video podataka u digitalnim televizijskim prijemnicima.

Na sledećim slikama prikazan je izgled *Android* aplikacije za ispitivanje i verifikovanje funkcionalnosti podsistema za reprodukciju multimedijalnog sadržaja.

Slika 4.1. prikazuje izleg prozora sa opcijama za reprodukciju.

Slika 4.2. prikazuje aplikaciju u trenutku reprodukovanja multimedijalne datoteka sa masovne memorije.



Slika 4.1 Prozor sa opcijama za reprodukciju



Slika 4.2 Reprodukcija multimedijalnog sadržaja sa masovne memorije

Sledeća slika prikazuje aplikaciju u trenutku reprodukovanja multimedijalne datoteke sa posluživača u okviru kućne mreže.



Slika 4.3 Reprodukcija multimedijalnog sadržaja sa posluživača unutar kućne mreže

Na sledećoj slici prikazana je aplikacija u trenutku reprodukovanja multimedijalne datoteke sa posluživača iz globalne mreže.



Slika 4.4 Reprodukcija multimedijalnog sadržaja sa posluživača u globalnoj mreži

Slika 4.5 prikazuje izgled kontrole za reprodukciju multimedijalnog sadržaja.



Slika 4.5 Izgled kontrole za reprodukciju multimedije

5. Zaključak

U ovom radu je prikazana realizacija podsistema i programske podrške za reprodukciju audio i video sadržaja na *Android* baziranim prijemnicima digitalnog televizijskog signala.

Prilikom realizacije korišćena je *Marvell BG2 SOC* platforma.

Nakon ispitivanja i verifikacije možemo primetiti mnogo bolju funkcionalnost i performansu podsistema za reprodukciju, kao i širi opseg podržanih multimedijalnih formata, u odnosu na originalni podsistem za reprodukciju multimedije i njegovu programsku podršku, koju *Android* kao operativni sistem nudi svojim korisnicima.

Prilikom razvijanja programske podrške vođeno je računa o detaljima specifičnim za ciljnu platformu, ali isto tako je bilo potrebno izvršiti što manje sistemskih izmena unutar *Android* operativnog sistema kako bi se zadržala mogućnost podrške širokog spektra *Android* baziranih digitalnih televizijskih platformi.

Imajući u vidu male izmene u logici i funkcionalnosti sistemskih programa unutar nižih slojeva operativnog sistema, smatra se da je ovaj zahtev u velikoj meri ispunjen.

Realizacijom ovog prodsistema i demonstracijom njegovih mogućnosti, uočava se i postojanje prostora za dalji napredak, kao što je to obično slučaj u svakom programskom rešenju.

Jedan od prostora za napredak je svakako količina redundantnih podataka prilikom komunikacije između slojeva programske podrške. Smanjenjem redundantcije dodatno bi se dobilo na performansi i boljem vremenu odziva čitavog podsistema. Takođe moguće je unaprediti algoritme dekodovanja i ispitivanja formata multimedijalnih tokova podataka i dodati podršku za neke od protokola za ramenu podataka (*SIP, VoIP, FTP, BitTorrent* i sl.).

Imajući u vidu da su socijalne mreže i multimedijalni servisi na globalnoj mreži sve popularniji. Takođe treba uzeti u obzir dodavanje podške za različite multimedijalne servise, kao i podršku za interakciju sa socijalnim mrežama.

Zaključno, demonstracijom funkcionalnosti ovakvog podsistema za reprodukciju multimedijalnog sadržaja uočavaju se mogućnosti koje nudi *Android* platforma u pogledu multimedijalne podrške i razvoja multimedijalnih aplikacija, što predstavlja važnu osobinu svih modernih digitalnih televizijskih prijemnika.

Daljim razvijanjem tehnologije, multimedijalna podrška digitalnih televizijskih prijemnika predstavljaće još veći značaj za potrošačko tržište, a zaključuje se da *Android* platforma pruža velike mogućnosti u pogledu razvoja najrazličitijih multimedijalnih aplikacija i interakcije sa krajnjim korisnikom, što predstavlja veoma svetlu budućnost.

6. Literatura

- [1] Wej-Meng Lee, *Android™ 4 Application Development*, Wiley / Sybex, Indianapolis, Indiana, 2012
- [2] Donn Felker: *Android Application Development For Dummies*, Willey Publishing, Inc., Indianapolis, Indiana, 2011
- [3] Reto Meier: *Professional Android™ Application Development*, Wiley Publishing, Inc., Indianapolis, Indiana, 2009
- [4] Allegro Software Development Corporation: *Networked Digital Media Standards, A UPnP / DLNA Overview*, Allegro Software Development Corporation, 2006
- [5] Milan Vidakovic, Nikola Teslic, Tomislav Maruna, Velibor Mihic: *Android4TV: a Proposition for Integration of DTV in Android Devices*, ICCE 2012, Las Vegas, 2012
- [6] Milan Vidaković, Nikola Crvenković: *Android4TV software design*, RT-RK Computer Based Systems, Novi Sad, 2012
- [7] T.Maruna, D.Kličković: *Network Media Player – Filelib Integration guide*, RT-RK Computer Based Systems, Novi Sad, 2010
- [8] W. Fischer, *Digital Video and Audio Broadcasting Technology, A Practical Engineering Guide*, Third Edition
- [9] Vladimir Kovačević, Miroslav Popović: *Sistemska programska podrška u realnom vremenu*, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2002
- [10] Android Developers, official Google Android documentation, developers.google.com
- [11] Wikipedia, the free encyclopedia, www.wikipedia.com
- [12] Marvell, *88DE3010 Software API User Manual*, dokumentacija ciljne platforme