УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Стеван Медић

Имплементација РВУ корисничке програмске подршке за Андроид платформу

МАСТЕР РАД

Нови Сад, 2014.

	УНИВЕРЗИТЕТ У НОВОМ САДУ ●**ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА МАСТЕР РАД	Датум:

(Податке уноси предметни наставник - ментор)

СТУДИЈСКИ ПРОГРАМ:	Рачунарство и аутоматика
РУКОВОДИЛАЦ СТУДИЈСКОГ ПРОГРАМА:	Никола Јорговановић

Студент:	Стеван Медић	Број	Е2 30/2013
Област:	Архитектуре и алгоритми ДСП 1		
Ментор:	др Јелена Ковачевић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ,ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;

НАСЛОВ МАСТЕР РАДА:

Имплементација РВУ корисничке програмске подршке за Андроид платформу

ТЕКСТ ЗАДАТКА:

Реализовати РВУ корисничку програмску подршку за Андроид платформу која омогућава изградњу функционалне графичке корисничке спреге праћене звучним ефектима, као и репродукцију мултимедијалног садржаја смештеног на удаљеном ДЛНА послужиоцу.

Руководилац студијског програма:	Ментор рада:

Примерак за: o - Студента;o -Ментора

УНИВЕРЗИТЕТ У НОВОМ САДУ ●**ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Дипломски – мастер рад	
Аутор, **АУ**:		**Стеван Медић**	
Ментор, **МН**:		**др Јелена Ковачевић**	
Наслов рада, **НР**:		**Имплементација РВУ корисничке програмске подршке за Андроид платформу**	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		**2014.**	
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)		**5/87/0/11/22/0/0**	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кqучне речи, **ПО**:		**РВУ, РУИ, Андроид, ДЛНА, УПнП, ЈНИ**	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		**Рад представља имплементацију РВУ корисничке програмске подршке за Андроид платформу, која омогућава изградњу функционалне графичке корисничке спреге праћене звучним ефектима, која се налази на удаљеном РВУ послужиоцу. Поред тога, решење омогућава репродукцију мултимедијалног садржаја смештеног на удаљеном ДЛНА послужиоцу.**	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	**др Никола Теслић**	
	Члан:	**др Растислав Струхарик**	Потпис ментора
	Члан, ментор:	**др Јелена Ковачевић**	

UNIVERSITY OF NOVI SAD●**FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Master Thesis	
Author, **AU**:		**Stevan Medić**	
Mentor, **MN**:		**Jelena Kovačević, PhD**	
Title, **TI**:		**Implementation of RVU client for Android plaftorm**	
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**:		**2014.**	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		**5/87/0/11/22/0/0**	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S**/**KW**:		**RVU, RUI, Android, DLNA, UPnP, JNI**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**This paper presents an implementation of RVU client for Android platform, which enables building of functional graphical user interface followed by sound effects stored on a RVU server. Besides, solution provides reproduction of a multimedia content stored on a DLNA server.**	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**Nikola Teslic, PhD**	
	Member:	**Rastislav Struharik, PhD**	Menthor's sign
	Member, Mentor:	**Jelena Kovacevic, PhD**	

Zahvalnost

Zahvaljujem se svojim roditeljima na bezrezervnoj podršci u toku studija. Takođe, zahvaljujem se Goranu Miljkoviću i članovima mog AMUSE tima na izuzetnoj saradnji i pomoći u toku izrade ovog rada.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

RUI	***R**emote **U**ser **I**nterface*, Udaljena grafička korisnička sprega
UPnP	***U**niversal **P**luga**n**d **P**lay*, Skup mrežnih protokola koji omogućavaju povezivanje multimedijalnih uređaja u cilju razmene informacija
DLNA	***D**igital **L**iving **N**etwork **A**lliance,* Protokol za deljenje digitalnog sadržaja između multimedijalnih uređaja
DMP	***D**igital **M**edia **P**layer*, Modul za reprodukciju u DLNA mreži
DMS	***D**igital **M**edia **S**erver*, Poslužilac u DLNA mreži
DMR	***D**igital **M**edia **R**enderer*, Modul za renderovanje u DLNA mreži
DMC	***D**igital **M**edia **C**ontroller*, Modul za kontrolu u DLNA mreži
DMPr	***D**igital **M**edia **Pr**inter***,*** Modul za kontrolu štampača u DLNA mreži
API	**A**pplication **P**rogramming **I**nterface, Aplikativna programska sprega
JNI	***J**ava **N**ative **I**nterface*, Java izvorna programska sprega u okviru Android platforme
URL	***U**niform **R**esource **L**ocator*, Jedinstvena adresa resursa
IP	***I**nternet **P**rotocol*, Internet protokol
TCP	***T**ransmission **C**ontrol **P**rotocol*, Protokol kontrole toka
DTCP	***D**igital **T**ransmission **C**ontent **P**rotection*, Protokol za zaštitu prenosa podataka
HTTP	***H**ypertext **T**ransfer **P**rotocol*, Aplikativni protokol za komunikaciju
XML	***E**xtensive **M**arkup **L**anguage*, Proširivi metajezik za označavanje tekstualnih dokumenata
QoS	***Q**uality **o**f **S**ervice,* Kvalitet servisa

SOAP	**S**imple **O**bject **A**ccess **P**rotocol, Mrežni protokol namenjen razmeni struktuiranih informacija o internet servisima u obliku XML podataka
GENA	**G**eneral **E**vent **N**otification **A**rchitecture, Arhitektura koja realizuje mehanizam prenosa obaveštenja između HTTP resursa

1. Uvod

U ovom radu je predstavljena realizacija RVU korisničke programske podrške za Android platformu. RVU (pseudo-akronim) predstavlja protokol za povezivanje i deljenje multimedijalnog sadržaja u okviru „umrežene“ kuće[1]. Baziran je na korisnik-poslužilac arhitekturi i oslanja se na postojeće standarde, UPnP i DLNA, podržane od strane velikog broja uređaja.

U poslednjih par godina težnja ka konceptu „umrežene“ kuće sve je više izražena. Broj domaćinstava koja poseduju lokalnu kućnu mrežu naglo je porastao, što otvara mogućnost za povezivanje pametnih uređaja u okviru doma.

Uređaji u koje spadaju telefoni, tableti, laptop računari, televizori, povezani su i međusobno i sa internetom preko žičanih (engl. LAN – Local Area Network) i bežičnih (engl. WLAN – Wireless Local Area Network) mreža što im otvara širok spektar mogućnosti, a ističu se automatizacija učestalih operacija, povećana bezbednost, brži i lakši pristup multimedijalnim sadržajima, udaljena kontrola i nadzor pojedinih kategorija uređaja poput bele tehnike itd.

Koncept „pametne“kuće nameće potrebu za odabirom platforme koja bi omogućila brzu nadogradnju postojećih i uvođenje novih funkcionalnosti, kako za distributere servisa (engl. service providers), tako i za projektante programske podrške i razvojne timova. U tu svrhu, za potrebe rada iskorišćena je Android platforma, koja je u poslednjih par godina stekla ogromnu popularnost i pronašla svoj put do različitih tipova uređaja, poput telefona, tableta, digitalnih TV prijemnika, pa čak i ručnih satova.

Jedan od osnovnih ciljeva umrežene kuće, jeste deljenje multimedijalnog sadržaja u okviru samog domaćinstva što omogućava svim članovima da, u bilo kom delu doma, imaju pristup multimedijalnom sadržaju skladištenom na kućnom poslužiocu. Problem predstavlja veliki broj aplikacija koje se koriste za tu namenu. Aplikacije nemaju jedinstven izgled, ne

nude iste funkcionalnosti i zahtevaju dodatno prilagođavanje prilikom prelaska sa jedne na drugu.

Postoji veliki broj standarda koji propisuju osnovna načela deljenja i reprodukcije multimedijalnog sadržaja. Osnovni cilj standardizacije je uvođenje jedinstvenog koncepta koji bi bio realizovan na velikom broju multimedijalnih uređaja u cilju olakšanog povezivanja istih. Najvažniji standardi za deljenje multimedijalnog sadržaja su DLNA (engl. Digital Living Network Alliance), AirPlay, Samsung Link, Chromecast, Miracast itd.

U cilju omogućavanja što šire dostupnosti multimedijalnog sadržaja u okviru kuće i rešavanju problema neujednačenosti prikaza i funkcionalnosti, zamišljen je i predstavljen RVU protokol. Njegova osnovna prednost, ogleda se u uvođenju RVU-RUI protokola koji omogućava svim multimedijalnim uređajima u sklopu umrežene kuće, posedovanje jedinstvenog grafičkog okruženja. Na taj način, korisnik ima isti doživljaj korišćenja aplikacija na različitim tipovima uređaja. Pored toga, RVU obezbeđuje deljenje AV multimedijalnog sadržaja u okviru umrežene kuće, koristeći postojeće funkcionalnosti DLNA standarda.

Uspeh na tržištu potrošačke elektronike, baziran je na brzom i jednostavnom uvođenju novih funkcionalnosti i aplikacija koje obično prate komercijalni sadržaj, kao i nadogradnji postojećih. To predstavlja jedan od osnovnih principa RVU protokola, koji realizovane aplikacije drži na strani RVU poslužioca i prosleđuje ih svim RVU korisnicima u istom obliku, sa istim izgledom i funkcionalnostima. Svaka promena aplikacija, manifestuje se promenom aplikacije na svim korisničkim uređajima, bez potrebe za preuzimanjem nove, unapređene verzije sa marketa, što je standardna praksa u većini postojećih rešenja.

RVU uvodi koncept programske podrške tzv. „tankog“ korisnika, čiji zadaci su reprodukcija multimedijalnog sadržaja, prikazivanje grafičke korisničke sprege i reagovanje na korisničke akcije, dok se procesorski zahtevne operacije obavljaju na strani multimedijalnog RVU poslužioca. Ovaj pristup omogućuje postojanje znatno jednostavnije korisničke aplikacije, koja zauzima manje prostora na disku i troši dosta manje resursa uređaja što se manifestuje kroz smanjenu potrošnju električne energije (posebno značajno u slučaju prenosivih uređaja) i manje zagrevanje što produžava životni vek samog uređaja.

Rad se sastoji iz sedam poglavlja.

Prvo poglavlje sadrži osnovne podatke o radu i uvod u problematiku umrežene kuće, koja predstavlja ciljno okruženje RVU protokola.

Drugo poglavlje ovog rada opisuje teorijske osnove, neophodne za razumevanje koncepta realizovanog i prezentovanog rešenja. Akcenat je na kratkom opisu Android platforme, kao i postojećih standarda koje RVU koristi, dok su za stvaranje opšte slike o

mestu koje RVU zauzima u sferi RUI rešenja, nabrojana postojeća rešenja i problemi koja ona poseduju.

U trećem poglavlju, izložen je koncept rešenja koji obuhvata opis gradivnih blokova RVU korisničke programske sprege, kao i način na koji oni interaguju i grade funkcionalnu celinu.

Četvrto poglavlje opisuje programsko rešenje i detaljnije zalazi u principe funkcionisanja gradivnih blokova kroz izlaganje funkcija i bitnih komponenata svakog pojedinačnog bloka.

Peto poglavlje prikazuje rezultate rada aplikacija specijalno realizovanih za ispitivanje kompletne funkcionalnosti RVU korisničke programske podrške. Kroz ovu proceduru, potvrđena je korektnost realizovanog rešenja.

Šesto poglavlje sadrži kratak pregled urađenog i pravce u kojima bi tekli dalji razvoj i optimizacija rešenja.

U sedmom poglavlju naveden je spisak korišćene literature u procesu izrade ovog rada.

2. Teorijske osnove

U ovom poglavlju dat je opis tehnologija neophodnih za razumevanje rešenja izloženog u ovom radu.

Predstavljena je Android platforma, kao i standardi koje koristi RVU protokol, UPnP i DLNA. Prikazani su i problemi koji su doveli do razvoja RUI protokola i na kraju, osnovne informacije u vezi samog RVU protokola.

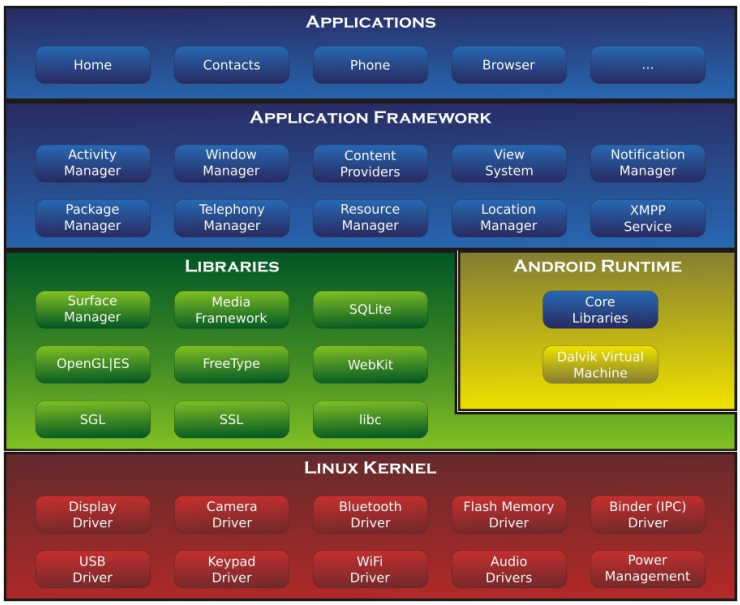
2.1 Android platforma

Android predstavlja platformu baziranu na Linuks jezgru, namenjenu prvenstveno prenosivim uređajima. Danas je prilagođena širokom spektru uređaja kao što su: mobilni telefoni, tablet računari, laptop računari, netbook računari, čitači elektronskih knjiga, igračke konzole, pa čak i ručni satovi.

Android se sastoji iz četiri osnovna sloja prikazana na slici 2.1:

- Linux jezgro (engl. Linux kernel) – omogućava komunikaciju na nivou fizičke arhitekture, upravljanje memorijom i procesima, logovanje, umrežavanje itd. Android koristi Linuksovo jezgro bazirano na Linuksovoj verziji jezgra sa dugoročnom podrškom (engl. Long Term Support (LTS)).
  Počevši od januara 2014. godine, Androidove verzije su bazirane na Linuks jezgru verzije 3.4 ili novijem.
  Androidovo Linuksovo jezgro poseduje izmene napravljene od strane Gugla (engl. Google), koje izlaze iz okvira razvojnog ciklusa standardnog Linuksovog jezgra. U te izmene spadaju postojanje bindera, ashmem, pmem, sistema za logovanje itd.
- Biblioteke i Androidova podrška u realnom vremenu (engl. Android runtime) – Biblioteke se koriste za razvoj aplikacija, a neke od biblioteka su: SQLite, WebKit, OpenGL ES, OpenSL ES, SSL, libc itd.
  Androidova podrška u realnom vremenu (engl. Android runtime), sadrži standardne Java biblioteke prilagođene Android platformi kao i Dalvik virtuelnu mašinu zaduženu za pokretanje aplikacija višeg nivoa napisanih u Java programskom jeziku.

- Razvojno okruženje (engl. Application framework) – predstavlja aplikativnu programsku spregu (eng. Application Programming Interface, API) napisanu u programskom jeziku Java, koja se koristi od strane Android aplikacija, za pristup nižem nivou arhitekture.
- Aplikacije – Android aplikacije koje su dostupne krajnjem korisniku, napisane u Java programskom jeziku i ukoliko je potrebno, delimično napisane u Java izvornoj programskoj sprezi (JNI).



Slika 2.1 Arhitektura Android platforme

Za razliku od ostalih operativnih sistema, kao što je na primer iOS koji pokreće iPhone i koji je pod potpunom kontrolom kompanije Apple, Android funkcioniše kao platforma otvorenog koda, tako da proizvođači mobilnih uređaja koji se odluče za ovu platformu mogu do izvesne mere da ga prilagode svojim potrebama. On omogućuje da se aplikacije izvršavaju podjednako dobro na svim mobilnim telefonima koji ga koriste kao platformu, nezavisno od proizvođača.

2.2 UPnP

UPnP (engl. Universal Plug-and-Play) [2] predstavlja set mrežnih protokola koji omogućuju mrežnim uređajima poput računara, štampača, WiFi tačaka pristupa i mobilnih uređaja, da se na jednostavan način prepoznaju i uspostave funkcionalne veze u cilju deljenja podataka, komunikacije i zabave. Namenjen je prvenstveno kućnim mrežama i predstavljen je od strane UPnP foruma koji predstavlja inicijativu računarske industrije da se razvije jedinstven princip komunikacije između velikog broja uređaja različitih proizvođača. Forum se sastoji od preko 800 proizvođača iz raznih sfera računarstva.

UPnP je distribuiran protokol, otvorene arhitekture, baziran na postojećim standardima poput TCP-IP, HTTP, XML, SOAP i sl.

U UPnP arhitekturi, postoje dve vrste uređaja:

- UPnP kontrolne tačke
- UPnP uređaji kontrolisani od strane UPnP kontrolnih tačaka

Osnovni koraci u toku umrežavanja su:

- Adresiranje – UPnP kompatabilni uređaji pri pristupanju mreži, dinamički dobijaju IP adresu
- Pronalaženje kompatabilnih uređaja – UPnP za pronalaženje uređaja koristi SSDP protokol (engl. Simple Service Discovery Protocol). Kada se UPnP kontrolisani uređaj pojavi na mreži, on preko SSDP poruka objavi kontrolnim tačkama svoje prisustvo i funkcionalnosti kroz implementirane servise.
- Opis – Nakon prepoznavanja odgovarajućeg servisa, kontrolna tačka i dalje ne zna dovoljno o kontrolisanom uređaju, pa zato dobavlja URL na kom se nalazi opis uređaja u XML formatu. Raščlanjivanjem tog dokumenta, kontrolna tačka saznaje informacije o uređaju, servisima i akcijama koje se mogu preduzeti.
- Kontrolisanje – U ovom koraku, uređaj je preko servisa sposoban da izazove određenu akciju na kontrolisanom uređaju. Poruke se šalju u XML formatu koristeći SOAP protokol (engl. Simple Object Access protocol).
- Informisanje o događajima – U slučaju promene argumenta akcije, kontrolisani uređaj preko GENA protokola (engl. General Event Notification Architecture) obaveštava kontrolnu tačku o novonastalom stanju.
- Prezentacija – Ukoliko kontrolisani uređaj sadrži prezentacioni URL, kontrolna tačka može da preuzme stranicu sa tog URL-a, učita je u internet pretraživaču i u zavisnosti od mogućnosti, kontroliše uređaj i ima uvid u stanje istog.

2.3 DLNA

DLNA (eng. Digital Living Network Alliance) [3] predstavlja neprofitnu organizaciju osnovanu od strane Sony-ja u Junu 2003. godine, čiji zadatak je definisanje osnovnih smernica za deljenje digitalnog multimedijalnog sadržaja između uređaja.

DLNA koristi UPnP protokol za upravljanje multimedijalnim sadržajem, kontrolisanje i pronalaženje kompatabilnih uređaja. UPnP definiše tipove uređaja koje DLNA koristi (renderer, poslužilac, upravljač). Prihvaćen je od strane vodećih proizvođača prenosnih i

stacionarnih multimedijalnih uređaja, kao što su Samsung, Sony, LG itd. Prema izvoru [4] broj prodatih DLNA sertifikovanih uređaja u naglom je porastu.

Slika 2.2 Primer DLNA arhtekture u sklopu umrežene kuće

DLNA kućni podržani uređaji se dele na sledeće klase:

- Digitalni media poslužilac (engl. Digital Media Server (DMS)): skladišti multimedijalni sadržaj i omogućava modulu za reprodukciju multimedijalnog sadržaja (DMP) i digitalnom media rendereru (DMR) da ga koriste.
- Modul za reprodukciju (engl. Digital Media Player (DMP)): pronalazi sadržaj na DMS uređaju i omogućava njegovu reprodukciju.
  U ove uređaje spadaju: kućni bioskop, bežični monitori, igračke konzole itd.
- Digitalni media renderer (engl. Digital Media Renderer (DMR)): reprodukuje sadržaj prema instrukciji DMC uređaja koji pronalazi sadržaj skladišten na DMS uređaju.
  U ove uređaje spadaju televizori, audio/video prijemnici, udaljeni zvučnici itd.
- Digitalni media kontroler (engl. Digital Media Controller (DMC)): pronalazi sadržaj skladišten na DMS uređaju i prosleđuje ga DMR uređaju
  U ove uređaje spadaju: tableti, WiFi kompatabilne digitalne kamere i pametni telefoni
- Digitalni media štampač (engl. Digital Media Printer (DMPr)): DMP i DMC uređaji mogu da koriste ovaj uređaj za štampanje.
  U ove uređaje spadaju mrežni štampači i mrežni sve-u-jednom (engl. all-in-one) uređaji.

2.4 RUI protokoli

RUI (engl. Remote User Interface) predstavlja koncept u kome se uređaj kontroliše sa udaljene lokacije. Kontroler prima odgovore na preduzete korisničke akcije što omogućava uvid u stanje kontrolisanog uređaja. Postoji više realizovanih RUI rešenja [5] koja se mogu grubo podeliti u dve osnovne kategorije [5] :

- Primitivni transfer grafičke korisničke sprege –predstavlja mehanizam za prikaz grafičke korisničke sprege na korisničkom uređaju. Poslužilac skladišti i određuje izgled grafičke korisničke sprege, dostavlja grafičke komponente i uklapa ih dok je korisnička programska podrška „tanka“ i jedini njen zadatak je iscrtavanje i reagovanje na akcije korisnika.

  Prednosti ovog pristupa su: isti doživljaj korišćenja grafičkog okruženja na različitim uređajima, jednostavnija programska podrška na korisničkoj strani, lakše održavanje i nadogradnja, smanjena potrošnja energije i prostora na disku...

  Mane ovog pristupa su: opterećenost mreže, kašnjenja u toku dostavljanja grafičkih komponenti, prilagođavanje različitim rezolucijama...

  Postoji više primitivnih RUI rešenja [5,6], a najpoznatija su: VNC (engl. Virtual Network Computing) [5,7], RDP (engl. Remote Desktop Protocol)[5,8], RFB (engl. Remote Frame Buffer) [5,9], X11, dok je RVU-RUI predstavljen u ovom radu.
- Označeni protokoli – Za razliku od prethodne kategorije, ova kategorija ima sasvim drugačiji pristup problemu udaljene grafičke korisničke sprege. Ovi protokoli ne opslužuju korisničke programske podrške grafičkim komponentama, već omogućuju globalni opis i radni okvir grafičke sprege. Kroz radni okvir, protokoli opisuju koje oblike grafičke korisničke sprege bi korisnička programska sprega trebala da sadrži i kako bi oni trebali da se ponašaju. Svi grafički elementi su skladišteni na strani korisnika i na taj način su rešeni problemi sa opterećenjem mreže i kašnjenjima u isporuci elemenata.

  Sa druge strane, korisnička programska podrška je dosta kompleksnija, procesorski i prostorno zahtevnija i ne nudi jedinstven doživljaj u toku korišćenja (iako sadrži iste funkcionalnosti na različitim uređajima).

  Opis grafičke korisničke sprege se dostavlja preko HTML (engl. Hyper Text Markup Language), XML (engl. eXtensible Markup Language) ili SVG (engl. Scalable Vector Graphics).

  U ovu kategoriju spadaju WIDEX i UPnP RUI protokoli.

2.5 RVU

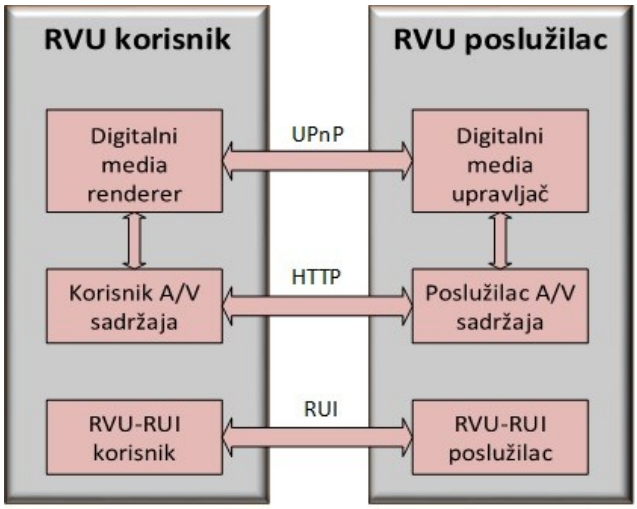
RVU (pseudo-akronim, izgovara se ”ar-vju”) [10] predstavlja protokol koji obezbeđuje ujednačene grafičke korisničke sprege i deljenje i reprodukciju multimedijalnog sadržaja na uređajima različitih proizvođača u domenu umrežene kuće. Pored toga, ubrzava uvođenje novih funkcionalnosti i programske podrške u postojeća rešenja koja prate komercijalni sadržaj.

RVU definiše i pravila distribucije A/V sadržaja ka korisničkim uređajima, korišćenjem DLNA standarda.

RVU-RUI predstavlja potprotokol RVU-a i njegovu glavnu karakteristiku koja ima sledeće prednosti :

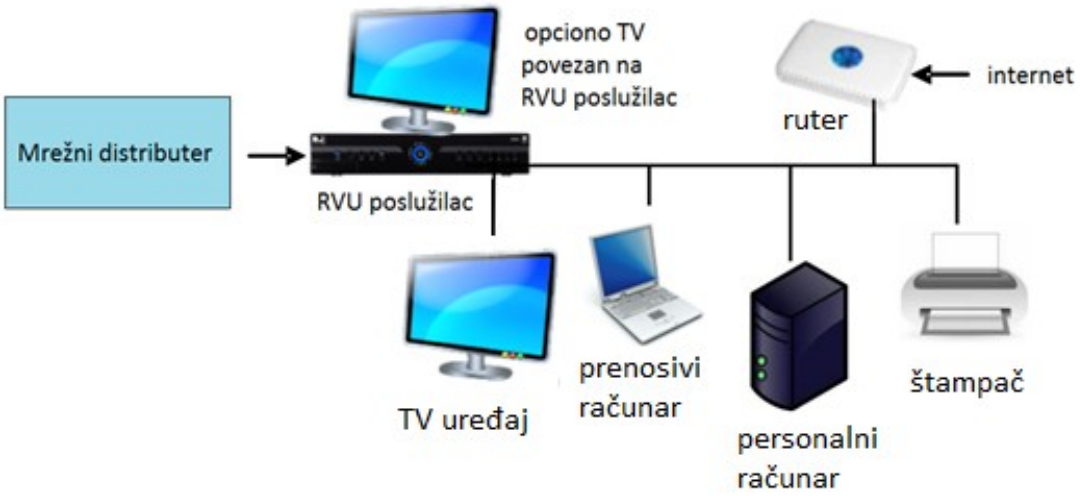
- RVU-RUI omogućava „tanku” (engl. thin) korisničku programsku podršku jer se sve grafičke komponente čuvaju na udaljenom RVU-RUI poslužiocu
- Poput ostalih primitivnih protokola, obezbeđuje ujednačenu grafičku korisničku spregu na svim korisničkim uređajima koji ostavljaju isti doživljaj korišćenja na korisnika.
- Optimizuje opterećenje mreže razdvajanjem logike za prenos grafičkih elemenata i AV sadržaja
- Grafičke komponente se mogu čuvati u skladišnim baferima i spajati na korisničkoj strani pod kontrolom RVU-RUI poslužioca
- Uvodi dodatne funkcionalnosti koje se ogledaju u ugrađivanju podrške za prikaz i kontrolu direktnog i odloženog televizijskog prenosa visokog kvaliteta, kao i ostalih oblika multimedijalnog sadržaja
- Uvodi zvučne efekte koji poboljšavaju opšti utisak u toku korišćenja aplikacije
- Uvodi napredne funkcionalnosti grafičke korisničke sprege, kao što je 3D

Logika dobavljanja grafičkih komponenata odvojena je od logike prenosa multimedijalnog sadržaja, a konačan izgled grafičke korisničke sprege, dobija se spajanjem grafičke ravni i jedne ili više video ravni.



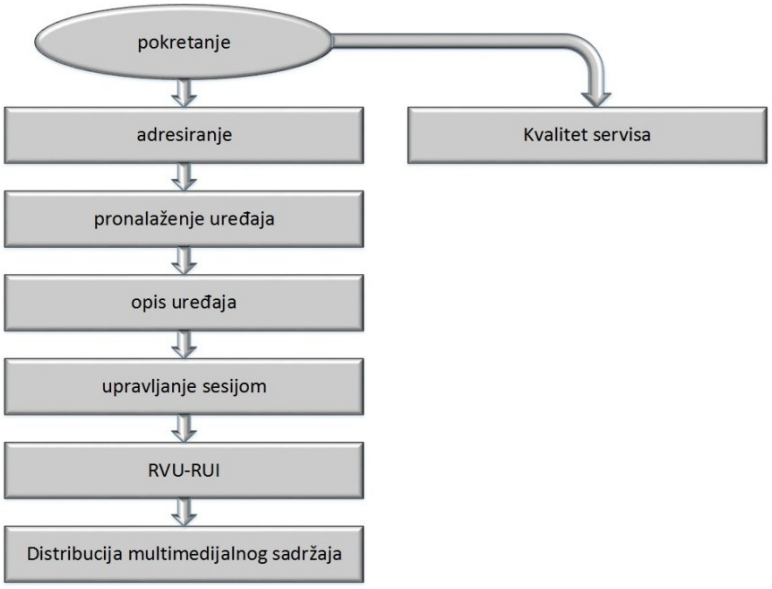
Slika 2.3 Arhitektura RVU protokola

Efikasno opsluživanje više korisnika istovremeno omogućeno je zahvaljujući mogućnosti svakog od korisnika da uspostavi više kanala sa poslužiocem. Kanali se mogu podeliti na komandne i kanale za podatke, što obezbeđuje dvosmernu komunikaciju sa minimalnim kašnjenjima u toku korišćenja programske podrške.



Slika 2.4 Primer RVU arhitekture u sklopu umrežene kuće

Da bi se korisnik i poslužilac povezali i bili u mogućnosti da razmenjuju podatke, neophodno je uspešno izvršiti niz koraka koji su navedeni na slici 2.5. Svaki od ovih koraka, predstavlja protokol označen kao RVU potprotokol.



Slika 2.5 Koraci u uspostavljanju veze između korisnika i poslužioca

2.5.1 Adresiranje, otkrivanje i opis uređaja

Adresiranje, otkrivanje i opis uređaja predstavljaju početne korake u procesu uspostavljanja veze i prepoznavanja drugih uređaja i njihovih sposobnosti. Navedeni koraci su bazirani na UPnP protokolu i označeni su kao RVU potprotokoli.

Osnovni zadaci pojedinačnih koraka su:

- Adresiranje – prijavljivanje na mrežu i dobavljanje mrežnih adresa
- Otkrivanje – RVU korisnik pronalazi kompatabilne uređaje i uspostavlja inicijalnu vezu sa njima
- Opis uređaja – RVU korisnik dobavlja XML dokument koji opisuje pronađeni RVU server i njegove sposobnosti tj. podržane servise

2.5.2 Upravljanje sesijom

Nakon što RVU korisnik dobavi opis pronađenih poslužioca koji sadrže RUI servis, neophodno je izdvojiti one koji imaju RVU-RUI podršku. Ukoliko RVU poslužilac podržava RVU-RUI, RVU korisnik dobavlja prolaz (engl. port) na kome će da u sledećem koraku otvori komandni kanal i razmeni inicijalne poruke.

2.5.3 RVU-RUI

RVU-RUI predstavlja jedinstvenu karakteristiku RVU protokola. Ovaj potprotokol je realizovan specijalno za potrebe komunikacije RVU korisnika i RVU poslužioca. Obezbeđuje kompletnu grafičku korisničku spregu u okviru korisničke aplikacije, prepoznatljivog izgleda i doživljaja korišćenja, bez potrebe za realizacijom dodatnih aplikacija koje bi obezbedile neophodnu grafičku korisničku spregu. RVU-RUI upotpunjuje doživljaj korišćenja

omogućivši zvučne efekte u toku korišćenja aplikacije. Pored toga, omogućava kontrolisanje reprodukcije AV sadržaja skladištenog u okviru RVU poslužioca.

2.5.4 Distribucija multimedijalnog sadržaja

Distribucija multimedijalnog sadržaja predstavlja RVU potprotokol koji opisuje mehanizam potreban za bezbedno opsluživanje RVU korisnika AV sadržajem (direktan ili odloženi televizijski prenos, kao i ostali oblici multimedijalnog sadržaja).

A/V sadržaj se distribuira shodno DLNA pravilima distribucije, koji za prenos koriste UPnP AV Media uređaje i HTTP protokol. Pored toga, distribucija multimedijalnog sadržaja definiše i osnovna načela distribucije sadržaja, izbor alternativnog izvora zvuka, kao i rukovanje neovlašćenim sadržajem.

Za obezbeđivanje zaštite prilikom isporučivanja multimedijalnog sadržaja, RVU koristi DTCP (engl.Digital Transmission Content Protection).

2.5.5 Kvalitet servisa i dijagnostika

Kvalitet servisa i dijagnostika predstavljaju niz standarda koji nastoje da održe visok nivo kvaliteta servisa u toku korišćenja RVU prokola u okviru umrežene kuće.

RVU kvalitet servisa je baziran na DLNAQoS modelu i koristi alate za dijagnostiku. Alati mogu biti bazirani na automatskom održavanju, kao i na korisničkom rešavanju problema.

2.5.6 Dobavljanje inicijalne slike

Dobavljanje inicijalne slike (engl. Client Image Acquisition (CIA)) predstavlja RVU potprotokol koji koristi TFTP protokol (engl. Trivial File Transfer Protocol). Osnovni cilj je da se RVU korisniku olakša informisanje o postojanju nove inicijalne slike, lokacije na kojoj se slika nalazi kao i načina na koji je tu sliku moguće dobaviti.

Na strani RVU korisnika, ovaj potprotokol nije obavezan i njegova realizacija nije predviđena za potrebe ovog rada.

3. Analiza problema i koncept rešenja

Problem umrežene kuće proizilazi iz činjenice da korisničke aplikacije kućnih uređaja poseduju neujednačene grafičke korisničke sprege (engl. GUI – Graphical User Interface). Kompletna grafička korisnička sprega se nalazi na samom uređaju u različitim verzijama i varijacijama u zavisnosti od realizacije proizvođača, što komplikuje korišćenje kućnih uređaja, a samim tim i umanjuje kvalitet i pouzdanost usluge.

Da bi se prevazišao problem neujednačenosti prikaza i funkcionalnosti, neophodno je obezbediti namensku aplikaciju za svaki od podržanih sistema. Ovakav pristup zahteva više vremena u procesu realizacije i održavanja, i samim tim, neophodno je obezbediti značajnija novčana sredstva za razvoj i održavanje višestrukih aplikacija. Koncept koji je definisan RVU protokolom, predviđa smeštanje kompletne grafičke korisničke sprege i funkcionalnosti na jedno mesto, na centralni poslužilac, čime se razvoj i održavanje višestruko pojednostavljuju.

RVU protokol realizuje koncept centralizovanog sistema tako što sadržaj smešta na RVU poslužioca, dok se na strani RVU korisnika nalazi „tanka“ (engl. thin) korisnička programska podrška. Omogućeno je postojanje jednostavnih aplikacija na korisničkoj strani, kao i njihovo lakše održavanje i nadogradnju. Svaka promena aplikacija, manifestuje se promenom aplikacije na svim korisničkim uređajima, bez potrebe za preuzimanjem nove, unapređene verzije sa marketa, što je standardna praksa u većini postojećih rešenja.

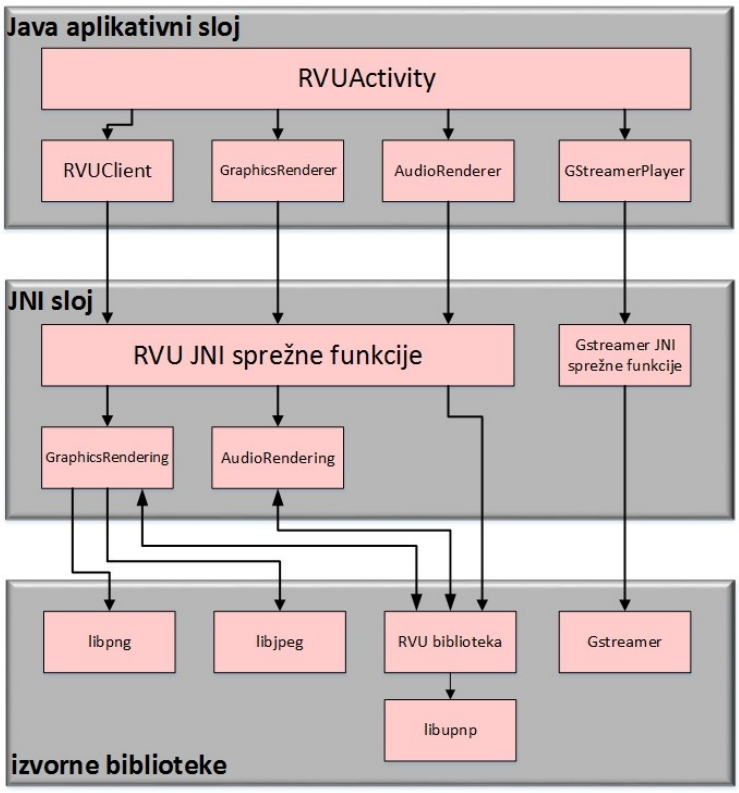
Zadaci RVU korisnika su:

- reprodukcija multimedijalnog sadržaja,
- prikazivanje grafičke korisničke sprege
- reprodukcija zvučnih efekata koji prate korišćenje grafičke korisničke sprege i
- reagovanje na korisničke akcije

RVU korisnička programska podrška smanjuje potrošnju energije jer se sve zahtevne operacije obavljaju na strani RVU poslužioca, što je posebno značajno kod prenosivih uređaja sa ograničenim trajanjem baterije. Poslužilac sadrži programsku podršku koja omogućava obavljanje operacija poput snimanja multimedijalnog sadržaja, direktnog prenosa jednog ili više televizijskih kanala, skladištenja i generisanja elemenata grafičke korisničke sprege koji se prikazuju na korisničkoj strani, uvođenja novih funkcionalnosti kroz proces nadogradnje i sl.

U nastavku teksta, opisana je RVU korisnička programska sprega, koja predstavlja slojevitu strukturu, sastavljenu od RVU biblioteke sa programskom aplikativnom spregom, srednjeg prilagodnog sloja i gornjeg Java aplikativnog sloja, kao što je prikazano na slici 3.1.

Poseban akcenat je stavljen na RVU korisničku biblioteku napisanu u C programskom jeziku. Ona sadrži osnovne funkcionalnosti rešenja, u koje spadaju komunikacija sa RVU poslužiocem, primanje grafičkih elemenata i zvučnih efekata, kao i prihvatanje komandi vezanih za reprodukciju AV sadržaja u okviru DMR komponente.



Slika 3.1 Struktura RVU korisničke programske podrške

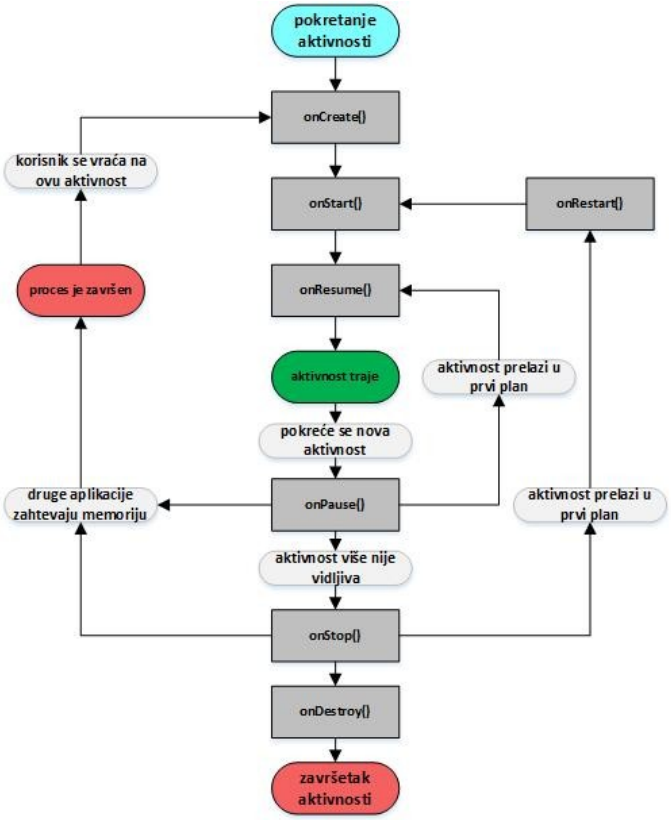
3.1 Java aplikativni sloj

Aplikativni sloj je napisan u osnovnom Androidovom razvojnom jeziku - Javi. Sastoji se od glavne aktivnosti – RVUActivity, kao i klasa koje apstrahuju funkcionalnost JNI sloja: GraphicsRenderer, AudioRenderer, RVUClient i GStreamerPlayer.

3.1.1 RVUActivity

RVUActivity predstavlja glavnu aktivnost programa čiji su osnovni zadaci:

- Rukovanje osnovnim fazama životnog veka aplikacije koje su prikazane na slici 3.2
- Inicijalizacija i deinicijalizacija RVU biblioteke
- Inicijalizacija audio komponenata koje u okviru JNI sloja, izvršavaju reprodukciju zvučnih efekata dostavljenih iz RVU biblioteke. To se obavlja preko klase AudioRenderer.
- Inicijalizacija grafičkih komponenata koje u okviru JNI sloja iscrtavaju grafičku korisničku spregu sastavljenu od grafičkih elemenata dostavljenih iz RVU biblioteke. To se obavlja preko klase GraphicsRenderer koja realizujeAndroidovu ugrađenu spregu (engl. interface) GLSurfaceView.Renderer.
- Inicijalizacija GStreamerPlayer komponenterealizovane u GStreamerPlayer klasi koja predstavlja korisnika kome se AV sadržaj isporučuje preko HTTP protokola
- Rukovanje GLSurfaceView komponentom po kojojOpenGL ES 2.0 vrši iscrtavanje
- Reagovanje na korisničke akcije kroz Listener Androidove mehanizme i prosleđivanje događaja RVU biblioteci preko metoda RVUClient klase i JNI sloja



Slika 3.2 Životni vek aplikacije

GLSurfaceView predstavlja SurfaceView komponentu, prilagođenu specijalno za potrebe OpenGL iscrtavanja.

Ima sledeće funkcionalnosti:

- Rukuje površinskim slojem, koji predstavlja deo memorije sa mogućnošću integracije u Androidov sistem prikaza
- Rukuje EGL prikazom, koji omogućava OpenGL-u da crta po površini
- Prihvata namenske rendererske objekte koji vrše fizičko iscrtavanje
- Iscrtava po specijalno namenjenoj niti, u cilju poboljšanja performansi
- Po potrebi, proverava uspešnost izvršavanja OpenGL poziva

Ukoliko je glavna aktivnost pauzirana ili nastavljena, neophodno je o tome obavestiti GLSurfaceView. To se radi preko metoda *glSurfaceView.onPause* i *glSurfaceView.onResume.* Na ovaj način se pauzira ili nastavlja crtanje u okviru niti renderera.

3.1.2 GStreamerPlayer

GStreamerPlayer predstavlja klasu koja realizuje HTTP korisnika čiji zadatak je da na zahtev digitalnog media upravljača (DLNA DMC), a preko digitalnog media renderera (DLNA DMR), započne primanje i prikaz AV sadržaja u okviru GStreamer [11] komponente. GStreamerPlayer koristi SurfaceView Androidovu komponentu za prikaz video zapisa.

Androidova standardna komponenta za reprodukciju AV multimedijalnog sadržaja – VideoView, ne podržava MPEG2, koji predstavlja standardni format za distribuciju sadržaja digitalne televizije. Iz tog razloga, VideoView je odbačen kao potencijalno rešenje, dok je za potrebe reprodukcije AV sadržaja iskorišćen Gstreamer.

GStreamer predstavlja prenosivu C biblioteku otvorenog koda, namenjenu za iscrtavanje i predstavljanje grafika, kao i rukovanje multimedijalnim sadržajem. Za omogućavanje korišćenja biblioteke u okviru Android platforme, realizovane su JNI sprežne funkcije, preko kojih Java aplikacioni sloj koristi funkcije C biblioteke.
JNI sprega je realizovana u okviru modula – GStreamer JNI sprežne funkcije.

3.1.3 GraphicsRenderer

Za iscrtavanje grafičke korisničke sprege realizovana je klasa GraphicsRenderer koja realizuje GLSurfaceView.Renderer spregu.

Grafička korisnička sprega se u obliku grafičkih elemenata dostavlja od strane RVU poslužioca do RVU korisnika i JNI sloja. Iscrtavanje se vrši u JNI sloju, zbog boljih performansi izvornih jezika (C/C++), kao i izbegavanja prosleđivanja grafičkih elemenata iz JNI sloja u Java sloj koje unosi dodatna kašnjenja. Objekat klase GraphicsRenderer je pridružen OpenGL površinskom sloju GLSurfaceView i na taj način pripremljen za crtanje.

3.1.4 AudioRenderer

AudioRenderer predstavlja klasu koja apstrahuje funkcionalnosti audio komponente JNI sloja, u cilju inicijalizacije i rukovanja istom.

Audio komponente tj. u našem slučaju, zvučne efekte, šalje RVU poslužilac, a prima RVU korisnik koji ih propagira do JNI sloja. Reprodukcija zvučnih efekata dobavljenih iz RVU biblioteke, izvršava se u JNI sloju, koristeći Androidovu ugrađenu biblioteku OpenSL ES 1.1.

3.1.5 RVUClient

RVUClient predstavlja klasu čiji osnovni zadaci su:

- inicijalizacija RVU biblioteke
- primanje instrukcija koje DLNA DMR uređaj prosleđuje ka višim slojevima i prosleđuje korisniku AV sadržaja – GStreamerPlayer-u. U instrukcije spadaju adresa sa koje je potrebno pokrenuti reprodukciju AV sadržaja smeštenog na DLNA DMS poslužiocu, akcija zaustavljanja reprodukcije, pauziranja, premotavanja unapred i unazad itd.
- sadrži metode za prosleđivanje događaja nastalih usled korisničke akcije

Korisnički događaji, u zavisnosti od uređaja na kome je pokrenuta aplikacija, mogu biti: dodir (engl. touch) i prevlačenje (engl. drag) (telefoni, tableti) ili pritisak na dugme (daljinski upravljač za TV ili digitalni TV prijemnik (engl. set-top box (STB)).

3.2 Prilagodni sloj - Java izvorna sprega

Java izvorna sprega (engl. Java Native Interface (JNI)) predstavlja programsko okruženje, koje omogućava Java sloju da iz Dalvik virtuelne masine, poziva izvorni kod, kao i da bude pozvan iz izvornog koda (C/C++, asembler).

RVU biblioteka je napisana u programskom jeziku C koji omogućava prenosivnost na različite platforme poput Androida, Linuksa ili iOS-a.

Da bi se iskoristila funkcionalnost biblioteke u okviru Android platforme koja je u fokusu ovog rada, neophodno je obezbediti vezu sa aplikativnim slojem u cilju prikaza grafičkih elemenata, zvučnih efekata i reagovanja na korisničke akcije.
Prikaz arhitekture JNI sloja prikazan je na slici 3.1.

3.2.1 RVU JNI sprežne funkcije

Ovaj modul sadrži funkcije koje apstrahuju funkcionalnosti modula GraphicsRendering i AudioRendering, kao i RVU biblioteke. Na ovaj način, biblioteke su vidljive iz Java aplikativnog sloja i moguće ih je koristiti.

3.2.2 Gstreamer JNI sprežne funkcije

Gstreamer JNI sadrži funkcije koje omogućavaju korišćenje GStreamer multimedijalne C biblioteke predstavljene u poglavlju 3.1.2. Gstreamer JNI funkcije se oslanjaju na GStreamer C biblioteku [11].

3.2.3 GraphicsRendering

GraphicsRendering predstavlja JNI modul za obradu i prikaz grafike uz pomoć Androidove ugrađene biblioteke za renderovanje grafike – OpenGL ES 2.0. Princip iscrtavanja pomoću OpenGL ES izvorne biblioteke, predstavljen je u [14].

Performanse RVU koncepta u velikoj meri zavise od kvaliteta i brzine mreže. Grafički elementi koji grade grafičku korisničku spregu, ne smeju da budu previše zahtevni sto se veličine i prostora na disku tiče, a opet određeni kvalitet grafičkih elemenata mora biti zadovoljen u cilju izgradnje što atraktivnije aplikacije. Iz tog razloga, potrebno je napraviti kompromis, u cilju ostvarivanja što većeg broja okvira, uz zadovoljavajući kvalitet grafičke korisničke sprege.

Slanje zahtevnih grafičkih elemenata poput sirovih podataka (engl. raw data), u slučaju velikih rezolucija, ozbiljno opterećuje mrežu i narušava doživljaj korišćenja same aplikacije. Zbog toga, realizovana RVU aplikacija koristi grafičke elemente u obliku kompresovanih podataka, kao što su jpeg i png formati. Grafičke elemente je neophodno dekompresovati uz pomoć libjpeg ili libpng biblioteka.

Nakon što su podaci dekompresovani, neophodno je napraviti openGL teksturu. Tekstura predstavlja kontejner za jednu ili više slika i njene osnovne osobine su: veličina, tip i format slika koje sadrži. Za potrebe RVU korisničke programske podrške, iskorišćena je 2D tekstura.

Tekstura se gradi od sirovih podataka i smešta u grafičku memoriju. Nakon toga, teksturi je moguće pristupiti uz pomoć jednoznačno određenog rukovaoca. Posto RVU poslužilac dostavlja veliki broj grafičkih elemenata u cilju izgradnje potpuno funkcionalne grafičke korisničke sprege, kreira se veliki broj tekstura koje je neophodno ukloniti iz memorije u cilju sprečavanja curenja i preopterećenja memorije. U tu svrhu, realizovana je lista koja čuva rukovalac teksturom, kome odgovaraju koordinate na kojima je iscrtana tekstura. Na ovaj način, ukoliko se tekstura više ne koristi, tj. nije više vidljiva jer se preko nje nalazi nova tekstura, ona se uklanja iz memorije i oslobađa se video memorija.

Korišćen je princip višestrukog baferovanja, čija je osnovna namena sakrivanje procesa renderovanja od korisnika. Na ovaj način, korisnik nema uvid u sam proces sastavljanja grafičke korisničke sprege, u kome može doći do pojave artifakta, nekompletnosti, treperenja i drugih deformiteta slike. Renderovanje se vrši u pozadinskom baferu koji nije vidljiv korisniku i kada se taj proces završi, pozadinski bafer se zamenjuje sa primarnim i prikazuje korisniku.

Prema specifikaciji RVU standarda, podržane operacije su: popunjavanje pravougaonika određenom bojom, Porter-Daf (engl. Porter-Duff) spajanje, promena dimenzija određenog regiona slike, iscrtavanje slike itd. Kompletan spisak grafičkih komandi, prikazan je u tabeli 3.7.

3.2.4 AudioRendering

AudioRendering predstavlja modul za reprodukciju zvučnih efekata koji poboljšavaju korisničko iskustva korišćenja aplikacije. Obavlja se preko Android NDK ugrađene biblioteke OpenSL ES 1.1.

Zvuk se u RVU biblioteci smesta u namenski bafer i transportuje do ovog modula, gde se reprodukuje u specijalno realizovanom OpenSL ES modulu za reprodukciju. Kontrolu reprodukcije obavlja RVU poslužilac, u zavisnosti od korisničke akcije.

Zvučni efekat je moguće sačuvati u privremenom baferu za ponovnu reprodukciju ukoliko to RVU poslužilac zahteva.

3.3 Izvorne biblioteke

Izvorne biblioteke predstavljaju celovite module čije funkcionalnosti se izlažu Java izvornom sloju (JNI) u cilju implementacije logike iscrtavanja grafičke korisničke sprege, reprodukcije zvučnih efekata, konfigurisanja Gstreamer modula za reprodukciju AV sadržaja, prihvatanja korisničkih događaja i reagovanja na iste itd.

Korišćene biblioteke predstavljene su na slici 3.1, a u njih spadaju RVU biblioteka, JPEG biblioteka (libjpeg), PNG biblioteka (libpng), UPNP biblioteka (libupnp) i GStreamer biblioteka.

3.3.1 Libpng i libjpeg

Libpng i libjpeg predstavljaju biblioteke za rad sa kompresovanim podacima koji reprezentuju grafičke elemente.

Za potrebe dekodovanja slike u jpeg ili png formatu, mogla se iskoristiti ugrađena klasa BitmapFactory i njena funkcija *decode()*. Pošto je RVU korisnička podrška predstavljena u ovom radu, prenosiva na različite platforme, koristeći eksterne biblioteke libpng i libjpeg obezbeđen je jedinstven način dekodovanja. U suprotnom, na svakoj platformi bi morao biti realizovan platformski-zavisan princip dekodovanja.

Libpng [12] predstavlja biblioteku za rad sa podacima u png formatu. Grafički elementi se preko mreže dopremaju do RVU biblioteke, koja ih dalje propagira do JNI sloja. U JNI sloju, podaci se dekompresuju uz pomoc *libpng* i poslužuju openGL-u u sirovom obliku.

Osnovne prednosti PNG formata su:

- Uvodi kompresiju bez gubitaka
- Postojanje alfa kanala koji omogućava prozirnost

Osnovne mane:

- Ne podržava animacije
- Problemi sa starijim pretraživačima (npr. Internet Explorer 6)

Na sličan način funkcioniše i libjpeg biblioteka [13], koja za razliku od libpng, koristi podatke u kompresovanom jpeg formatu. Za potrebe ovog rada, iskorišćena je za dekompresiju podataka iz jpeg komprimovanog oblika u sirovi oblik.

Osnovne prednosti jpeg formata su:

- Malo zauzeće prostora na disku usled otklanjanja elemenata koje ljudsko oko ne može da zapazi

- Sadrži obimnu paletu boja
- Pogodan za fotografije
- Poseduje bogat gradijent boja

Mane:

- Koristi kompresiju sa gubicima tako da se određeni detalji slika nepovratno gube
- Loš za prikaz logoa i linija

3.3.2 RVU biblioteka

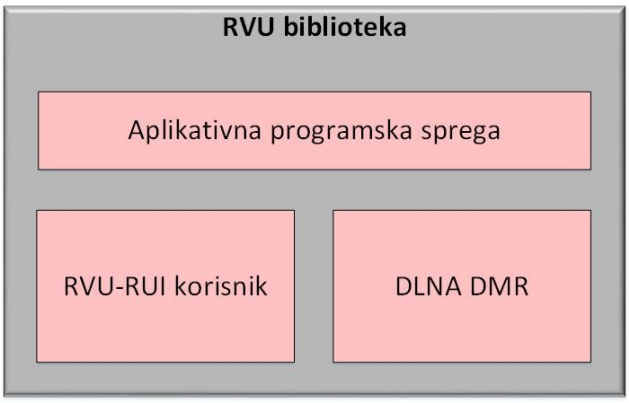
Cela logika RVU protokola, nalazi se u ovom sloju rešenja. RVU se oslanja na postojeće standarde i protokole poput UPnP, DLNA, TCP-IP itd.

RVU biblioteka ima zadatak da omogući višim slojevima korišćenje funkcionalnosti RVU protokola.

U osnovne funkcionalnosti spadaju:

- Razmena inicijalnih informacija sa RVU-RUI poslužiocem
- Konfiguracija osnovnih komponenata i parametara neophodnih za funkcionisanje protokola, poput zauzimanja bafera za grafičke i audio elemente
- Primanje grafičkih elemenata i zvučnih efekata
- Propagiranje koordinata korisničkih događaja ka RVU-RUI poslužiocu
- Realizacija DLNA DMR uređaja prema DLNA standardu, uključujući izmene predviđene RVU protokolom
- Primanje instrukcija u okviru DMR poslatih od strane DMC uređaja smeštenog na strani RVU poslužioca
- Izlaganje funkcionalnosti RVU-RUI korisnika i DLNA DMR uređaja JNI sloju
- Realizacija funkcija sa povratnim pozivom koji propagiraju događaje ka višim slojevima

Osnovni gradivni blokovi RVU korisnika prikazani suna slici 3.3.



Slika 3.3 Arhitektura RVU biblioteke

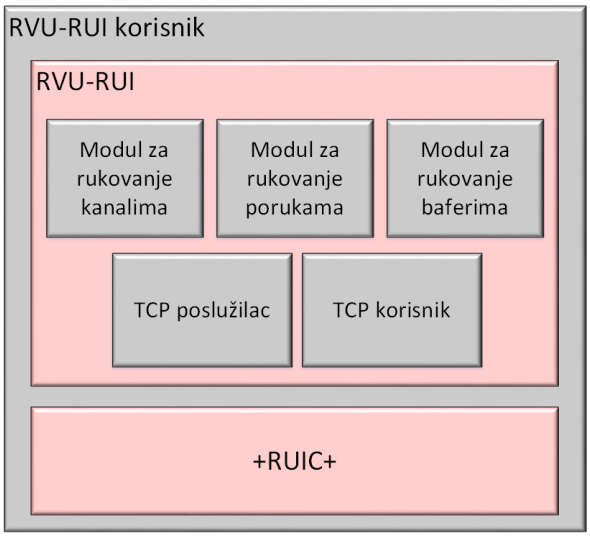
3.3.2.1 Aplikativna programska sprega

Aplikativna programska sprega RVU biblioteke predstavlja sloj koji izlaže funkcionalnost biblioteke kroz apstrakciju njenih funkcija i omogućava njeno olakšano korišćenje i integraciju u složenije sisteme.

3.3.2.2 RVU-RUI korisnik

RVU-RUI korisnik predstavlja najvažniju kariku u RVU arhitekturi, koja omogućava komunikaciju između RVU korisnika i udaljenog RVU poslužioca. Pored toga, omogućava poslužiocu da opslužuje korisnika sa grafičkim elementima i zvučnim efektima u cilju izgradnje potpuno funkcionalne grafičke korisničke sprege.

Realizovan je u skladu sa RVU specifikacijom 1.0.



Slika 3.4 Arhitektura RVU-RUI korisnika

RUIC modul sadrži implementaciju potprotokola opisanih u poglavlju 2.5. Registracija na mreži se obavlja korišćenjem potprotokola za adresiranje. Otkrivanje RVU poslužilaca se obavlja tako što korisnik prima ssdp:alive poruku ili šalje SSDP poruku na koju prima odgovor po principu UPnP otkrivanja (2.5.1). U fazi opisivanja, RUIC saznaje osnovne karakteristike poslužioca što uključuje osnovne informacije i podržane servise. Nakon toga, sledi upravljanje sesijom (2.5.2) u kome RUIC šalje *GetCompatibleUIs* poruku kojom se obraća RUI servisu. Ukoliko poslužilac sadrži RVU-RUI servis, on odgovara UI listom čijim parsiranjem RVU korisnik saznaje prolaz na kome će otvoriti komandni kanal i uspostaviti inicijalnu vezu sa poslužiocem.

TCP korisnik uz pomoć modula za rukovanje kanalima, otvara komandni kanal preko koga se obraća poslužiocu posredstvom XML poruka. Inicijalnu komunikaciju započinje isključivo RVU korisnik.

U prvoj poruci, koja je nazvana pozdravna (engl. Hello) poruka, korisnik poslužiocu saopštava identifikacioni broj kanala koji je otvorio, kao i prolaz na kome RVU poslužilac može da otvori novi kanal za naredne korake u komunikaciji.

Kroz komandni kanal, RVU korisnik šalje sledeće vrste poruka:

- Pozdravne poruke (engl. Hello) – započinjanje sesije i uvek je prva poruka poslata nakon otvaranja komandnog kanala ili kanala podataka

atributi poslužioca:	atributi korisnika:	opis:	tip:
commandToken	commandToken	jedinstveni ID koji predstavlja komandu	uint
channelId	channelId	identifikator kanala	uint
callbackPort	callbackPort	povratni prolaz koji se koristi za kreiranje novih kanala	uint
tlsPort	tlsPort	opcioni atribut: prolaz koji se koristi za kreiranje novog	uint
version	version	verzija protokola	string
n/a	UDN	opcioni atribut: vrednost koja predstavlja UDN UPnP	string

Tabela 3.1 Struktura pozdravne poruke

- Odjavne(engl. shutdown) poruke – završetak sesije, zaustavljanje primanja svih poruka, kao i obustavljanje mogućnosti za otvaranje novih kanala

atributi poslužioca:	atributi korisnika:	opis:	tip:
commandToken	commandToken	jedinstveni ID koji predstavlja komandu	uint

Tabela 3.2 Struktura odjavne poruke

- Informacije o karakteristikama- podržane korisničke akcije, A/V formati, rezolucije, operacije iscrtavanja, informacije o slobodnoj memoriji itd.
- Korisničke akcije (događaji sa ekrana osetljivog na dodir, tastature, miša i daljinskog upravljača). Na izdate komande korisnik se kreće kroz menije, pri čemu RVU korisnička aplikacija preko RVU-RUI protokola obaveštava RVU poslužioca o koordinatama grafičkih elemenata koje treba izmeniti i osvežiti. Na ovaj način, izbegnuto je učestalo osvežavanje celog zaslona što rezultuje povećanjem brzine i kvaliteta korisničke aplikacije.

komanda:	HDMI korisnička akcija[2]	HDMI ID operacije	CDI vrednost	Komentar:
PWR[1]	pwr	N/A	N/A	
format[1]	N/A	N/A	N/A	
TV ulaz[1]	N/A	N/A	N/A	
označi	označi	0x00	0xE001	
gore	gore	0x01	0xE100	
dole	dole	0x02	0xE101	
levo	levo	0x03	0xE102	
desno	desno	0x04	0xE103	
meni	meni	0x0A	0xE503	
kanal unapred	kanal unapred	0x30	0xE006	
kanal unazad	kanal unazad	0x31	0xE007	
pojačavanje zvuka[1]	pojačavanje	0x41	0xE003	

smanjivanje	smanji zvuk	0x42	0xE004	
brojevi 0-9	brojevi 0-9	0x20-0x09	0xE300-0xE309	
utišavanje[1]	utišavanje	0x43	0xE005	
prethodni	prethodni	0x32	0xE504	
enter	enter	0x2B	0xE505	
crta	crta	0x2A	0xE506	
reprodukuj	reprodukuj	0x44	0xE400	
prekid	prekid	0x45	0xE402	
pauziranje	pauziranje	0x46	0xE401	
snimanje	snimanje	0x47	0xE403	
premotavanje	premotavanje	0x48	0xE406	
brzo unapred	brzo unapred	0x49	0xE405	
napredno	unapred	0x4B	0xE408	napredna reprodukcija po 30 sekundi
skok unazad	unazad	0x4C	0xE409	premotavanje unazad po 10 sekundi
vodič	EPG	0x53	0xE00B	
aktivan	N/A	0x91[3]	0xE500	interaktivni portal
izlistavanje	meni sadržaja	0x0B	0xE501	prikaži listu za reprodukciju
izlaz	izađi	0x0D	0xE502	vrati se na prenos uživo ili na snimljeni sadržaj
povratak	N/A	0x90[3]	0xE002	vrati se jedan nivo
informacije	informacije o	0x35	0xE00E	prikaži informacije o izabranom programu
plavo	F1(plavo)	0x71	0xE203	prikaži mini-vodič
crveno	F2(crveno)	0x72	0xE200	obriši odabranu stavku liste
zeleno	F3(zeleno)	0x73	0xE201	izaberi zvučne opcije
žuto	F4(žuto)	0x74	0xE202	izaberi TV opcije

Tabela 3.3 Korisničke akcije

[1] – ove komande se ne šalju RVU poslužiocu, stoga ih nije potrebno mapirati na HDMI

[2] – Nazivi operacija

[3] – Komande nemaju odgovarajuće HDMI korisničke operacije

atribut:	opis:	tip:
commandToken	jedinstveni ID koji predstavlja komandu	uint
keyVal	ID operacije predstavljen u HDMI specifikaciji u CEC tabeli,	hex
event	1: događaj pritiska na dugme 0: događaj otpuštanja dugmeta	uint

Tabela 3.4 Atributi HDMIKeyEvent komande

Korisničke akcije je moguće zaštititi od neovlašćenog pristupa uz pomoć TLS protokola (engl. Transport Layer Protocol). Zaštita se koristi u slučaju kada korisnik unosi osetljive informacije, poput šifri, broja računa i slično.

TCP poslužilac prima poruke koje šalje RVU poslužilac. Poruke se raspoređuju po kanalima u odnosu na tip (komandne poruke ili podaci) i prosleđuju do modula za rukovanje porukama, koji ih parsira i obavlja zahtevanu akciju. Poruke se smeštaju u red (engl. First in first out (FIFO)) i shodno komandama, obrađuju u unapred predviđenim vremenskim intervalima.

Tipične poruke koje TCP poslužilac prima su:

- Grafičke poruke – Sadrže grafičke komponente
- Audio poruke - Sadrže audio komponente (zvučne efekte)
- Konfiguracione poruke– Zauzimanje bafera za audio i grafičke komponente, podešavanje prozora za video, otvaranje i zatvaranje audio dekodera itd.
- Kontrolne poruke (tabela 3.3)– Kontrola reprodukcije zvuka (startovanje, stopiranje), jačine zvuka, dobavljanje informacija o mogućnostima RVU korisnika itd.

Modul za rukovanje kanalima je zadužen za otvaranje novih kanala, zatvaranje kanala, kao i prosleđivanje poruka na dalju obradu u modul za rukovanje porukama. Pod terminom „kanal“, podrazumevamo jednu TCP utičnicu između RVU korisnika i poslužioca. Kanali se mogu podeliti u odnosu na tip poruka kojim rukuju. Mogu biti:

- Komandni – komande se šalju posredstvom XML poruka (UTF-8 enkodovane XML poruke). Inicijalna poruka mora biti Hello. Ukoliko je pozdravna poruka poslata u bilo kom drugom trenutnu osim na početku, RVU element koji je primi, mora obavezno odgovoriti sa ERR_INVALID_STATE.
  Kanal se smatra komandnim ukoliko je prvi poslati bajt znak „<“.
  Format poruke je sledeći:

```
<commandName commandToken=”commandTokenValue”
attributeName1=”attributeValue1” ...
attributeNameN=”attributeValueN”/>
```

  Svaka XML poruka počinje sa nazivom komande, koji je string. Atribut commandToken je obavezan, dok commandTokenValue predstavlja string predstavu 31-bitnog nenegativnog broja. Po RVU elementu, jedinstven je u periodu od 10 sekundi. Svaki XML može sadržati dodatne atribute. AttributeNames je takođe string, dok attributeValueN može biti string predstava različitog formata (zavisi od komande).
  Na komande primljene na određenom kanalu, odgovori se šalju na tom istom kanalu u obliku sledećeg formata odgovora:

```
<Response errCode=”errorCodeValue”commandToken=”commandTokenValue”
attributeName1=”attributeValue1” ...
attributeNameN=”attributeValueN”/>
```

Odgovor uvek počinje sa „Response“ stringom. Sadrži errCode atribut i errorCodeValue je string. CommandTokenValue je uvek isti kao i u početnoj poruci opisanoj iznad. Broj dodatnih atributa može biti 0 ili više. Nazivi atributa (attributeNames) su u obliku stringa, dok format attributeValues zavisi od komande.

- Kanali podataka – podaci se šalju kao nizovi bajtova. Vrste podataka koje se mogu slati preko ovih kanala su:
  - Pozdravna (engl. Hello) – Obavezna, inicijalna poruka, predstavljena preko ASCII cifara
  - PixelData – Sadrži piksele koji predstavljaju grafičke komponente
  - CompressedPixelData – Kompresovani oblik predstavljanja grafičkih komponenata, enkodovan u zlib formatu
  - JPEGPixelData - Predstavlja piksele enkodovane u JPEG formatu
  - PNGPixelData - Predstavlja piksele enkodovane u PNG formatu
  - CLUT – Binarni CLUT podati gde je svaka stavka poslata u 32-bitnom ARGB-32 formatu
  - AudioData – Audio podaci. Podržani audio kontejneri su: MPEG2-ES, MPEG2-PES, MPEG1-PACKET, PCM.

    Podržani audio formati su: MPEG2, MPEG2-LAYER3, PCM, AC3, DTS, MPEG2-AAC, MPEG4-AAC, AC3-PLUS

Inicijalna poruka mora biti pozdravna. Ukoliko je ona poslata u bilo kom drugom trenutku osim na početku, RVU element koji je primi, mora je odbaciti. RVU element koji šalje radni okvir, mora dostaviti i odgovarajući ID kanala podataka. Podatke mogu slati i RVU poslužioci i RVU korisnici.

U sledećoj tabeli, prikazana je struktura poruke poslate kroz kanal podataka.

Sintaksa:	Broj bita:	Format:	Komentar:
RVU-RUI-Data-Frame() {			
frame-header {			
frame-header-len	32	ASCII-kodiran decimalni	4-cifre, 0 postavljenih
Separator1	8	0x20	
frame-type-id	8*broj karaktera	ASCII karakteri	
separator2	8	0x20	
command-token	8* broj karaktera	ASCII- kodiran decimalni	
separator3	8	0x20	
frame-data-len	8* broj karaktera	ASCII- kodiran decimalni	
CRLF1	16	0x20	
}		0x0D0A	
frame-data	8*frame-data-len	uimsbf	
CRLF2	16	0x0D0A	
}			

Tabela 3.5 Struktura poruke u kojoj se šalju podaci

- frame-header-len predstavlja polje veličine 4 bajta, enkodovano u ASCII formatu, koje označava dužinu okvira zaglavlja.
- frame-type-id predstavlja tip podataka koji može biti (Hello, PixelData,AudioData itd.).
- command-token – predstavlja celobrojnu cifru, koja odgovara CommandToken atributu odgovarajuće komandne poruke. Ako podaci nisu pridruženi nijednoj komandnoj poruci, ovo polje ima vrednost 0. Vrednost ovog polja enkodovana je kao broj u ASCII formatu.
- frame-data-len – ovo polje označava dužinu okvira podataka. Ne uključuje poslednje CRLF bajte i enkodovano je kao broj u ASCII formatu.
- frame-data – ovo polje promenljive dužine, sadrži podatke pridružene određenoj RUI komandnoj poruci. Broj bajta naveden je u frame-data-len polju, koje je definisano u frame-header-len.

Modul za rukovanje porukama ima zadatak da raščlanjuje poruke primljene preko modula za rukovanje kanalima, kao i da sastavlja odgovore na te poruke, ukoliko je to neophodno. Poruke mogu nositi podatke ili informacije o komandama.

Kada stigne komandna poruka, ona se parsira i preduzima se određena akcija. U zavisnosti od uspešnosti akcije, sastavlja se odgovori šalje se kroz kanal kroz koji je poruka i stigla.

Osnovne komandne poruke navedene su u sledećim tabelama:

Naziv komande:	Opis komande:	Tip
Hello	Inicijalna poruka za otvaranje kanala	-
Shutdown	Zatvaranje kanala	-
GetMemInfo	Dobavljanje informacija o memorijskim resursima	-

Tabela 3.6 Uspostavljanje i raskidanje RVU-RUI sesije i informacije o memoriji

Naziv komande:	Opis komande:	Tip
HDMIKeyEvent	Dodađaj u HDMI formatu (obavezan format)	-
CDIKeyEvent	Dodađaj u CDI format (opcioni format)	-
GetKeyList	Dobavljanje informacija o listi podržanih događaja	-

Tabela 3.7 Događaji na strani korisnika

Naziv komande:	Opis komande:	Tip
AllocateBuffer	Zauzimanje memorije za grafičke elemente	-
DeallocateBuffer	Oslobađanje memorije zauzete za grafičke elemente	-
Write	Upis podataka u grafički bafer	JPEGPixelData
Read	Čitanje podataka iz grafičkog bafera	JPEGPixelData
BlitQueue	Komanda za grupisanje blit komandi	-
Dispatch	Izvršavanje grupisanih blit komandi	-
EmptyQueue	Brisanje grupisanih komandi	-
WaitVSync	Čekanje korisničkog Vsync perioda	-
CopyBlit	Kopiranje grafičkih podataka	-

FillBlit	Popunjavanje regiona određenom bojom	-
ResizeBlit	Skalirano kopiranje grafičkih podataka	-
ShadeBlit	Porter-Duff spajanje konstantom bojom	-
BlendBlit	Porter-Duff spajanje dve grupe grafičkih podataka	-
MultiSourceBlendBlit	Porter-Duff spajanje dva grafička bafera	-
ResizeAndBlendBlit	Skalirano Porter-Duff spajanje dve grupe grafičkih podataka	-
ColorKeyResizeBlit	Skalirano kopiranje grafičkih podataka uz poređenje sa bojom	-
GetGraphicsCaps	Informacije o podržanim grafičkim operacija i parametrima	-
GetZList	Dobavljanje rasporeda grafičkih bafera po Z osi	-
SetZList	Postavljanje rasporeda grafičkih bafera po Z osi	-
SetCLUT	Upis CLUT tabele u grafički bafer	CLUT

Tabela 3.8 Grafičke komande

Nazivkomande:	**Opis komande:**	**Tip**
GetVideoBuffer	Povezivanje AVTransport sesije sa video baferom	-
ReleaseVideoBuffer	Otpuštanje kontrole nad video baferom	-
SetBackgroundColor	Postavljanje pozadinske boje	-
ConfigureDisplayBuffer	Konfigurisanje bafera za prikaz	-
ReconfigureDisplayBuffe	Ponovno konfigurisanje bafera za prikaz	-
ConfigureVideoFullscree	Konfigurisanje video bafera za prikaz preko celog	-
ConfigureVideoWindow	Konfigurisanje video bafera za prikaz unutar GKS-e	-
ConfigureWindowedVid	Konfigurisanje dela video bafera za prikaz unutar GKS-	-
ConfigureVideoDecoder	Konfigurisanje rezolucije za dekodovanje videa	-
BlindVideo	Prikrivanje video bafera	-
GetOutputSettings	Dobavljanje rezolucije ekrana	-
OutputSettingsChanged	Promena rezolucije ekrana	-
GetVideoDisplaySettings	Dobavljanje podešavanja video bafera	-
VideoDisplaySettingsCha	Promena podešavanja video bafera	-
AllowedClosedCaptionin	Dozvola prikaza prevoda	-
EnableClosedCaptioning	Uključivanje/isključivanje prevoda	-
GetCloseCaptioningState	Dobavljanje podešavanja za prevode	-
ReconfigureDisplayBuffe	Ponovno konfigurisanje bafera za prikaz u slučaju 3DTV podrške	-
Set3DTVFlattenStructure	Podešavanje 3DTV parametara	-

Tabela 3.9 Komande za reprodukciju AV sadržaja

Naziv komande:	**Opis komande:**	**Tip**
OpenAudioDecoder	Otvaranje audio dekodera	-
CloseAudioDecoder	Zatvaranje audio dekodera	-
GetNumAudioDecoders	Dobavljanje broja audio dekodera	-
GetAudioDecoderCaps	Dobavljanje informacija o audio dekoderima	-
AllocateAudioBuffer	Zauzimanje audio bafera	-
DeallocateAudioBuffer	Oslobađanje audio bafera	-
WriteAudioData	Upis audio podataka u zadati audio bafer	AudioData
Play	Reprodukcija audio podataka dostupnih na kanalu za podatke	AudioData
PlayBuffer	Reprodukcija audio podataka dostupnih u audio baferu	-
PlayStatus	Status reprodukcije audio podataka	-
Stop	Zaustavljanje reprodukcije audio podataka	-

Tabela 3.10 Audio komande

Podaci se, takođe, raščlanjuju i u zavisnosti da li se radi o grafičkim ili zvučnim komponentama, popunjavaju se grafički ili audio baferi koje je RVU poslužilac zauzeo u modulu za rukovanje baferima. Nakon toga, baferi se prosleđuju do prilagodnog sloja u kome se obrađuju ili propagiraju dalje.

Elementi grafičke korisničke sprege se šalju u kompresovanim formatima PNG ili JPEG, zbog činjenice da slanje nekompresovanih elemenata opterećuje mrežu i unosi kašnjenja u prenosu sadržaja.

Modul za rukovanje baferima ima zadatak da omogući rad sa listama koje sadrže grafičke ili audio komponente. Baferi se mogu dinamički zauzeti i osloboditi u toku sesije. RVU korisnik mora odvojiti minimalno 64 MB memorije, za slučaj najgoreg scenarija.

- Zauzimanje grafičkih bafera - Nakon inicijalnog uspostavljanja veze između RVU korisnika i RVU poslužioca, poslužilac zauzima memoriju (tj. zaslonski bafer) za smeštanje grafičke korisničke sprege koja je trenutno vidljiva korisniku. U toku sesije, RVU poslužilac može da zauzme više dodatnih grafičkih bafera u koje se smeštaju grafičke komponente.
  Korišćen je koncept višestrukog baferovanja, gde se grafički elementi smeštaju u više privremenih bafera i nakon završetka obrade iscrtavaju na glavnom, zaslonskim baferu i prikazuju korisniku. Ovaj segment je detaljnije opisan u poglavlju 3.2.3.
- Zauzimanje audio bafera – RVU poslužilac može da zauzme jedan ili više audio bafera za smeštanje audio komponenata koje upotpunjuju doživljaj korišćenja aplikacije.

3.3.2.3 Digitalni media renderer

RVU standard propisuje da svaki RVU korisnik mora sadržati Digitalni media renderer (engl. Digital Media Renderer (DMR)) uređaj, prema specifikaciji DLNA standarda uz izmene svojstvene RVU korisniku. Zadatak ovog uređaja jeste da prima komande koje mu prosleđuje DLNA DMC uređaj i na taj način opslužuje korisnika A/V sadržaja informacijama.

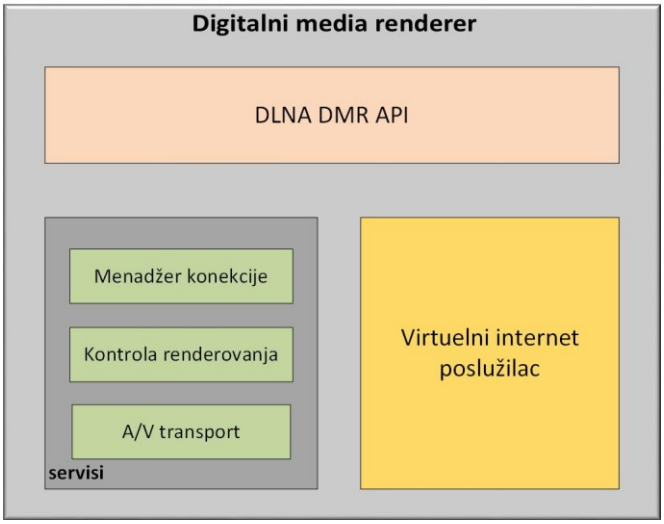
Nakon što korisnik kretanjem kroz aplikaciju, odabere određeni A/V sadržaj skladišten na DMS uređaju, upravljač media rendereru dostavlja HTTP adresu sa koje treba da pokrene reprodukciju A/V sadržaja. Pored toga, na zahtev korisnika i preko komandnih poruka media upravljača, media renderer kontroliše reprodukciju multimedijalnog sadržaja.

U podržane komande spadaju:

- Započni reprodukciju (engl. Normal Playback)
- Prekini reprodukciju (engl. Stop)
- Ubrzano premotavanje unapred (engl. Fast forward scan)
- Ubrzano premotavanje unazad (engl. Fast rewind scan)

- Sporo premotavanje unapred (engl. Slow forward scan)
- Premotavanje okvira (engl. Frame step forward)
- Pauziraj/nastavi (engl. Pause/resume)
- Vremensko poziciono premotavanje (engl. Time-based position seeking)

Prema DLNA standardu, DMR uređaj mora da sadrži servise prikazane na slici 3.5. Pored servisa, za potrebe implementacije, iskorišćen je UPnP virtuelni internet poslužilac koji izlaže informacije o DMR uređaju i njegovim servisima u obliku XML dokumenta.



Slika 3.5 Arhitektura digitalnog media renderera (DMR)

DLNA DMR API predstavlja aplikativnu programsku spregu koja omogućava korišćenje DMR uređaja predviđenog DLNA standardom u okviru RVU korisničke programske sprege.

DMR aplikativna programska sprega apstrahuje funkcije servisa i virtuelnog internet poslužioca i obezbeđuje jednostavno integrisanje DMR uređaja u bilo koje rešenje.

Kontrola renderovanja (engl. Rendering Control) [15] predstavlja UPnP servis čiji osnovni zadatak je da dinamički podešava određene podesive parametre renderovanja. Na primer, televizori spadaju u uređaje sposobne da reprodukuju video sadržaj i omogućuju korisniku da podešava oštrinu slike, osvetljenje, kontrast i slično. Audio rendereri omogućavaju podešavanje balansa zvuka, jačine zvuka, jačine basa itd.

Osnovna namena RenderingControl:1 servisa, jeste da omogući kontrolnim tačkama upoznavanje sa vrednostima parametara renderera, kao i njihovu dinamičku izmenu.

RenderingControl:1 omogućava kontrolnim tačkama:

- Otkrivanje seta atributa podržanih od strane uređaja

- Dobavljanje vrednosti određenog atributa
- Promena vrednosti atributa (tj. Kontrola)
- Vraćanje atributa na početnu vrednost

Servis kontrole renderovanja odgovara sledećem šablonu: *urn:schemas-UPnP-org:service:RenderingControl:1* koji propisuje obavezne i opcione akcije, atribute i promenljive stanja. RVU proširuje skup funkcionalnosti ovog servisa, tako što omogućuje rad sa audio i video tokovima podataka.

RVU protokol proširuje skup promenljivih stanja sa sledećim stanjima:

- **X_AudioPID** (RVU-M) – predstavlja PID toka audio podataka koji se trenutno dekoduju. Ovoj neoznačenoj dvobajtnoj promenljivoj se pristupa isključivo pomoću akcija: X_UpdateAudioSelection i X_GetAudioSelection.
- **X_AudioEncoding** (RVU-M) - predstavlja tip enkodovanja trenutno enkodovanog toka audio podataka. Moguće vrednosti su: „mp3”, „mpeg1layer2”, „pcm”, „aac”, „ac3”, „eac3”. Promenljivoj se pristupa pomoću akcija: X_UpdateAudioSelection i X_GetAudioSelection.
- **X_VideoPID** (RVU-M) - predstavlja PID toka MPEG2 video podataka koji se trenutno dekoduju. Ovoj neoznačenoj dvobajtnoj promenljivoj se pristupa isključivo pomoću akcija: X_UpdateVideoSelection i X_GetVideoSelection.
- **X_VideoEncoding** (RVU-M) - predstavlja tip enkodovanja trenutno enkodovanog toka audio podataka. Moguće vrednosti su: „mpeg2” i „h264”. Promenljivoj se pristupa pomoću akcija: X_UpdateVideoSelection i X_GetVideoSelection.

I sledećim akcijama:

- **X_UpdateAudioSelection** (RVU-M) – omogućuje RVU poslužitelju da osveži trenutni audio PID i tip enkodovanja na rendereru. Kao argumente, koristi InstanceID (ulazni), AudioPID (ulazni) i AudioEncoding (ulazni) koji odgovaraju sledećim stanjima A_ARG_TYPE_InstanceID, X_AudioPID i X_AudioEncoding, respektivno.
- **X_GetAudioSelection** (RVU-M) - omogućuje RVU poslužitelju da dobavi trenutni audio PID i tip enkodovanja na rendereru. Kao argumente, koristi InstanceID (ulazni), AudioPID (izlazni) i AudioEncoding (izlazni) koji odgovaraju sledećim stanjima A_ARG_TYPE_InstanceID, X_AudioPID i X_AudioEncoding, respektivno.
- **X_UpdateVideoSelection** (RVU-M) - omogućuje RVU poslužitelju da osveži trenutni video PID i tip enkodovanja na rendereru. Kao argumente, koristi InstanceID (ulazni),

VideoPID (ulazni) i VideoEncoding (ulazni) koji odgovaraju sledećim stanjima A_ARG_TYPE_InstanceID, X_VideoPID i X_VideoEncoding, respektivno.

- **X_GetVideoSelection** (RVU-M) - omogućuje RVU poslužitelju da dobavi trenutni video PID i tip enkodovanja na rendereru. Kao argumente, koristi InstanceID (ulazni), VideoPID (izlazni) i VideoEncoding (izlazni) koji odgovaraju sledećim stanjima A_ARG_TYPE_InstanceID, X_VideoPID i X_VideoEncoding, respektivno.

Menadžer konekcije (engl. ConnectionManager) [16] predstavlja UPnP servis zadužen za uspostavljanje veze između uređaja u cilju deljenja A/V sadržaja. Svaki uređaj koji poseduje mogućnost da šalje ili prima A/V sadržaj, mora sadržati ovaj servis. Menadžer konekcije omogućava kontrolnim tačkama da:

- Utvrde da li dva uređaja poseduju mogućnost da se povežu u cilju deljenja multimedijalnog sadržaja
- Pronađu informacije o deljenju A/V sadržaja koji je u toku
- Uspostave i prekinu konekciju između dva uređaja

Servis menadžera konekcije odgovara sledećem šablonu: *urn:schemas-UPnP-org:service:ConnectionManager:1* koji propisuje obavezne i opcione akcije, atribute i promenljive stanja.

RVU uvodi nova pravila u funkcionisanju ovog servisa, tako što uvodi dve UPnP opcione akcije kao obavezne:

- **PrepareForConnection** – Ova akcija omogućava uređaju da se pripremi za povezivanje na mrežu u cilju deljenja A/V sadržaja.
- **ConnectionComplete** – Kontrolna tačka poziva ovu akciju za sve konekcije uspostavljene preko *PrepareForConnection* akcije u cilju utvrđivanja da li su svi resursi pridruženi konekcijama uspešno oslobođeni.

A/V transport [17] omogućava kontrolu nad prenosom audio i video transportnih tokova. Ovaj servis definiše model A/V transporta koji odgovara određenoj grafičkoj korisničkoj sprezi. Koristi se za kontrolu širokog spektra uređaja poput CD i MP3 plejera, frekvencijskog odabirača, VCR itd.

A/V transport definiše prelazna stanja kao što je definisano u specifikaciji servisa. RVU protokol uvodi 3 dodatna stanja:

- Pokušaj premotavanja (engl. Seek) ne sme biti odbačen u stanju pauzirane reprodukcije (engl. PAUSED_PLAYBACK) ukoliko je opseg premotavanja ispravan

- Pokušaj premotavanja ne sme biti odbačen u stanju u stanju reprodukcije (engl. PLAYING) ukoliko je opseg premotavanja ispravan
- Pokušaj premotavanja ne sme biti odbačen u stanju zaustavljene reprodukcije (engl. STOPPED) ukoliko je opseg premotavanja ispravan

Virtuelni internet poslužilac (engl. virtual web server) ima zadatak da deli datoteke na internetu posredstvom UPnP ugrađenog mehanizma za realizaciju internet poslužioca.

Datoteke su zapravo stringovi koji se posredstvom ovog modula dele na internetu u vidu XML datoteka, kojima pristupaju UPnP uređaji u cilju ostvarivanja faze opisivanja.

Primer virtuelne datoteke predstavlja string koji opisuje DLNA DMR uređaj. Ukoliko mu se neki DLNA DMC obrati, DMR šalje internet adresu sa koje DMC preuzima XML dokument koji sadrži osnovne informacije o uređaju, poput jedinstvenog identifikacionog broj (UDN), imena (engl. Friendly name), naziva proizvođača (engl. Manufacturer name), adresa proizvođača (engl. Manufacturer URL) itd.

3.3.3 Libupnp

Libupnp predstavlja biblioteku otvorenog koda prvenstveno namenjenu obezbeđivanju funkcionalnosti za izgradnju i kontrolisanje UPNP uređaja, poput kontrolnih tačaka i poslužilaca. Osnovne informacije o UPnP protokolu prezentovane su u teorijskim osnovama (poglavlje 2.2).

Libupnp obezbeđuje sledeće alate:

- threadutil – sadrži osnovne funkcionalnosti za rad sa nitima (ithread, ThreadPool), spregnutim i slobodnim listama (LinkedList, FreeList) i vremenskim brojačima (TimerThread)
- ixml – omogućuje rukovanje xml dokumetima (ixml), npr: izgradnju dokumenta, raščlanjivanje, izmenu itd, i uklanjanje grešaka (ixmldebug).
- UPnP – sadrži funkcionalnosti za izgradnju UPnP odgovarajućih uređaja i realizaciju njihovih sposobnosti kroz pridruživanje servisa svakom od njih. Pored toga, omogućuje i rad sa virtuelnim poslužiocem u cilju deljenja informacija i sadržaja u okviru internet mreže.

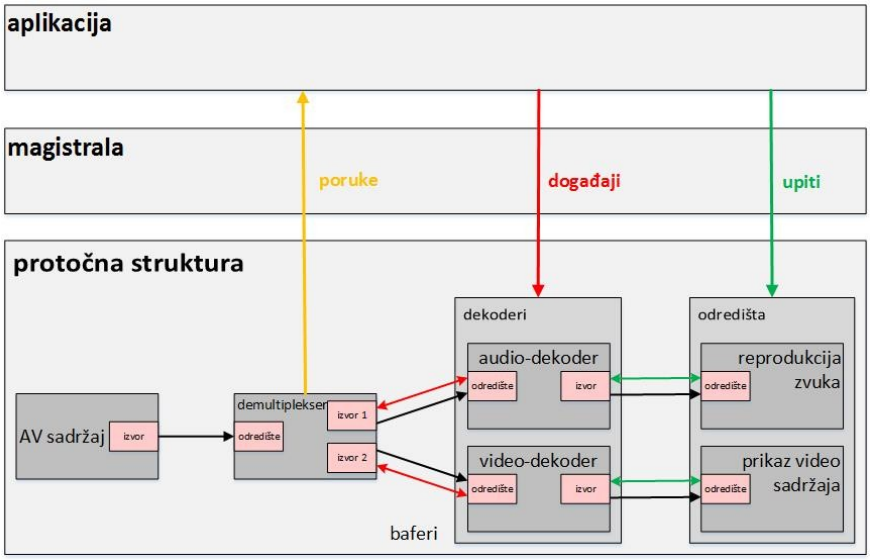
Za potrebe rada iskorišćena je libupnp biblioteka u verziji 1.6.12.

3.3.4 GStreamer

GStreamer [11] predstavlja biblioteku otvorenog koda napisanu u C programskom jeziku. Biblioteka je prevashodno namenjena za rukovanje multimedijalnim sadržajem i iscrtavanje grafika. U ovom slučaju, iskorišćena je za potrebe reprodukcije AV

multimedijalnog sadržaja u MPEG2 formatu, dostavljenog od strane HTTP poslužioca smeštenog na strani RVU poslužioca.

GStreamer biblioteka poseduje arhitekturu baziranu na protočnoj strukturi (engl. pipeline). Protočna struktura (slika 4.6) predstavlja niz elemenata koji omogućavaju protok multimedijalnih podataka u cilju njhove obrade i funkcioniše u zasebnoj niti.



Slika 3.6 Protočna struktura GStreamer biblioteke

Kao što je prikazano na slici, komunikacija između aplikacije i protočne strukture ostvaruje se slanjem poruka, događaja i upita, dok se preko bafera, prosleđuju podaci između elemenata protočne strukture.

- baferi – služe sa skladištenje podataka koji se prenose između elemenata protočne strukture i uvek se transportuju od izvora ka odredištu
- događaji – su objekti koji se prenose između elemenata protočne strukture, ili između aplikacije i protočne strukture. Mogu se prenositi između izvora i odredišta u oba smera.
- poruke – predstavljaju objekte koji koje elementi protočne strukture prenose do magistrale, odakle im može pristupiti aplikacija. Poruke služe da prenose informacije, poput grešaka, promena stanja, stanja baferovanja itd, od elemenata protočne strukture pa do aplikacije.
- upiti – objekti koji omogućavaju aplikaciji da zatraži informacije kao što su dužina trajanja reprodukcije ili trenutna pozicija reprodukcije, od protočne strukture.

U cilju osposobljavanja biblioteke za korišćenje u okviru Androida, biblioteka je prilagođena ciljnoj platformi kroz realizaciju JNI sprege, kao i apstrakcije poziva biblioteke u Java aplikativnom sloju.

4. Programsko rešenje

Programsko rešenje sadrži detaljan opis modula neophodnih za realizaciju RVU korisničke programske podrške. U konceptu rešenja, predstavljeni su osnovni slojevi koji čine RVU korisničku programsku podršku, a to su: Java aplikativni sloj, prilagodni sloj (Java izvorna sprega) i izvorne biblioteke.

4.1 Java aplikativni sloj

Aplikativni sloj napisan je u osnovnom Androidovom programskom jeziku – Javi. Pošto je ideja RVU protokola izgraditi „tanku" korisničku programsku spregu, u ovom sloju neophodno je obezbediti osnovni koncept za iscrtavanje grafičkih elemenata koji grade grafičku korisničku spregu, kao i reakcije na korisničke akcije.

Ovaj sloj se sastoji iz cetiri Java klase, sadržane u paketu com.amuse.rvu:

- RVUActivity
- AudioRenderer
- GraphicRenderer
- RVUClient
- GStreamerPlayer

4.1.1 RVUActivity

RVUActivity klasa nasleđuje Androidovu ugrađenu klasu Activity i predstavlja osnovnu i jedinu aktivnost RVU korisničke programske podrške. U njoj se postavlja i podešava GLSurfaceView komponenta neophodna za korišćenje OpenGL ES 2.0 aplikativne programske sprege, inicijalizuju JNI komponente za crtanje grafičke korisničke sprege, reprodukciju zvučnih efekata, RVU korisnička biblioteka, GStreamer modul za reprodukciju AV sadržaja,kao i reagovanje na korisničke akcije.

Ova klasa sadrži dva privatna polja: GLSurfaceView glSurfaceView i boolean rendererSet. Prvo polje omogućava rad sa GLSurfaceView Android komponentom, dok drugo polje pokazuje da li je Renderer pridružen GLSurfaceView komponenti.

Svaka aktivnost ima svoj životni vek, čije faze je neophodno ispoštovati u cilju realizacije korektne Android aplikacije.

Pri kreiranju aktivnosti, potrebno je preklopiti metodu *onCreate* nasleđene Androidove klase Activity. *OnCreate* metoda se poziva pri kreiranju aktivnosti i u njoj se inicijalizuju i podešavaju osnovne komponente i objekti koji se koriste u programu.

U glavnoj aktivnosti, pravi se objekat RVUClient klase, koja učitava RVU izvornu biblioteku, omogućava inicijalizaciju i deinicijalizaciju biblioteke i prosleđivanje korisničkih događaja RVU poslužiocu.

Posto RVU korisnička programska sprega koristi OpenGL ES verzije 2.0, neophodno je proveriti da li uređaj na kome je pokrenuta aplikacija, podržava ovu veziju OpenGL-a. To je urađeno dobavljanjem sistemskog servisa *Context.ACTIVITY_SERVICE*preko objekta klase ActivityManager. Uz pomoć dobavljenog objekta, pravi se objekat klase ConfigurationInfo uz pomoć metode *getDeviceConfigurationInfo().*Preko objekta klase ConfigurationInfo, možemo saznati da li je OpenGL ES 2.0 podržan:

configurationInfo.reqGlEsVersion>=0x20000.

Ukoliko je prethodni izraz zadovoljen, pravi se objekat klase GlSurfaceView. Nakon toga se nad tim objektom poziva metoda *setEGLContextClientVersion(int version)* kojoj se prosleđuje verzija OpenGL ES, sto je u našem slučaju - *glSurfaceView.setEGLContextClientVersion(2).*

Da bi se omogućilo iscrtavanje grafičkih elemenata, neophodno je pridružiti objekat klase Renderer objektu glSurfaceView na sledeći način: *glSurfaceView.setRenderer(newRendererWrapper()).* Polje rendererSet se podešava na *true* i glSurfaceView se prosleđuje metodi *setContentView(View)* koja omogućava prikaz grafičke korisničke sprege koristeći OpenGL ES 2.0 aplikativnu spregu u okviru GLSurfaceView.Ukoliko uređaj ne podržava OpenGL ES 2.0, RVU aplikacija neće biti funkcionalna i korisnik se preko *Toast.makeText* metode obaveštava o tome.

Pored objekata za pristup izvornim bibliotekama za obradu grafike i kontrolisanje RVU korisničke programske podrške, neophodno je napraviti i objekat klase AudioRenderer, zadužene za učitavanje audio biblioteke u JNI sloju, kao i za inicijalizaciju, deinicijalizaciju i podešavanje audio komponenata za reprodukciju zvučnih efekata dobavljenih iz RVU biblioteke.

Aplikacija je prilagođena različitim vrstama uređaja, poput televizora, tableta ili telefona, i podržava tri tipa korisničkih akcija: dodir, prevlačenje i pritisak dugmeta. Prve dve korisničke akcije se koriste na tabletima i telefonima, dok se poslednja akcija koristi na televizorima i digitalnim TV prijemnicima u slučaju izdavanja komande preko daljinskog upravljača.

Da bi aplikacija reagovala na korisničke akcije, neophodno je postaviti osluškivače događaja nad glSurfaceView komponentom.

Osluškivače predstavljaju:

- *glSurfaceView.SetOnTouchListener(OnTouchListener)* – kao parametar se prosleđuje realizovana sprega View.OnTouchListener koji, kada se desi događaj, poziva metodu*onTouch(View v, MotionEvent motionEvent)*.

  Možemo razlikovati dva tipa događaja: dodir i prevlačenje prsta po ekranu. Kada se desi neki od ovih događaja, koordinate x i y se normalizuju zbog razlika između Androidovog i openGL koordinatnog sistema i prosleđuju rendereru, koji ih dalje propagira ka JNI sloju uz pomoć metode*int RvuSendMessage(String message, int key_code, String event)*(tabela 3.4),koja prima poruku, komandu (tabele 3.1 – 3.3) i događaj da li je pritisnuto ili otpušteno dugme.
- *glSurfaceView.SetOnClickListener(OnClickListener)* – kao parametar se prosleđuje realizovana sprega View.OnClickListener, koji kada se desi događaj, poziva metodu *onClick(View v).*

  Kada se desi ovaj događaj, koordinate X i Y se normalizuju i prosleđuju rendereru uz pomoć metode int *RvuSendMessage(String message, int key_code, String event).*

Osim *onCreate* metode, neophodno je preklopiti i metode *onPause* i *onResume* u cilju obaveštavanja glSurfaceView komponente o stanju aktivnosti. Ukoliko aktivnost prelazi u stanje pauziranosti (*onPause()*), a polje rendererSet je podešeno na *true*, poziva se *glSurfaceView.onPause()*.

Ako aplikacija nastavlja svoje izvršavanje (*onResume()*), a polje rendererSet je takođe podešeno na *true*, poziva se *glSurfaceView.onResume().*

Ovi pozivi omogućavaju GLSurfaceView komponenti da zaustavi ili nastavi izvršavanje niti koja koja koristi OpenGL za iscrtavanje grafičke korisničke sprege.

Kao sto je detaljno objašnjeno, aplikacija je bazirana na OpenGL ES 2.0 i neophodno je u AndroidManifest.xml datoteci isključiti podršku za starije uređaje koji ne podržavaju tu verziju OpenGL ES. To se radi na sledeći način:

```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

Kao minimalna verzija SDK, navedena je API verzija 10 (Android Gingerbread 2.3.3) iz razloga što je to prva verzija Androida koja u potpunosti podržava OpenGL ES verzije 2.0.

```
uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="17" />
```

4.1.2 GraphicsRenderer

GraphicsRendererklasa realizuje GLSurfaceView.Renderer spregu koja ima zadatak da koristi pozive OpenGL ES aplikativne programske sprege u cilju iscrtavanja grafičkih elemenata. Pored toga, u ovoj klasi se učitava grafička izvorna dinamička biblioteka smeštena u direktorijumu */libs* i deklarišu izvorne metode koje se pozivaju u preklopljenim metodama GLSurfaceView.Renderer sprege.

Sledeće metode članice GLSurfaceView.Renderer se preklapaju u cilju osposobljavanja Renderera za crtanje u JNI sloju:

- *public void onSurfaceCreated(GL10 gl, EGLConfig config)* – poziva se kada je napravljen površinski sloj (engl. Surface). U ovoj funkciji se poziva *on_surface_created()* izvorna metoda koja poziva istoimenu funkciju u JNI sloju.
- *public void onSurfaceChanged(GL10 gl, int width, int height)* – poziva se kada se dođe do promene veličine površinskog sloja. U ovoj funkciji se poziva *on_surface_changed* izvorna metoda koja poziva istoimenu funkciju u JNI sloju.
- *public void onDrawFrame(GL10 gl)* – poziva se kada je potrebno iscrtati okvir u površinskom sloju. U ovoj funkciji se poziva *on_surface_created* izvorna metoda koja poziva istoimenu funkciju u JNI sloju.

Funkcionalnost klase se zasniva na sledećim dinamičkim bibliotekama: libpng.so, libjpeg.so i opengl_renderer.so, generisanim uz pomoć Android NDK alata i komande *ndk_build*. One omogućavaju da se u JNI sloju koriste libpng i libjpeg biblioteke za rad sa grafikom u formatu png i jpeg, kao i biblioteka za iscrtavanje grafike.

Sledeće izvorne metode klase GraphicsRenderer se oslanjaju na JNI sloj:

- *private statice void on_surface_created();*

- *private static native void on_surface_changed(intwidth,intheight);*

  Parametri:

  - int width – širina nove veličine GLSurfaceView komponente
  - int height – visina nove veličine GLSurfaceView komponente

- *private static native void on_draw_frame();*

4.1.3 AudioRenderer

AudioRenderer klasa ima zadatak da rukuje AudioRendering JNI modulom, tako što inicijalizuje elemente neophodne za funkcionisanje Androidove ugrađene izvorne biblioteke OpenSL ES. Zvučne efekte šalje RVU poslužilac ka RVU korisniku koji ih prima i u vidu bafera prosleđuje do JNI sloja, gde se reprodukuju.

Klasa sadrži sledeće metode:

- *CreateEngine()* – metoda koja služi za izgradnju mehanizma neophodnog za funkcionisanje specijalne verzije OpenSL biblioteke, prilagođene Android platformi, a nazvane OpenSL ES verzije 1.1. Ova metoda poziva izvornu metodu *createEngine();*
- *CreateAudioPlayer* – metoda čiji je zadatak izgradnja funkcionalnog modula za reprodukciju audio sadržaja (engl. Audio player). Poziva izvornu metodu: *createAudioPlayer.*
- *ShutdownAudio()* – metoda koja izvršava deinicijalizaiju komponenata neophodnih za reprodukciju audio sadržaja OpenSL ES biblioteke, kao i brisanje napravljenih objekata i oslobađanje zauzete memorije. Poziva izvornu *shutdown_audio* metodu.

Sledeće izvorne metode se oslanjaju na JNI audio biblioteku:

- *public static native void createEngine();*
- *public static native void createAudioPlayer();*
- *public static native void shutdownAudio ();*

4.1.4 RVUClient

RVUClient klasa ima zadatak da inicijalizuje RVU biblioteku i deinicijalizuje je, da omogući propagiranje korisničkih događaja primljenih u okviru modula 4.1.1, kao i primanje događaja i podataka od funkcija sa povratnim pozivima iz JNI sloja. Događaji se prosleđuju do JNI sloja, odakle se dalje prenose RVU biblioteci koja obaveštava RVU poslužioca o korisničkoj akciji.

Ova klasa sadrži sledeće metode:

- *int RvuInit()* – metoda koja inicijalizuje RVU biblioteku i pokreće esencijalne mehanizme neophodne za funkcionisanje biblioteke. Poziva *rvuInit* izvornu metodu.

- *int RvuDeinit()* - metoda koja deinicijalizuje RVU biblioteku i uništava mehanizme neophodne za funkcionisanje biblioteke. Poziva *rvuDeinit* izvornu metodu.
- *void RvuSendScreenRes(int widht, int height)* – metoda koja dobavlja aktuelnu rezoluciju uređaja na kome je aplikacija pokrenuta, u cilju obaveštavanja RVU poslužioca, koji u odnosu na rezoluciju, prilagođava crtanje grafičkih komponenata u *GraphicsRendering* JNI modulu. Poziva sledeću izvornu metodu: *rvuSendScreenRes.*
- *void RvuSendMessage(String message, int key_code, String event)* – metoda koja RVU biblioteci prosleđuje informacije o preduzetoj korisničkoj akciji. Nakon toga, informacije se prenose do RVU poslužioca koji preduzima odgovarajuću akciju.
- Poziva izvornu metodu*RvuSendMessage.*
- *void handleEvent(int event, int value)* – metoda koja se poziva iz JNI sloja u cilju prosleđivanja DMR događaja ka Java aplikacionom sloju. Pristigli događaji se prosleđuju GStreamer komponenti.

  Događaj može biti (vrednost parametra *event*):
  - DMR_EVENT_PLAY
  - DMR_EVENT_STOP
  - DMR_EVENT_PAUSE
  - DMR_EVENT_RESUME
  - DMR_EVENT_BYTE_SEEK
  - DMR_EVENT_TIME_SEEK
  - DMR_EVENT_SET_VOLUME
  - DMR_EVENT_SET_MUTE

Sledeće izvorne metode se oslanjaju na JNI RVU biblioteku:

- *public static native int rvu_init();*
- *public static native int rvu_deinit();*
- *public static native int rvu_send_screen_res(int width, int height);*

  Parametri:
  - int width - širina rezolucije zaslona Android uređaja
  - int height - visina rezolucije zaslona Android uređaja
- *public static native void rvu_send_message(String message, int key_code, String event);*

  Parametri:
  - String message – tip poruke koja se šalje (HDMIKeyEvent, CDIKeyEvent)
  - int key – identifikator operacije (tabele 3.1 – 3.4)

- String event – informacija koji događaj je u pitanju (pritisak ili otpuštanje)

4.1.5 GStreamerPlayer

GStreamerPlayer klasa ima zadatak da rukuje GStreamer multimedijalnom komponentom i da vrši reprodukciju AV sadržaja prosleđenog od strane HTTP poslužioca smeštenog u okviru RVU poslužioca.

Ukoliko korisnik odabere AV sadržaj smešten u okviru određenog DMS poslužioca, koordinate događaja se preko metode *RvuSendMessage* prosleđuju RVU-RUI poslužiocu, koji uz pomoć DMC utvrđuje o kom AV sadržaju se radi. Nakon toga, RVU-RUI poslužilac preko DMC modula, dostavlja RVU korisniku i njegovom modulu DMR (4.3.2.3) URL multimedijalnog sadržaja koji se propagira do ove komponente koja prikazuje AV sadržaj u okviru SurfaceView ugrađene Androidove komponente. Realizacija logike prikaza u SurfaceView komponenti, odrađena je u podklasi klase GStreamerPlayer – GStreamerSurfaceView.

GStreamer predstavlja biblioteku otvorenog koda, realizovanu u C programskom jeziku i moguće ju je kontrolisati iz ove klase pozivima JNI funkcija.

Osnovne metode za rukovanje GStreamer komponentom su:

- *public void Init()* – gradi protočnu strukturu i vrši inicijalizaciju GStreamer izvorne biblioteke preko izvorne metode *nativeInit.*
- *public void Finalize()* – uništava protočnu strukturu pozivom izvorne metode *nativeFinalize.*
- *public void Play()* – metoda koja poziva izvornu metodu *nativePlay()* i pokreće reprodukciju AV sadržaja. Poziva se nakon metode *setMediaUri*koja podešava URI sa kog je potrebno pokrenuti reprodukciju
- *public voidsetMediaUri(String uri)*-metoda kojoj se prosleđuje URI AV sadržaja koji treba reprodukovati. Poziva izvornu metodu *void nativeSetUri.*
- *public void Stop()* – metoda koja prekida reprodukciju i podešava protočnu strukturu na NULL. Poziva izvornu metodu *nativeStop.*
- *void Pause()* – metoda koja pauzira reprodukciju i podešava protočnu strukturu na stanje pauziranosti.

Sledeće izvorne metode se oslanjaju na JNI Gstreamer sprežne funkcije:

- *void nativeInit();*
- *void nativeFinalize();*
- *void nativeSetUri(String mediaUri);*

Parametri:

- String mediaUri – putanja do sadržaja koji je potrebno reprodukovati

- *void nativeStop();*
- *void nativePause();*

4.2 Prilagodni sloj – Java izvorna sprega

Java izvorna sprega predstavlja sloj preko koga se obavlja komunikacija između Java aplikativnog sloja i izvornih aplikacija i biblioteka pisanih u C, C++ i asembleru. U našem slučaju, Java aplikativni sloj samo rukuje izvornim bibliotekama, dok je cela logika, osim reprodukcije AV sadržaja, napisana u C programskom jeziku. Pod logikom se podrazumeva ceo koncept iscrtavanja grafike, reprodukcije zvuka, kao i propagiranje korisničkih akcija iz glavne UI niti aplikativnog sloja, pa do C biblioteka, na koje se oslanja ovaj sloj.

Osnovni gradivni blokovi JNI sloja, predstavljeni su na slici 3.1 i oni su:

- RVU JNI sprežne funkcije
- GStreamer JNI sprežne funkcije
- GraphicsRendering
- AudioRendering

4.2.1 RVU JNI sprežne funkcije

Ovaj modul sadrži funkcije koje su izložene Java aplikativnom sloju na korišćenje. One apstrahuju funkcije za rukovanje komponentama za iscrtavanje grafike i reprodukciju zvučnih efekata kao i RVU bibliotekom, koja se nalazi u sloju ispod.

Kada se u Java aplikativnom sloju, npr. desi promena veličine površine, poziva se metoda *onSurfaceChanged(GL10 gl,int width, int height)*, koja dalje poziva izvornu metodu *native void on_surface_changed(int width, int height);*

Izvorne metode imaju svoje odgovarajuće JNI sprežne funkcije u JNI sloju. Metoda *nativevoidon_surface_changed(int width, int height)* poziva JNI sprežnu funkciju:

*void JNICALL Java_com_amuse_rvuclient_RendererWrapper_on_1surface_1changed (JNIEnv * env, jclass cls, jint width, jint height)*

koja se obraća modulu GraphicsRendering preko funkcije *on_surface_changed(int width, int height);*

Prikaz veza između modula koji sadrži sprežne funkcije i ostatka sistema, predstavljen je na slici 3.2.

4.2.2 GStreamer JNI sprežne funkcije

GStreamer JNI modul sadrži sprežne funkcije preko kojih se GStreamerPlayer modul Java aplikacionog sloja obraća GStreamer izvornoj biblioteci u cilju propagiranja poziva funkcija koje inicijalizuju i deinicijalizuju biblioteku, pokreću, pauziraju ili stopiraju reprodukciju, kao i podešavaju putanju do multimedijalnog sadržaja koji je potrebno reprodukovati.

4.2.3 GraphicsRendering

GraphicsRendering predstavlja modul u kome se vrši iscrtavanje grafičkih elemenata pristiglih iz izvorne RVU biblioteke, u cilju izgradnje potpuno funkcionalne grafičke korisničke sprege. Koncept korišćenja OpenGL ES izvorne biblioteke, prezentovan je u [14].

Grafičke elemente šalje RVU-RUI poslužilac preko TCP kanala namenjenih za podatke, prima ih RVU-RUI korisnik, pakuje u bafere i dostavlja JNI sloju u cilju iscrtavanja koje se vršipo GLSurfaceView komponenti, inicijalizovanoj u okviru Java aplikativnog sloja. Sastoji se iz sledećih podmodula:

- **asset_util** – podmodul koji je zadužen za učitavanje grafičkih programa (engl. shaders) iz direktorijuma /asset koji se prilikom prevođenja aplikacije smešta u .apk datoteku. Za učitavanje datoteka iz /assets direktorijuma, koriste se funkcije deklarisane u android/asset_manager_jni.h zaglavlju (engl. header).

  Najvažnije funkcije ovog podmodula su:

  - *void build_program_from_assets(char *vertex_shader, char * fragment_shader)* – funkcija koja učitava grafičke programe iz /assets direktorijuma i generiše program na osnovu učitanih grafičkih programa

    Parametri:

    - char *vertex_shader – putanja do vertex shader-a koji se učitava iz /assets direktorijuma - /assets/shaders/shader.vsh
    - char * fragment_shader – putanja do fragment shader-a koji se učitava iz /assets direktorijuma - /assets/shaders/shader.fsh

  - *GLuint load_png_buffer_into_texture(uint8_t* buffer, uint32_t size)* – funkcija koja uzima bafer koji sadrži grafičku komponentu u png formatu, pretvara ga u sirove podatke pozivom funkcije *get_raw_image_data_from_png*, a zatim od njih pravi teksturu

    Parametri:

    - uint8_t* buffer – bafer koji sadrži grafičku komponentu u png formatu pristiglu od RVU poslužioca

- uint32_t size – veličina pristiglog bafera

- *GLuint load_jpeg_buffer_into_texture(uint8_t* buffer, uint32_t size)* - funkcija koja uzima bafer koji sadrži grafičku komponentu u jpg formatu, pretvara ga u sirove podatke pozivom funkcije *get_raw_image_data_from_jpeg*, a zatim od njih pravi teksturu

  Parametri:
  - uint8_t* buffer – bafer koji sadrži grafičku komponentu u jpeg formatu pristiglu od RVU poslužioca
  - uint32_t size – veličina pristiglog bafera

- **VBO** – podmodul čiji zadatak predstavlja kreiranje bafera koji skladišti temena prosleđenog mnogougla (poligona).

  Najvažnija funkcija ovog podmodula je:
  - *GLuint create_vbo(const GLsizeiptr size, const GLvoid* data, const GLenum usage)* – funkcija koja pravi objekat koji sadrži tačke poligona (engl. vertex buffer object).

    Parametri:
    - GLsizeiptr size – veličina niza koji sadrži temena poligona
    - GLvoid* data – pokazivač na niz koji sadrži temena poligona
    - GLenum usage –šablon korišćenja podataka. Moguće vrednosti su: STREAM_DRAW,GL_STREAM_READ, GL_STREAM_COPY, GL_STATIC_DRAW, GL_STATIC_READ, GL_STATIC_COPY, GL_DYNAMIC_DRAW, GL_DYNAMIC_READ ili GL_DYNAMIC_COPY.
- **Image** – podmodul koji omogućava dekodovanje pristiglog grafičkog elementa iz enkodovanog oblika png ili jpeg u sirovi oblik i kreiranje OpenGL teksture. Za skladištenje dekodovane slike, napravljena je struktura RawImageData:

```
struct RawImageData {
    const int width;
  const int height;
  const int size;
  const GLenum gl_color_format;
  const void* data;
};
```

Width i *height* predstavljaju širinu i visinu slike, *size* je veličina dekodovane slike, gl_color_format - GL_LUMINANCE, GL_RGBA ili GL_LUMINANCE_ALPHA, dok *data* predstavlja bafer dekodovanih podataka.

- *RawImageData get_raw_image_data_from_png(const void* png_data,const int png_data_size)* – funkcija koja izdvaja sirove podatke iz png grafičke komponente, kroz proces dekodovanja.

  Parametri:
  - const void* png_data – bafer koji predstavlja grafičku komponentu u png formatu
  - int png_data_size – dužina bafera

  Povratna vrednost:
  - RawImageData – struktura koja sadrži informacije o dekodovanoj grafičkoj komponenti, kao i same dekodovane podatke
- *RawImageData get_raw_image_data_from_jpeg(const void* png_data,const int png_data_size)* - funkcija koja izdvaja sirove podatke iz jpg grafičke komponente, kroz proces dekodovanja

  Parametri:
  - const void* png_data – bafer koji predstavlja grafičku komponentu u jpeg formatu
  - int png_data_size – dužina bafera

  Povratna vrednost:
  - RawImageData – struktura koja sadrži informacije o dekodovanoj grafičkoj komponenti, kao i same dekodovane podatke
- *void release_raw_image_data(const RawImageData* data)* – funkcija koja oslobađa sirove podatke nakon što je kreirana tekstura u cilju oslobađanja memorije.

  Parametri:
  - RawImageData * data - struktura koja sadrži informacije o dekodovanoj grafičkoj komponenti, kao i dekodovane podatke

Za iscrtavanje se koristi OpenGL ES 2.0 ugrađena Androidova izvorna biblioteka. Na početku izvršavanja aplikacije, u toku kreiranja GLSurfaceView površine, poziva se *on_surface_created* funkcija u kojoj se briše sadržaj površine pozivom ugrađene OpenGL ES funkcije *glClearColor*. Nakon toga, kreira se VBO (engl. Vertex buffer object) pozivom

funkcije *GLuint create_vbo* kojoj se prosleđuju inicijalne koordinate poligona koji će biti iscrtan. Na ovaj način, temena poligona se čuvaju u video memoriji grafičke kartice, a ne u sistemskoj radnoj memoriji, što značajno poboljšava performanse iscrtavanja. Temena se mogu odmah koristiti za iscrtavanje od strane grafičke kartice, bez potrebe za prenosom podataka između dva tipa memorije.

Kada je bafer napravljen, učitavaju se programi namenjeni za izvršavanje na grafičkoj kartici (engl. Shaders) pozivom funkcije *build_program_from_assets*, kojoj se prosleđuju putanje do navedenih grafičkih programa smeštenih u direktorijumu /assets/shaders. Navedena funkcija poziva funkciju GLuint *build_program*, kojoj se prosleđuju izvorni kodovi grafičkih programa (engl. shader) i njihove dužine. U grafičke programe spadaju verteks grafički programi i fragment grafički programi, gde verteks grafički programi obrađuju temena poligona u smislu rukovanja položajem, koordinatama tekstura, bojom itd, a fragment grafički programi predstavljaju naredni korak u protočnoj strukturi OpenGL ES 2.0 (engl.pipeline) i kao izlaz daju boju koja se nanosi na poligone. Funkcija *build_program* prevodi grafičke programe i povezuje ih.

Kada su grafički programi spremni, potrebno je dobaviti lokacije njihovih atributa i uniformnih promenjivih. To se radi preko OpenGL funkcija *GLint glGetAttribLocation(GLuint program, const GLchar *name)* i *GLint glGetUniformLocation(GLuint program, const GLchar *name).*

Nakon podešavanja inicijalne konfiguracije i komponenata OpenGL-a, korisnička programska podrška je spremna da prihvati grafičke elemente i iscrta ih u okviru grafičke korisničke sprege. Kada RVU poslužilac pošalje grafički element, on se smešta u bafer u okviru RVU biblioteke i šalje do GraphicsRendering modula. Grafičke poruke se smeštaju u red (engl. FIFO – First-in-First-out) iz razloga što je prihvatna nit iz RVU biblioteke, dok se renderovanje vrši u drugoj niti.

Grafička komponenta se uzima iz reda, i ukoliko RVU biblioteka zahteva dekodovanje grafičkih podataka u jpeg formatu, poziva se *load_jpeg_buffer_into_texture* dok ukoliko je png format, poziva se *load_png_buffer_into_texture. load_jpeg_buffer_into_texture koristi* libjpeg i *preko get_raw_image_data_from_png* funkcije, dekoduje podatke iz jpeg formata u sirove podatke. Nakon toga se poziva funkcija *load_texture* koja koristeći OpenGL ES otprema teksturu u video memoriju uz pomoć ugrađene funkcije *glTexImage2D.*

Kada je tekstura spremna, iz RVU biblioteke se dobavljaju koordinate na kojima je potrebno iscrtati tu teksturu i to se obavlja u funkciji *void draw_texture(float x, float y, int w, int h).* Koordinate se normalizuju u OpenGL ES prostor i upisuju u translacionu matricu. Ova matrica omogućuje verteks grafičkom programu da prosledi nova temena koja označavaju

poziciju na kojoj treba iscrtati dobavljenu teksturu. Kada je matrica podešena, ona se zajedno sa ostalim uniformnim lokacijama i atributima u funkciji *on_draw_frame* prosleđuje verteks grafičkom programu koji izračunava nove koordinate na koje će se tekstura iscrtati.

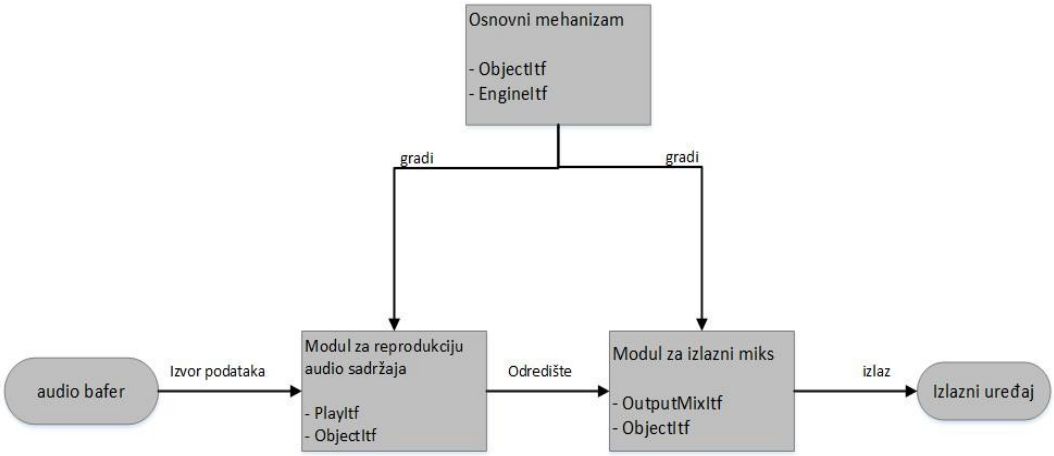
U slučaju da je potrebno ispuniti region određenom bojom, što takođe spada u RVU podržane akcije iscrtavanja, koordinate regiona se dobavljaju iz RVU biblioteke i poziva se *void draw_rect(float x, float y, int w, int h, int color)* funkcija. Ona podešava translacionu matricu i šalje grafičkim programima koordinate na kojima treba obojiti region, kao i boju kojom se region želi ispuniti.

Ukoliko dođe do promene dimenzija GLSurfaceView površine, neophodno je o tome obavestiti RVU poslužioca, u cilju prilagođenja grafičke korisničke sprege novoj veličini površine. To se radi tako što GLSurfaceView u Java aplikativnom sloju preko metode *onSurfaceChanged*, poziva izvornu metodu *native void on_surface_changed* koja se propagira do GraphicsRendering izvornog modula gde se u funkciji *void on_surface_changed*poziva funkcija RVU biblioteke int *send_res(int width,int height)* i dalje obaveštava RVU poslužilac preko komandnog kanala i poruke OutputSettingsChanged (tabela 3.9). Osim toga, promenjive koje sadrže veličinu površine se menjaju, u cilju korektne normalizacije koordinata iz standardnog koordinatnog sistema u OpenGL koordinatni sistem (od -1 do 1) i korektnog smeštanja grafičkih komponenata u grafičku ravan.

4.2.4 AudioRendering

AudioRendering predstavlja modul koji koristi funkcionalnosti ugrađene OpenSL ES biblioteke za reprodukciju zvučnih komponenata koji poboljšavaju opšti utisak korišćenja aplikacije.

Slanju zvučnih efekata prethodi konfiguracija RVU korisnika od strane RVU poslužioca u cilju zauzimanja bafera i saznavanja karakteristika korisničkog uređaja. Audio komponente se šalju od RVU-RUI poslužioca ka RVU-RUI korisniku, koji ih prima, smešta u bafere i mehanizmom funkcija sa povratnim pozivom, prosleđuje do ovog modula gde se vrši reprodukcija.



Slika 4.1 Struktura modula za reprodukciju audio sadržaja

Prilikom pokretanja aplikacije, pozivaju se metode opisane u poglavlju 4.1.3, koje se preslikavaju na JNI sprežne funkcije i pozivaju funkcije ovog modula. U te funkcije spadaju: void *create_engine* i *create _audio_player* i njihova funkcionalnost je opisana u poglavlju 4.1.3.

Funkcija void *create_engine* (blok „Osnovni mehanizmi" na slici 4.1) kreira mehanizam (engl. engine) pomoću OpenSL ES ugrađene funkcije *slCreateEngine*, realizuje mehanizam, dobavi spregu mehanizma i preko njega kreira ostale objekte poput izlaznog miksa.

Druga funkcija koja se poziva u fazi inicijalizacije, jeste *void create_audio_player* (blok „Modul za reprodukciju audio sadržaja" na slici 4.1) koja inicijalizuje i podešava izvor i odredište zvuka, kreira modul za reprodukciju zvuka (engl. audio player), realizuje ga i podešava dodatne objekte poput reda za smeštanje bafera i statusa reprodukcije.

Kada se desi događaj koji treba da bude propraćen zvučnim efektom, poziva se funkcija *int play_sound(char *buffer, int filesize)* kojoj se kao prvi parametar prosleđuje bafer koji sadrži zvučni efekat, dok drugi parametar predstavlja dužina bafera. Na ovaj način se vrši reprodukcija zvučnih efekata.

4.3 Izvorne biblioteke

Izvorne biblioteke predstavljaju biblioteke napisane u izvornim jezicima (C, C++, asembler), sa ciljem da obezbede osnovnu funkcionalnost RVU korisničke programske podrške, u vidu komunikacije sa RVU poslužiocem, realizacije DLNA DMR uređaja, AV reprodukcije i rukovanja grafičkim elementima.

Biblioteke u ovom modulu poseduju svojstvo prenosivosti i moguće ih je iskoristiti za realizaciju RVU korisničke sprege na platformama poput iOS, Windows Phone, Linuks itd.

U ove biblioteke spadaju:

- RVU biblioteka
- libpng
- libjpeg
- libupnp
- GStreamer

4.3.1 Libpng i libjpeg

Ove biblioteke otvorenog koda su iskorišćene u cilju omogućavanja dekodovanja grafičkih elemenata neophodnih za izgradnju funkcionalne grafičke korisničke sprege.

Grafičke elemente RVU-RUI korisniku dostavlja RVU-RUI poslužilac u formatu png ili jpeg iz razloga što prenos sirovih podataka preko mreže unosi kašnjenja zbog velike količine informacija koja se šalje.

Libpng i libjpeg su iskorišćene u cilju ostvarivanja olakšane prenosivosti na druge platforme. Svaka platforma ima svoj mehanizam preko kojeg vrši dekodovanje, što bi zahtevalo ponovnu realizaciju cele logike. Na ovaj način, postoji jedinstveni princip dekodovanja upotrebljiv na raznim platformama poput Linuksa, iOS-a, Windows Phone-a itd.

4.3.2 RVU biblioteka

U ovom modulu sadržana je osnovna funkcionalnost RVU programske podrške. Implementirana je biblioteka isključivo u programskom jeziku C, što omogućava prenosivost na druge platforme poput Linuksa, iOS, Windows Phone itd. Osnovni gradivni blokovi predstavljeni su na slici: 3.4.

4.3.2.1 Aplikativna programska sprega

Aplikativna programska sprega izlaže funkcionalnosti RVU biblioteke i omogućava njeno korišćenje i interakciju sa drugim modulima. RVU biblioteka se sastoji iz dva modula, a to su: RVU-RUI korisnik i DLNA DMR, ako što je prikazano na slici 3.3. Modul aplikativna programska sprega apstrahuje ove module i sastoji se iz sledećih funkcija koje kao povratne vrednosti vraćaju neki od članova sledeće enumeracije:

```
typedef enum {
    RVU_CLIENT_API_SUCCESS,
    RVU_CLIENT_INIT_ERROR,
    RVU_CLIENT_DMR_ERROR,
    RVU_CLIENT_SERVER_ERROR,
    RVU_CLIENT_NO_DEVICES_FOUND,
    RVU_CLIENT_RUIC_ERROR,
    RVU_CLIENT_BAD_PARAMETER_ERROR
} RVU_Client_API_Error;
```

- *RVU_CLIENT_API_SUCCESS* – funkcija izvršena bez grešaka
- *RVU_CLIENT_INIT_ERROR* – greška prilikom inicijalizacije. Mogući razlozi su nedostupnost mreže ili višestruka inicijalizacija
- *RVU_CLIENT_DMR_ERROR* – greška prilikom kreiranja ili startovanja DMR uređaja
- *RVU_CLIENT_SERVER_ERROR* – greška prilikom startovanja komponente za prijem poruka
- *RVU_CLIENT_NO_DEVICES_FOUND* – obaveštenje da nijedan RVU kompatabilan poslužilac nije pronađen
- *RVU_CLIENT_RUIC_ERROR* – greška prilikom kreiranja i pokretanja UPnP kontrolne tačke, nazvane RUIC prema RVU specifikaciji
- *RVU_CLIENT_BAD_PARAMETER_ERROR* – indikator da određenoj funkciji nije prosleđen korektan parametar

Funkcije aplikativnog programske sprege su:

- *RVU_Client_API_Error RVU_init(void)* - funkcija koja inicijalizuje UPnP biblioteku pozivom funkcije *dlna_init*. I RVU-RUI korisnik i DLNA DMR koriste UPnP, tako da je inicijalizacija smeštena u najviši sloj biblioteke
- *RVU_Client_API_Error RVU_DMR_start(dlna_callback_t)* – funkcija koja podešava parametre DMR uređaja pozivom funkcije *dlna_dev_set_prop,* kreira ga (*dlna_dmr_create*) , pokreće*dlna_dmr_start*. i registruje funkciju sa povratnom vrednošću *dlna_error_t dlna_event_handler (dlna_event_type_t, dlna_event_value_t *, void *)* preko funkcije *dlna_set_event_cb*.

  U parametre spadaju:
  - *dlna_dev_manufacturer*
  - *dlna_dev_manufacturer_url*
  - *dlna_dev_model_name*
  - *dlna_dev_model_number*
  - *dlna_dev_model_url*
  - *dlna_dev_model_desc*
  - *dlna_dev_frandly_name*

  Parametar ove funkcije je pokazivač na funkciju preko koga je implementirana logika poziva funkcija sa povratnom vrednošću. Preko ovih funkcija se događaji propagiraju do JNI sloja, gde se dalje prosleđuju u Java aplikativni sloj korisniku AV sadržaja.

*typedef void (*dlna_callback_t) (dlna_event_type_t, dlna_event_value_t *, void *);* koji sadrži tip događaja, vrednost događaja i dodatnu vrednost ukoliko je potrebna.

- *RVU_Client_API_Error RVU_client_start(ruic_callback_t*) – funkcija koja registruje funkciju sa povratnom vrednošću *ruic_event_handler(RVU_Client_Notification, GraphicMessage *)* preko funkcije *ruic_set_event_cb*. i inicijalizje module za rukovanje baferima, kanalima i porukama, kao i modula za primanje i slanje TCP poruka.

  Parametar ove funkcije je pokazivač na funkciju preko koga je implementirana logika poziva funkcija sa povratnom vrednošću. Preko ovih funkcija se događaji propagiraju do JNI sloja, gde se dalje obrađuju.

  *typedef void (*graphics_callback_t)(RVU_Graphic_event event, GraphicMessage *graphicMessage)* – gde je prvi parametar enumeracija:

```
typedef enum {
    HELLO_RECEIVED_INIT,
    COPYBLIT_OPERATION,
    FILLBLIT_OPERATION,
    RESIZEBLIT_OPERATION,
    TEXTBLIT_OPERATION
} RVU_Graphic_event;
```

  Drugi parametar je struktura koja sadrži audio ili grafički bafer, kao i informacije o tom baferu.

- *RVU_Client_API_Error RVU_deinit()* – funkcija koja deinicijalizuje UPnP, zaustavlja DMR i RUI i deinicijalizuje module RVU biblioteke.

4.3.2.2 RVU-RUI korisnik

RVU-RUI korisnik predstavlja najvažniji element RVU korisničke programske sprege i njegova struktura je predstavljena na slici 3.5.

RUIC predstavlja modul RVU-RUI korisnika zadužen za uspostavljanje veze između korisnika i poslužioca. Ovaj modul vrši pretragu dostupnih poslužilaca u cilju pronalaženja onih koji poseduju kompatabilan RUI, uspostavljanje veze sa njima i razmenu osnovnih podataka predviđenih RVU specifikacijom.

Jedan korisnik ne može biti povezan sa više od jednog poslužioca. Ukoliko je pronađeno više kompatibilnih poslužilaca, korisnik može da izabere sa kojim će uspostaviti vezu.

Na samom početku, iz JNI sloja se poziva funkcija *RVU_init()* opisana u poglavlju 4.3.2.1. Na taj način se inicijalizuje UPnP biblioteka i omogućava korišćenje njenih funkcionalnosti.

U okviru RUIC funkcije int *RUI_ctrl_point_start(),* poziva se UPnP ugrađena funkcija koja registruje UPnP kontrolnu tačku i šalje *UPNP_EVENT_MSEARCH_CHECK_REQUEST* poruku:

- *int UpnpRegisterClient (UPnP_FunPtr Callback, const void * Cookie, UpnpClient_Handle * Hnd).*

Kao prvi parametar funkcije prosleđuje se pokazivač na funkciju u kojoj se rukuje sa asinhronim događajima nastalim u toku interakcije sa drugim UPnP kompatibilnim uređajima. U našem slučaju, funkciji se prosleđuje rukovaoc događajima *event_handler(UPnP_EventType EventType, void *Event, void *Cookie)* koji prima navedene asinhrone događaje. Drugi parametar predstavlja pokazivač na korisničke podatke dobavljene kada je pozvana funkcija sa povratnim pozivom. Treći parametar predstavlja pokazivač na promenljivu u koju će biti smešten rukovalac registrovanom kontrolnom tačkom.

Nakon registrovanja kontrolne tačke, vrši se brisanje svih poslužilaca koji su smešteni u jednostruko-spregnutu listu povezanih uređaja. Kada je lista prazna, poziva se ugrađena UPnP funkcija *UpnpSearchAsync* koja traži poslužioce sledećeg tipa: *"urn:schemas-UPnP-org:device:MediaServer:1".*

Ukoliko je na mreži prisutan poslužilac navedenog tipa, funkcija *event_handler* prima događaj *UPNP_DISCOVERY_ADVERTISEMENT_ALIVE*, a zatim i *UPNP_DISCOVERY_SEARCH_RESULT* uz koji stiže i struktura struct *UPnP_Discovery* koja sadrži IP adresu poslužioca, osnovne informacije o mrežnoj lokaciji dokumenta koji opisuje poslužioca, vreme isticanja objave prisutnosti, kao i informacije o grešci.

Ako je odgovor stigao od strane poslužioca tipa: "urn:schemas-UPnP-org:device:MediaServer:1", i stanje greške je *UPNP_E_SUCCESS*, kreira se posebna ITHREAD nit kojoj se prosleđuje strukturaUpnp_Discovery.Na početku izvršavanja niti, iz strukture se izdvaja polje d_event->Location i prosleđuje UPnP funkciji *int UpnpDownloadXmlDoc(const char * url, IXML_Document ** xmlDoc)* koja preuzima XML datoteku koja opisuje poslužioca. Nakon toga, dokument se prosleđuje funkciji *add_device* koja iz XML dokumenta izdvaja osnovne informacije o poslužiocu.

Najvažniji elementi koja se izdvajaju iz XML dokumenta su:

- UDN – jedinstveni identifikacioni broj uređaja, npr "uuid:0622707c-da97-4286-5433-001ff3590148"

koji se sastoji iz vremenskog pečata (vreme i datum) i MAC adrese uređaja (poslednjih 12 cifara)

- deviceType – tip uređaja npr: "urn:schemas-UPnP-org:device:MediaServer:1"
- friendlyName – npr. “AMUSE RVUServer”
- servisi – RVU-RUI kompatabilan uređaj mora da poseduje sledeći servis: "urn:schemas-UPnP-org:service:RemoteUIServer:1";

Nakon izdvajanja elemenata, zauzima se memorija za strukturu TvDevice i popunjavaju se njena polja:

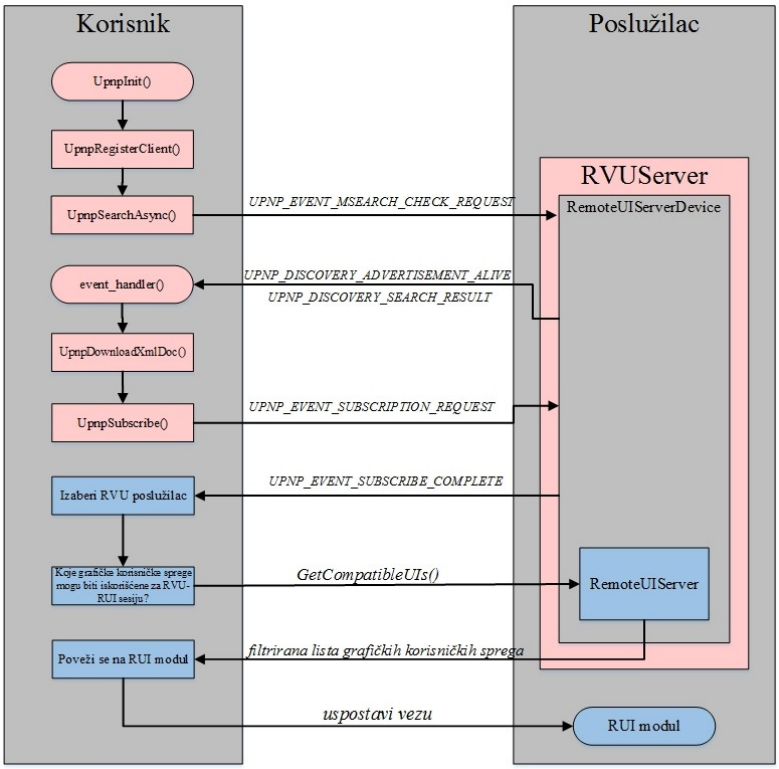
```
struct TvDevice {
    char UDN[NAME_SIZE];
    char DescDocURL[NAME_SIZE];
    char FriendlyName[NAME_SIZE];
    char PresURL[NAME_SIZE];
    int isEmbedded;
    int AdvrTimeOut;
    char IP[16];
    struct DeviceService TvService[TV_SERVICE_SERVCOUNT];
};
```

Kao što je navedeno, RVU-RUI kompatabilan uređaj mora da sadrži "urn:schemas-UPnP-org:service:RemoteUIServer:1" servis. Ukoliko sadrži, preuzima se XML dokument u kome su navedene osnovne osobine servisa [18], izdvajaju se komponente servisa i zapisuju u sledeću strukturu:

```
struct DeviceService {
    char ServiceName[NAME_SIZE];
    char ServiceId[NAME_SIZE];
    char ServiceType[NAME_SIZE];
    char *VariableStrVal[TV_MAXVARS];
    char EventURL[NAME_SIZE];
    char ControlURL[NAME_SIZE];
    char SID[NAME_SIZE];
};
```

Da bi RVU-RUI korisnik utvrdio da li poslužilac poseduje odgovarajući UI, neophodno je poslužiocu poslati *GetCompatibleUIs* poruku u kojoj nalazi spisak UI koje korisnik podržava. Kada RVU-RUI poslužilac primi ovu poruku, on filtrira UI-eve i odgovara sa listingom onih koji su obostrano podržani. Ukoliko RVU-RUI poslužilac ne podržava nijedan UI naveden od strane RVU-RUI korisnika, poslužilac odgovara sa praznim listingom. Sesija započinje ukoliko RUIS podržava bar jedan UI.

Primer uspostavljanja veze od startovanja RVU-RUI poslužioca do odabira grafičke korisničke sprege, prikazan je na slici ispod:



Slika 4.2 RVU-RUI sesija

Naredni XML predstavlja primer UI listinga (InputDeviceProfile) koji se šalje preko *GetCompatibleUIs* poruke:

```
<deviceprofile xmlns=\"urn:schemas-UPnP-org:remoteui:devprofile-1-0\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:schemaLocation=\"urn:schemas-UPnP-org:remoteui:devprofile-1-
0DeviceProfile.xsd\">
<maxHoldUI>0
</maxHoldUI>
<protocol shortName=\"RVU-RUI\">
</protocol>
</deviceprofile>
```

Prethodni xml kod je predstavljen u obliku stringa i prosleđuje se funkciji *send_GetCompatibleUIs* koja sastavlja akciju *GetCompatibleUIs*. Prvi parametar akcije je "InputDeviceProfile" i njemu se prosleđuje navedeni string, dok drugi parametar "UIFilter" ostaje prazan. Na ovaj način će poslužilac poslati sve RVU UI koje podržava.

Pozivom funkcije *UpnpAddToAction*, u poruku koja je na redu za otpremanje se dodaje novi parametar, i kada je poruka kompletirana, šalje se uz pomoć funkcije *UpnpSendAction*. UpnpSendAction predstavlja sinhroni poziv, koji se ne završava dok akcija nije završena. Ova funkcija vraća dokument u kome se nalaze informacije o grafičkim korisničkim spregama koje RVU-RUI poslužilac podržava. Primer takvog dokumenta je prikazan u nastavku rada:

```
<?xml version="1.0" encoding="UTF-8"?>
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schemas-UPnP-org:remoteui:uilist-1-0
CompatibleUIs.xsd">
<ui>
<uiID>1</uiID>
<name>RVU Remote UI</name>
<protocol shortName="RVU-RUI">
<uri>rvurui://[ip]:[port]</uri>
</protocol>
</ui>
</uilist>
```

U ovom primeru [ip] predstavlja ip adresu na kojoj se nalazi RUIS, dok [port] predstavlja prolaz na kome RVU-RUI treba da otvori komunikacioni kanal tj. TCP tok i uspostavi inicijalnu vezu. To se obavlja u modulu za rukovanje TCP tokovima.

TCP korisnik modul omogućava otvaranje komunikacionog kanala od strane RVU-RUI korisnika.

Nakon što se u prethodnom modulu pokrenuo RVU-RUI korisnik, pronašao RVU-RUI poslužioca i dobavio listing, neophodno je otvoriti TCP komunikacioni kanal. Inicijalnu komunikaciju započinje isključivo RVU-RUI korisnik.

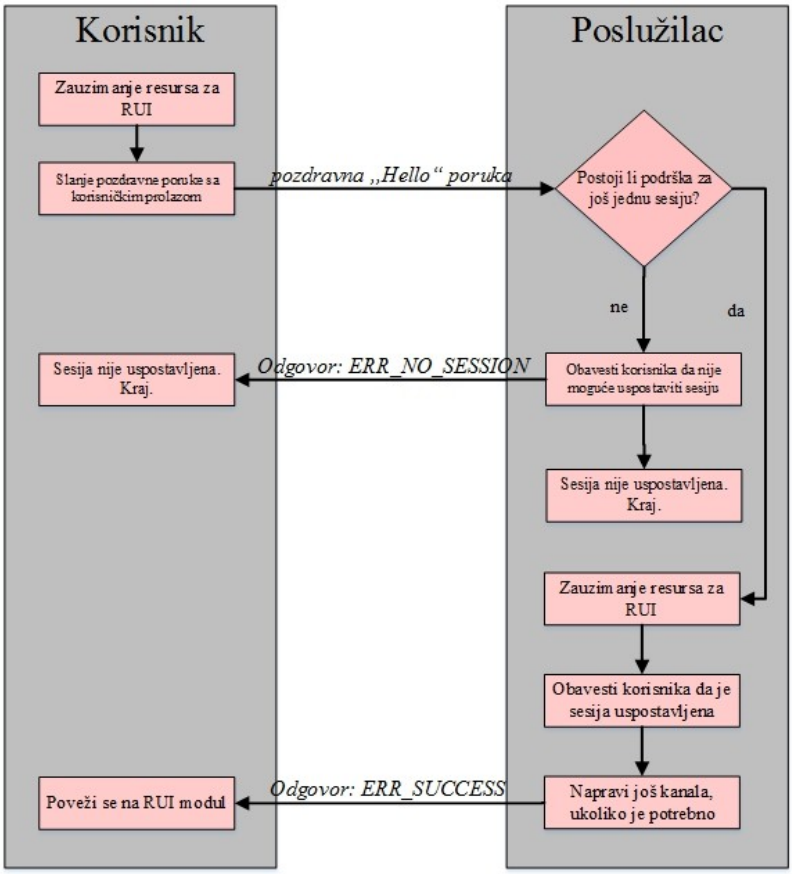
Funkcija koja inicijalizuje ovaj modul *rvu_client_init()* poziva se u okviru inicijalizacione API funkcije – *RVU_init()*. Ona zauzima memoriju za opis TCP poslužioca, čiji broj ne može biti veći od jedan.

Najvažnija funkcija ovog modula je int *rvu_client_connect_to_server(char * hostname, int port, ChannelDesc** channel)* čiji zadatak je, kao što joj ime kaže, uspostavljanje veze sa RUIS. Ona kao prvi parametar prihvata IP adresu. Kao drugi prihvata prolaz na kome će se uspostaviti TCP kanal, dok je treći parametar pokazivač na pokazivač na ChannelDesc strukturu koja će detaljnije biti opisana u modulu za rukovanje kanalima.

Nakon uspostavljanja konekcije, zauzima se novi kanal pozivom funkcije iz modula za rukovanje kanalima: *rvu_channel_allocate* kojoj prosleđujemo objekat strukture ChannelDesc. Kanalu se daje identifikator i poziva se *rvu_channel_start* koji pokreće novu nit koja će da prima odgovore na poruke ne remeteći osnovnu nit. Preko funkcije rvu_client_send_message se šalje pozdravna poruka koja mora biti prva poruka poslata preko novootvorenog kanala. Sledeći XML predstavlja primer pozdravne poruke:

```
<Hello commandToken=”111” channelId=”3” version=”1.0” callbackPort=”9999”/>
```

Sledeći grafik objašnjava logiku otvaranja TCP komunikacionog kanala, u slučaju uspešnog i neuspešnog otvaranja:



Slika 4.3 Otvaranje TCP komunikacionog kanala

Osim otvaranja kanala, ovaj modul sadrži i funkcije za deinicijalizaciju modula *rvu_client_deinit(),* kao i funkcije za zatvaranje kanala *rvu_client_disconnect(ElementDesc*),* podešavanje kanala - *rvu_client_set_channel(ElementDesc*,ChannelDesc*),* uklanjanje kanala - *rvu_client_remove_channel(ChannelDesc*),* dobavljanje kanala po identifikatoru rvu_client_get(uint8_t, ElementDesc**), IP adresi - *rvu_client_get_by_address(in_addr_t, ElementDesc**),* kao i dobavljanje identifikatora kanala prema opisu elementa - *rvu_client_get_channel_number(ElementDesc*, uint8_t*).*

Pored otvaranja komunikacionog kanala, ovaj modul sadrži funkcionalnost za prosleđivanje poruka kroz kanal.

Sve poruke koje se šalju iz Java aplikacionog sloja, preko JNI se propagiraju do ovog modula koji šalje poruke preko TCP kanala. To se radi uz pomoć funkcije:

*int rvu_client_send_message(char *message_name, uint32_t key_code, char *event).*

Parametri:

- char *message_name – predstavlja naziv poruke koja se šalje. Poruke koje šalje RUIC mogu biti „Hello“, „HDMIEvent“ i „CDIEvent“.

- uint32_t key_code – ukoliko je poruka namenjena za prosleđivanje korisničkog događaja. Tabele 3.1-3.3
- char *event – događaj – pritisak ili otpuštanje, predstavljen u sledećoj enumeraciji:

```
typedef enum {
    KEY_EVENT_UP = 0,
    KEY_EVENT_DOWN
} RVU_Client_Key_Event;
```

TCP poslužilac modul ima zadatak da na zahtev RVU-RUI poslužioca prihvati zahtev za otvaranje kanala i prima poruke poslate preko kanala, kao i da odgovara na njih prema RVU specifikaciji.

Funkcija *RVU_init*, članica API-ja RVU biblioteke, poziva funkciju ovog modula *rvu_server_init*koja priprema TCP terminalnu tačku (engl. socket) za prijem zahteva za uspostavljanje konekcije i slušanje na portu dostavljenom u inicijalnoj pozdravnoj poruci. Osim toga, ova funkcija kreira novu nit čiji je zadatak primanje poruka i odgovaranje na iste.

Odgovori na primljene poruke se prosleđuju preko funkcije *rvu_server_send_message(ChannelDesc* channel, uint8_t* message, uint32_t message_size).*

U kreiranoj niti se primaju komandne i podaci poslati od strane RUI-RUI poslužioca. Komandne poruke prethode podacima i sadrže informacije o odredišnim baferima u koje se smeštaju podaci.

Kada RVU-RUI zatraži otvaranje novog kanala, pozivaju se funkcije *rvu_channel_allocate* i *rvu_channel_start* iz modula za rukovanje kanalima, koje implementiraju mehanizam za rukovanje porukama unutar kanala.

Modul za rukovanje kanalima ima zadatak da rukuje kanalima, bilo da su oni komandni ili kanali za prenos podataka, kao i da rukuje redom čekanja u koji se smeštaju poruke. Red čekanja je iskorišćen i za prenos i obradu poruka u različitim nitima, pošto je implementirana zaštita u slučaju višenitnog izvršavanja. Za implementaciju zaštite, iskorišćena je uslovna promenjiva (engl. conditional variable) u kombinaciji sa uzajamno isključivom bravom (engl. mutex) logikom.

Kanal se kreira tako što se poziva funkcija *int rvu_channel_allocate(ChannelDesc** channel)* koja zauzima resurse za novi kanal, ukoliko maksimalni broj kanala nije već zauzet. Nakon toga se poziva *int rvu_channel_start(ChannelDesc* channel)*, funkcija koja pokreće

novu nit u kojoj se poruke preuzimaju iz reda i obrađuju u zavisnosti od statusa BlitQueue poruke.

Ukoliko RVU-RUI poslužilac preko komandnog kanala pošalje BlitQueue poruku, RVU-RUI korisnik ima obavezu da sve naredne primljene poruke smesti u red čekanja, i samim tim odloži njihovo izvršavanje. Smeštanje u red može biti osposobljeno (atribut ima vrednost različitu od 0) i onesposobljeno (atribut ima vrednost 0).

atribut:	Opis:	Tip
commandToken	Jedinstveni identifikator koji predstavlja komandu	uint
queuing	0 – prestanak smeštanja komandi u red i početak obrade skladištenih Različita vrednost od 0: početak smeštanja komandi u red	uint

Tabela 4.1 Atributi BlitQueue komande

RVU korisnik smešta u red sledeće komande: CopyBlit, FillBlit, ResizeBlit, ShadeBlit, BlendBlit, MultiSourceBlendBlit, ResizeAndBlendBlit, ColorKeyResizeBlit, ili WaitVSync komande ukoliko je BlitQueue komanda primljena sa atributom različitim od 0. Smeštanje u red se obavlja pozivom funkcije *int rvu_queue_enqueue(Queue* queue, void* message)*. RVU korisnik poseduje mogućnost smeštanja najmanje 256 komandi i to važi za sve kanale.

Ukoliko je komandni kanal u stanju svrstavanja poruka u red čekanja, to stanje traje sve dok se atribut svrstavanja ne podesi na vrednost 0, ili dok se ne pojavi komanda Dispatch. Komanda Dispatch obavezuje RVU korisnika da momentalno krene sa obradom svrstanih komandi u red čekanja, pritom, ne menjajući stanje atributa za svrstavanje. Preuzimanje komandi iz reda čekanja se obavlja preko funkcije: *rvu_queue_dequeue(Queue* queue, void** message)*.

Pristigle poruke se dele na komandne poruke i podatke. Komandne poruke uvek počinju sa znakom „<“ koji predstavlja početak XML zaglavlja: <?xml version="1.0"?>, dok podaci kreću sa numeričkim znakovima.

Ukoliko je u pitanju poruka koja predstavlja podatke i prevelika je pa ne može da bude primljena odjednom, neophodno je pročitati u zaglavlju koliko je poruka dugačka, a zatim primati segmente poruke i spajati ih.

Sve poruke primaju se u okviru određenog kanala, bilo da je upitanju komandni kanal ili kanal namenjen za primanje podataka, prosleđuju modulu za rukovanje porukama.

Modul za rukovanje porukama ima zadatak da prima poruke poslate iz modula za rukovanje kanalima i da ih raščlanjuje, preduzima određenu akciju i šalje odgovor o uspešnosti izvršavanja akcije.

Modul se inicijalizuje pozivom funkcije *int rvu_message_init()* u funkciji API-ja *RVU_init()*. Deinicijalizuje se pozivom *rvu_message_deinit()* funkcije koja se poziva u API funkciji *RVU_deinit()*.

Odgovor na poruku je predstavljen strukturom MessageResp:

```
struct MessageResp {
    RVU_Message_Error message_error;
char response_for_command[50];
};
```

RVU_Message_Error predstavlja enumeraciju u kojoj su navedene celobrojne vrednosti grešaka do kojih može doći u toku izvršavanja komandi.

Razlikuju se dva slučaja obrade poruke:

- Poruka spremna za obradu – atribut *queuing* ima vrednost 0
- Poruka namenjena za smeštanje u red čekanja – atribut *queuing* ima vrednost različitu od 0

Ukoliko je poruka spremna za obradu u modulu za obradu kanala, a atribut smeštanja u red queuing podešen na 0, poziva se funkcija *int rvu_parse_message(ChannelDesc*, uint8_t*, MessageResp*)* kojoj se prosleđuje opis kanala u okviru koga je primljena poruka, sama poruka predstavljena nizom bajtova, kao i odgovor koji treba poslati RVU poslužiocu.
Na početku funkcije *rvu_parse_message* proverava se prvi bajt poruke. Ukoliko je u pitanju komandna poruka, ona počinje sa znakom „<“ i poruka će biti tretirana kao komandna u daljem nastavku obrade. Ako je komandna poruka pristigla kroz kanal rezervisan za podatke, neophodno je stopirati kanal jer se desio kritični događaj. Stopiranje kanala obavlja funkcija modula za rukovanje kanalima*: int rvu_channel_stop(ChannelDesc* channel).* Nakon toga se, u strukturu koja sadrži grešku koja će biti prosleđena RVU poslužiocu, upisuje *msg_response->message_error = RVU_MESSAGE_INVALID_MESSAGE*, kao obaveštenje da je došlo do greške u kanalu. Isto važi i za podatke koji pristignu preko komandnog kanala.

Ako je poruka poslata kroz odgovarajući kanal, a u pitanju je komandna poruka, poziva se *static uint8_t parse_command(ChannelDesc*, uint8_t*, MessageResp*)* koja pronalazi da li je poruka podržana, raščlanjuje poruku predstavljenu u XML formatu i preduzima određenu akciju.

Za slučaj da je u pitanju poruka koja sadrži podatke, poziva se funkcija *static uint8_t parse_data(ChannelDesc*, uint8_t*, MessageResp*)*koja raščlanjuje podatke i zavisnosti da li se radi o pozdravnoj poruci ili poruci za upis (engl. Write), i preduzima odgovarajuću akciju.

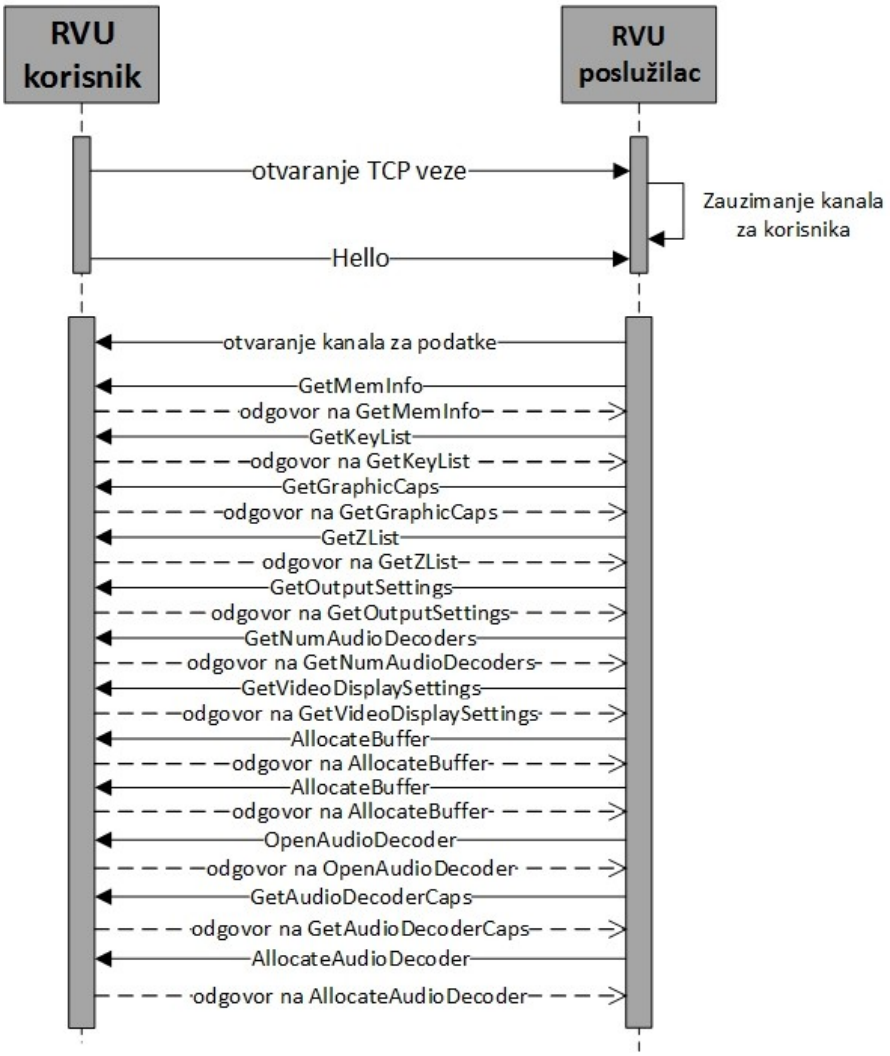
Ako je u pitanju poruka za upis, ona se zapisuje u listu koja sadrži bafere, da bi se posle iz te liste baferi preuzimali i preko funkcija sa povratnim pozivom, prosleđivali do JNI sloja i modula za iscrtavanje ili reprodukciju audia.

Ukoliko je poruka pristigla u modul za obradu kanala, dok je atribut smeštanja u red podešen na vrednost različitu od 0, poziva se funkcija *int rvu_parse_msg_comm_name (ChannelDesc*, uint8_t*, MessageResp*)* kojoj se prosleđuje opis kanala u okviru koga je primljena poruka, sama poruka predstavljena nizom bajtova, kao i odgovor koji treba poslati RVU poslužiocu.

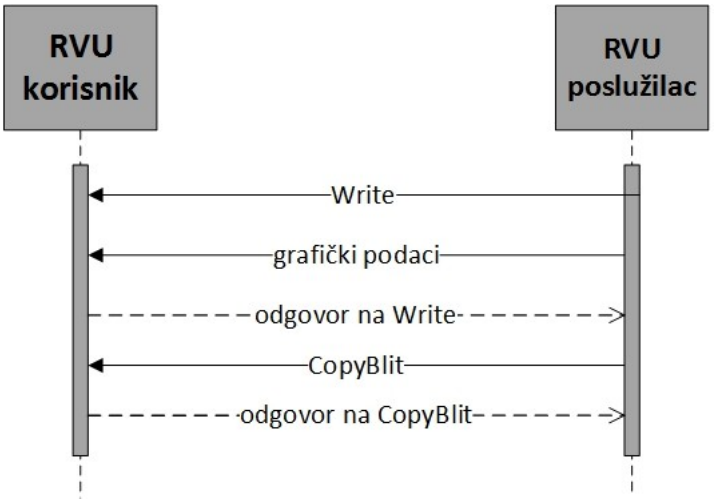
Za razliku od *rvu_parse_message*, ova funkcija ne vrši obradu i raščlanjivanje XML poruke, već to ostavlja za trenutak u kome će se atribut za smeštanje u red promeniti. Kao što je rečeno, obrada poruka smeštenih u red nakon BlitQueue komande započinje nakon Dispatch komande ili ponovne BlitQueue poruke, koja atribut queueing podešava na vrednost 0.

Nakon što je poruka obrađena i raščlanjena, potrebno je RVU poslužiocu poslati odgovor. To se vrši preko poruke *int rvu_send_message_response(ChannelDesc*, MessageResp)* koja sastavlja XML dokument koji se šalje preko kanala u okviru koga je poruka primljena. Na poruke koje nose podatke se ne sastavljaju odgovori.

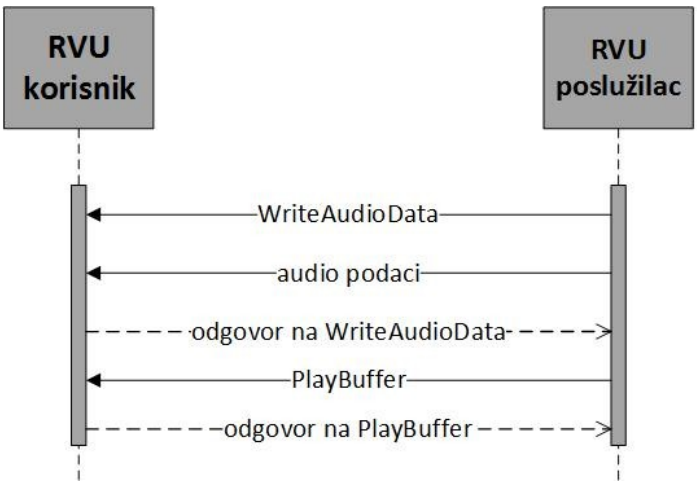
Na sledećem grafiku, predstavljene su poruke koje se koriste u procesu inicijalizacije:



Slika 4.4 Proces inicijalizacije



Slika 4.5 Proces slanja grafičkog elementa



Slika 4.6 Proces slanja audio poruke

Modul za rukovanje baferima predstavlja modul koji rukuje jednostruko spregnutom listom u cilju čuvanja i brzog i jednostavnog pristupa željenim baferima. Baferi mogu biti:

- zaslonski – bafer koji sadrži grafičke komponente trenutno prikazane korisniku
- grafički – pozadinski baferi u koje korisnik nema uvid
- audio – baferi u koje su smešteni zvučni efekti. Ukoliko je određeni bafer sačuvan i predviđen za višestruko korišćenje, moguće ga je reprodukovati više puta prema instrukcijama RVU poslužioca

Element jednostruko spregnute liste predstavljen je strukturom:

```
struct RVUBuffer {
    int write_to;
    int type;
    unsigned bufId;
    uint8_t * buffer;
    int buffer_size;
    char error_code[20];
};
```

Ovaj modul se inicijalizuje kao i svi ostali moduli u API funkciji RVU_init() pozivom funkcije *RVU_buffer_init*.

Bafer je moguće zauzeti preko funkcije: *int RVU_buffer_allocate (unsigned long bufId, int type, int size)* kojoj se prosleđuje identifikacioni broj bafera, tip i veličina memorije koju je potrebno zauzeti pozivom ugrađene C funkcije *malloc*.

Moguće je dobaviti član liste preko polja bufId, write_to ili type. Pored toga, moguće je obrisati određeni član liste, prikazati ga u konzoli, kao i obrisati celu listu po potrebi.

4.3.2.3 Digitalni media renderer

Digitalni media renderer predstavlja uređaj predviđen DLNA specifikacijom [3]. Arhitektura digitalnog media renderera (DMR) prikazana je na slici 3.6.

Sastoji se iz osnovnih servisa koje propisuje DLNA, koji su specijalno prilagođeni RVU protokolu kao što je opisano u konceptu rešenja i virtuelnog internet poslužioca koji omogućava deljenje XML datoteka na mreži. U XML datotekama se nalazi opis uređaja i njegovih funkcionalnosti predstavljenih kroz UPnP servise.

DLNA DMR API apstrahuje funkcionalnost DMR uređaja i omogućava njegovo jednostavno korišćenje u okviru RVU API sloja.

Na samom početku, funkcija RVU API sloja *dlna_error_t dlna_dmr_start(dlna_dev_hnd_t handle)* poziva funkciju *char * generate_uuid()* koja generiše jedinstveni identifikacioni broj DMR uređaja koga je potrebno kreirati. Nakon toga, poziva funkciju ovog sloja - *int dmr_init(char *uuid)* kojoj prosleđuje napravljeni uuid. *dmr_init* sastavlja XML koji opisuje DMR uređaj i dobavlja implementirane servise predstavljene na slici 3.6. U sledećem koraku, pravi se virtuelni direktorijum u koji se smešta generisani XML koji opisuje DMR uređaj i njegove servise. To se radi preko UPnP funkcije *UpnpAddVirtualDir(char *dir)* u cilju omogućavanja obraćanja uređaju i korišćenja njegovih funkcionalnosti.

Da bi DMR bio vidljiv ostalim UPnP kompatabilnim uređajima na mreži, potrebno je registrovati uređaj pozivom funkcije *UpnpRegisterRootDevice2*, a zatim objaviti prisustvo pozivom funkcije *UpnpSendAdvertisement*.

DMR uređaj prima instrukcije od strane DLNA DMC uređaja smeštenog u okviru RVU poslužioca i zbog toga je potrebno registrovati rukovaoca događajima. To se radi preko funkcije *void dmr_register_handler(dmr_event_handler_t handler).*

Instrukcije koje DMR prima, prosleđuju se do preko JNI sloja do GStreamer modula za reprodukciju AV sadržaja. Na ovaj način je moguće kontrolisati reprodukciju sa udaljene lokacije. Naravno, kontrolisanje uređaja zavisi od interakcije korisnika sa aplikacijom.

Kada korisnik izabere određeni AV sadržaj, pritiskom na opciju, RVU poslužilac saznaje X i Y komponente događaja. U tom trenutku, RVU poslužilac saznaje da je na toj poziciji naslov koga je potrebno reprodukovati i aktivira DLNA DMC uređaj koji DMR uređaju šalje instrukcije gde se sadržaj nalazi (u vidu HTTP adrese). GStreamer zatim pokreće reprodukciju sa navedene adrese.

Slična logika važi i za kontrolisanje uređaja pritiskom na komande za pauziranje, stopiranje ili premotavanje AV sadržaja.

Kontrola renderovanja predstavlja servis opisan u poglavlju 3.3.2.3. Za implementaciju ovog servisa, iskorišćen je standardizovani UPnP servis [15].

Prilikom izvršavanja funkcije dmr_init, poziva se funkcija ovog servisa: *char* get_rendering_ctrl_scpd(void)* koja generiše XML opis servisa u kome se nalaze akcije, promenljive i argumenti servisa.

Menadžer konekcije predstavlja servis opisan u poglavlju 3.3.2.3. Za implementaciju ovog servisa, iskorišćen je standardizovani UPnP servis[16].

Prilikom izvršavanja funkcije *dmr_init*, poziva se funkcija ovog servisa: *char* get_conn_manager_scpd(void)* koja generiše XML opis servisa u kome se nalaze akcije, promenljive i argumenti servisa.

AV transport predstavlja servis opisan u poglavlju 3.3.2.3. Za realizaciju ovog servisa, iskorišćen je standardizovani UPnP servis [17].

Prilikom izvršavanja funkcije *dmr_init*, poziva se funkcija ovog servisa : *char* get_avt_scpd(void)* koja generiše XML opis servisa u kome se nalaze akcije, promenljive i argumenti servisa.

Virtuelni internet poslužilac predstavlja modul opisan u poglavlju 3.3.2.3. Za korišćenje ovog modula, neophodno je inicijalizovati osnovne UPnP komponente namenjene kreiranju virtuelnog internet poslužioca. To se obavlja preko funkcije *int init_dmr_ws_handler().*

Ukoliko je potrebno dodati novu virtuelnu datoteku i podeliti je na mreži, potrebno je pozvati sledeću funkciju:

int add_virtual_file(char path, unsigned char* content, int content_len, char* content_type);*

Na primer, da bi se dodala virtuelna datoteka koja opisuje AV transport servis, neophodno je pozvati navedenu funkciju na sledeći način:

add_virtual_file(AV_TRANSPORT_SCPD, scpd, strlen((char)scpd), "text/xml; charset=\"utf-8\"");*

gde se scpd (opis servisa sa svim akcijama i argumentima) dobavlja pozivom funkcije *avt_get_scpd()* na sledeći način:

scpd = (unsigned char) get_avt_scpd();*

Pored dodavanja virtuelne datoteke predstavljene u obliku stringa u radnoj memoriji, moguće je dodati i lokalnu datoteku pozivom funkcije:

int add_local_file(char path, char* content_type, char* virtual_path);*

Ova funkcija se koristi za deljenje ikone koja prikazuje DMR uređaj.

4.3.3 Libupnp

Libupnp predstavlja biblioteku otvorenog koda čije se osnovne funkcionalnosti ogledaju u mogućnosti izgradnje i kontrolisanja UPNP uređaja, poput kontrolnih tačaka i poslužilaca. Osnovne informacije o UPnP protokolu prezentovane su u teorijskim osnovama (poglavlje 2.2).

U slučaju realizacije RVU protokola, cilj UPnP protokola jeste pretraživanje i uspostavljanje inicijalne veze sa RVU-RUI poslužiocem, u cilju konfigurisanja RVU korisnika i njegovog osposobljavanja za primanje grafičkih komponenata, zvučnih efekata i AV sadržaja.

Primer sesije je predstavljen na slici 4.2.

Korišćenje UPnP biblioteke se ostvaruje pozivom funkcije *UpnpInit* kojoj se prosleđuju IP adresa i prolaz. Ukoliko se proslede NULL vrednosti, biblioteka sama dobavlja potrebne parametre preko poziva *UpnpGetServerIpAddress* i *UpnpGetServerPort*. Nakon toga se korisnik registruje na mreži pomoću *UpnpRegisterClient* funkcije. Ona registruje rukovalac koji će u nastavku životnog veka aplikacije, da prima događaje poslate od strane UPnP poslužioca koji se nalazi u sklopu RVU poslužioca. Događaji [2] koje UPnP poslužilac u okviru RVU poslužioca šalje su:

- UPNP_DISCOVERY_ADVERTISEMENT_ALIVE
- UPNP_DISCOVERY_SEARCH_RESULT
- UPNP_DISCOVERY_SEARCH_TIMEOUT
- UPNP_DISCOVERY_ADVERTISEMENT_BYEBYE
- UPNP_CONTROL_ACTION_COMPLETE
- UPNP_EVENT_SUBSCRIBE_COMPLETE
- UPNP_EVENT_UNSUBSCRIBE_COMPLETE
- UPNP_EVENT_RENEWAL_COMPLETE

Kada stigne UPNP_DISCOVERY_SEARCH_RESULT, sa njim stižu i informacije o poslužiocu, koje se preuzimaju u cilju saznanja o podržanim servisima kroz koje poslužilac prikazuje svoje sposobnosti. Ukoliko je u pitanju RVU-RUI poslužilac, uspostavlja se veza i razmenjuju se osnovne informacije u cilju uspostavljanja logike za razmenu podataka, predstavljene u poglavlju 4.3.2.2.

4.3.4 GStreamer

GStreamer [11] predstavlja višeplatformsku biblioteku otvorenog koda napisanu u C programskom jeziku. Bibliteka je prilagođena Android platformi i iskorišćena je za reprodukciju AV multimedijalnog sadržaja u MPEG2 formatu, dostavljenog od strane HTTP poslužioca smeštenog na strani RVU poslužioca.

GStreamer biblioteka poseduje arhitekturu baziranu na protočnoj strukturi (engl. pipeline). Protočna struktura (slika 4.6) predstavlja niz elemenata koji omogućavaju protok multimedijalnih podataka u cilju njihove obrade i funkcioniše u zasebnoj niti.

GStreamer biblioteka se inicijalizuje pozivom funkcije *gst_init*. Nakon toga, potrebno je napraviti elemente protočne strukture preko funkcije *gst_element_factory_make*. Za potrebe AV reprodukcije, potreban je element koji čita sadržaj dostavljen od strane HTTP poslužioca, i izdvaja elementarne tokove (audio i video), a to je demultiplekser. Audio sadržaj se šalje ka audio dekoderu, dok video sadržaj ide ka video dekoderu. Da bi se uvezali dekoderi i demultiplekser, potrebno je podesiti „set-pads" rukovalac događajima. Dalje je potrebno dodati izlazne audio i video elemente, koji automatski prepoznaju audio i video uređaje koji vrše reprodukcije sadržaja. Na kraju, sve elemente treba dodati u GstPipeline protočnu strukturu.

5. Ispitivanje i verifikacija

Za potrebe ispitivanja i verifikacije realizovane RVU korisničke programske sprege, razvijene su dve aplikacije koje se nalaze na strani RVU poslužioca i na jednostavan način se mogu prikazati na korisničkom Android uređaju. Zadatak RVU korisnika, jeste da prima grafičke komponente i da ih prikazuje u okviru grafičke korisničke sprege, da reprodukuje zvučne efekte i AV sadržaj i da obaveštava RVU poslužioca o korisničkim događajima.

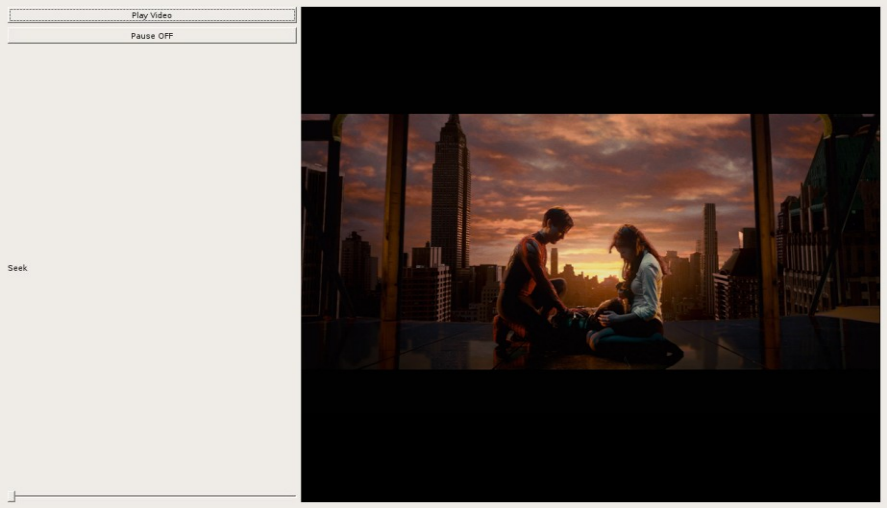
Kao uređaj na kome je ispitana RVU korisnička programska podrška, iskorišćen je mobilni telefon, proizvođača HTC. Radi se o modelu sa oznakom One X koji poseduje četvorojezgarni procesor na 1.5GHz, 1 Gb radne memorije i tvrdi disk veličine 16Gb.

Rešenje je ispitano u okviru WiFi mreže, brzine 64/16 Mb, koja omogućava nesmetano izvršavanje aplikacije, bez mogućih usporenja usled preopterećenosti mreže.

Prva ispitna aplikacija je jednostavna i služi za verifikaciju osnovnih funkcionalnosti aplikacije. Pomoću nje je verifikovana uspešnost povezivanja, vreme trajanja povezivanja i konfiguracije RVU, odziv na osnovne komande, pravilno osvežavanje delova grafičke korisničke sprege i uspešno pokretanje, pauziranje i premotavanje reprodukcije AV sadržaja.

Za konfigurisanje RVU korisnika i njegovo osposobljavanje za rad, potrebno je u proseku 250 milisekundi. Konfigurisanje video prozora uzima u proseku 40 milisekundi, dok konfigurisanje zvuka uzima oko 35 milisekundi.

Ukoliko se grafičke komponente dostavljaju u PNG formatu, a ne u JPEG, znatno se smanjuje broj okvira u sekundi. Razlog leži u prirodi PNG formata koji ne unosi gubitke, već uvodi efikasan algoritam kompresije. Drugi razlog leži u veličini slika, ali i pored toga, slike premašuju veličinu JPEG slika, što se prostora na disku tiče.



Slika 5.1 Izgled aplikacije koja ispituje osnovne funkcionalnosti

Kao što je prikazano na slici, aplikacija omogućava pritisak na dugme „Play video" koje započinje reprodukciju sadržaja smeštenog u okviru RVU poslužioca sa unapred definisane HTTP adrese.

Kada korisnik klikne na navedeno dugme, RVU poslužilac prepoznaje region u kome se desio događaj, tako što mu se dostavljaju X i Y koordinate događaja. Odmah zatim, DLNA DMC dostavlja DLNA DMR uređaju adresu sa koje je potrebno pokrenuti reprodukciju. Svi događaji pristigli DMR-u, propagiraju se do Java sloja gde se prenose GStreamer komponenti koja se ponaša u skladu sa instrukcijama.

Pored dugmeta za početak reprodukcije, moguće je kliknuti i na dugme „pause" koje pauzira reprodukciju. Koordinate događaja se šalju RVU poslužiocu, koji preko DMC obaveštava DMR uređaj da je potrebno pauzirati reprodukciju.

Poslednja funkcionalnost koju poseduje aplikacija, ogleda se u mogućnosti premotavanja video za određeni procenat unapred ili unazad. Logika je slična kao i za prethodnu dugmad, s tim da je ovde u pitanju klizač koji se pomera zadržavanjem kursora i pomeranjem u levo ili desno.

Aplikacija zadovoljava osnovne kriterijume u pogledu funkcionalnosti i omogućava korišćenje RVU protokola u svrhe kontrole AV reprodukcije i izgradnje funkcionalne grafičke korisničke sprege. Što se tiče razlike u formatima grafičkih elemenata koji se dostavljaju, zaključeno je da u ovom slučaju, prednosti jpeg formata ne igraju značaju ulogu, iz razloga što aplikacija ne sadrži zahtevne elemente poput animacija.

Druga ispitna aplikacija je poslužila za subjektivnu ocenu korisničkog doživljaja u toku realne upotrebe. Stoga je ova aplikacija mnogo zahtevnija i sadrži napredne efekte (animacije). Akcenat je stavljen na upotrebljivosti, tj. informaciji koliko ukupno kašnjenje utiče na korisnički doživljaj u toku korišćenja aplikacije.

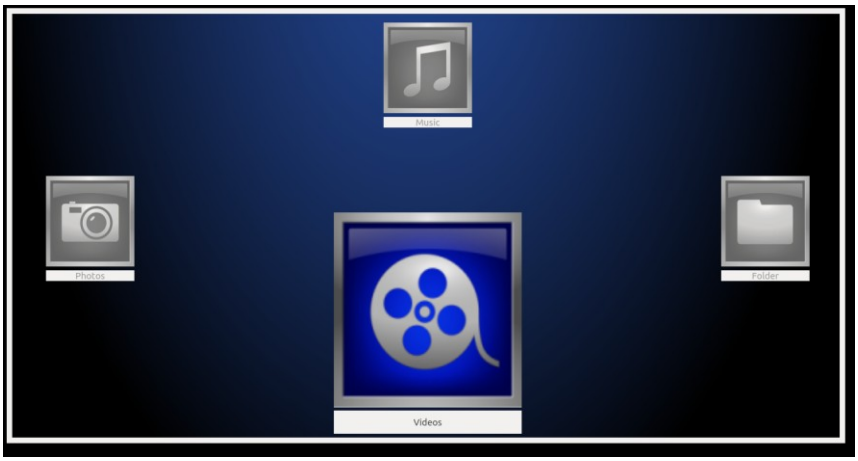
Cilj je bio verifikovati sledeće aspekte rada aplikacije:

- Pravilno pozicioniranje i obeležavanje elemenata korisničke sprege
- Tečna tranzicija u toku rada
- Uticaj upotrebe efekata (animacije) na funkcionalnost aplikacije

Druga ispitna aplikacija izgleda daleko složenije, uvodi animacije i mogućnost pretraživanja sadržaja koji se nalazi na udaljenom DMS uređaju.

Slika 5.2 Izgled aplikacije koja koristi animacije

Aplikacija se sastoji iz velikih ikona, koje predstavljaju sadržaj odabranog DMS uređaja.



Slika 5.3 Izgled aplikacije koja koristi animacije – meni 2

Kretanje kroz menije sadrži animacije, koje se ogledaju u pomeranju ikona na levo ili na desno, kao i na okretanju ikona oko svoje ose.

U slučaju slanja png grafičkih elemenata, broj okvira po sekundi je znatno manji (oko 3 puta) nego u slučaju jpg formata, iz razloga što enkodovanje png grafičkih elemenata na strani poslužioca zahteva 3 puta više vremena.

Pored toga, korišćenje libpng zahteva više vremena za dekompresiju png slike na strani korisnika, nego što je to slučaj kada libjpeg vrši dekompresiju jpg slike. Odnos je 1:3.5 u korist libjpeg i jpg formata

Odabirom opcije „Videos" ulazi se u narednu etapu aplikacije, u kojoj su ikone koje predstavljaju video zapise, poređane horizontalno, kao što je prikazano na slici ispod.

Slika 5.4 Izbor video sadržaja u aplikaciji koja koristi animacije

Odabirom određenog video sadržaja, pokreće se reprodukcija u GStreamer komponenti, na isti način kao i u prethodnoj aplikaciji.

Slika 5.5 Reprodukcija video sadržaja

Nakon ispitivanja druge aplikacije, zaključeno je da ona ispunjava osnovne kriterijume što se tiče korišćenja naprednih elemenata poput animacija, ali samo u slučaju korišćenja grafičkih elemenata u jpeg formatu. U tom slučaju, ostvaruje se zadovoljavajući broj okvira po sekundi, što nudi korisnicima ugodan doživljaj korišćenja aplikacije.

Kada je reč o grafičkim elementima u png formatu, utvrđeno je da ne zadovoljavaju kriterijeme iz razloga što zahtevaju više vremena za enkodovanje na strani RVU poslužioca, kao i za dekodovanje na strani RVU korisnika. Time se uvode značajna kašnjenja u dostavi grafičkih elemenata i aplikacija čini mnogo manje privlačnom.

Ispitivanjem aplikacija u realnom vremenu, utvrđeno je da predviđena realizacija RVU korisničke programske sprege zadovoljava osnovne zahteve i nudi zadovoljavajući kvalitet grafičke korisničke sprege. To se ogleda u ostvarenom broju okvira po sekundi prilikom korišćenja grafičkih komponenata u jpg formatu. Takav scenario omogućava postojanje naprednih elemenata poput animacija što značajno poboljšava korisnički doživljaj prilikom korišćenja aplikacija u svakodnevnom životu.

6. Zaključak

U ovom radu prikazana je implementacija RVU korisnika za Android platformu u domenu umrežene kuće. Android predstavlja platformu za prenosive uređaje, otvorenog je koda i prebačen je na mnoge platforme poput televizora i DTV prijemnika (engl. Set Top box). Kao takav pogodan je za realizaciju široke lepeze rešenja namenjenih za upotrebu u domenu umrežene kuće koja se sve više nameće kao izvor digitalne zabave.

RVU uvodi koncept tzv. „tankog" korisnika, čiji osnovni zadaci su registrovanje korisničkih akcija, iscrtavanje grafičke korisničke sprege, reprodukcija zvučnih efekata i multimedijalnog sadržaja. Procesorski zahtevan posao se odrađuje na poslužiocu, što značajno pojednostavljuje strukturu i prostorne zahteve korisničke aplikacije.

Grafička korisnička sprega, zvučni efekti i multimedijalni sadržaj se nalaze na poslužiocu i preko mreže se dostavljaju svim RVU kompatabilnim korisnicima u umreženoj kući. Na ovaj način svi korisnici imaju isti doživljaj u toku korišćenja aplikacije. Pored toga, pojednostavljena je nadogradnja same aplikacije, unapređena energetska efikasnost prenosivih uređaja i smanjena cena samog razvoja i održavanja.

Osnovni segmenti RVU korisničke programske sprege na Android platformi, iscrtavanje grafičke korisničke sprege i reprodukcija zvučnih efekata, realizovani su u okviru JNI prilagodnog sloja u cilju ostvarivanja što boljih performansi, uzimajući u obzir značajne prednosti (pre svega brzinu izvršavanja) izvornih jezika- C/C++ u odnosu na Java programski jezik koja se izvršava u Androidovoj Dalvik virtuelnoj mašini.

Iscrtavanje grafičke korisničke sprege, implementirano je koristeći Androidovu ugrađenu izvornu biblioteku OpenGL ES 2.0.

Reprodukcija zvučnih efekata koji značajno doprinose sveopštem doživljaju korišćenja aplikacije, realizovana je koristeći Androidovu ugrađenu izvornu audio biblioteku OpenSL ES 1.1.

Kad je reč o jako značajnom segmentu RVU protokola – reprodukciji AV sadržaja, neophodno je reći da standardna Androidova komponenta VideoView ne omogućava reprodukciju MPEG2 sadržaja koji predstavlja standard u digitalnoj televiziji. Iz tog razloga, neophodno je bilo pribeći rešenju koje podržava MPEG2. Izabran je GStreamer, multimedijalno okruženje otvorenog koda realizovano u C programskom jeziku, sa dostupnom verzijom namenjenom za Android platformu.

Buduće nadogradnje aplikacije i RVU biblioteke, biće usmerene ka:

- optimizaciji dekodovanja i iscrtavanja grafičkih komponenti u cilju ostvarivanja što više okvira u sekundi i smanjenja kašnjenja u toku prenosa elemenata grafičke korisničke sprege
- realizaciji naprednih RVU funkcionalnosti poput 3D grafičke korisničke sprege i protokola za dobavljanje inicijalne slike (CIA)
- realizaciji TLS mehanizma za zaštitu komandnog kanala kroz koji se šalju osetljive informacija poput brojeva kreditnih kartica, šifri itd.
- realizaciji DTCP protokola u cilju obezbeđivanja zaštite prilikom isporučivanja multimedijalnog sadržaja

7. Literatura

[1] Papadopoulos, N.; Meliones, A.; Economou, D.; Karras, I.; Liverezas, I.; , "A Connected Home Platform and Development Framework for smart home control applications," Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on, pp.402-409, 23-26 June 2009

[2] UPnP arhitektura, preuzeto 28.06.2014. sa internet adrese : http://UPnP.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[3] DLNA Alliance, 30646_DLNA_Guideline_August_2009 , preuzeto sa http://www.dlna.org/

[4] DLNA Alliance, www.dlna.org /dlna-for-industry/newsroom

[5] Juha Leppilahti: "Remote UI protocols for home environment", TKK T-110.5190 Seminar on Internetworking, 3 May 2007 (2007-05-03), XP055049595, Helsinki Preuzeto 28.06.2014. sa internet adrese : http://www.tml.tkk.fi/Publications/C/23/papers/Leppilahti_final.pdf

[6] Y. Bae; J. Park; “A Seamless Remote User Interface System Supporting Multi-Screen Services in Smart Devices, “Consumer Electronics (ICCE), 2013 IEEE International Conference on, pp. 462 – 463, 11-14 Jan. 1989

[7] T.Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper, “Virtual Network Computing,” IEEE Internet Computing, Vol.2, No.1, pp.33-38, Jan.-Feb. 1998

[8] [MS-RDPBCGR]: Remote Desktop Protocol: Basic Connectivity and Graphics Remoting, preuzeto 28.06.2014. sa internet adrese:

http://msdn.microsoft.com/en-us/library/cc240445.aspx

[9] T. Richardson and J. Levine, "The Remote Framebuffer Protocol," IETF RFC 6143, Mar. 2011, preuzeto 28.06.2014. sa internet adrese: http://www.stolerman.net/studies/cs544/astolerman_paper1.pdf

[10] RVU Protocol: Networked Home Entertainment With Pixel Accurate Remote Graphics – White Paper, preuzeto 28.06.2014. sa internet adrese: http://www.rvualliance.org/files/static_page_files/RVU_White_Paper.pdf

[11] GStreamer, http://gstreamer.freedesktop.org

[12] Libpng, http://www.libpng.org/pub/png/libpng.html

[13] Libjpeg, http://libjpeg.sourceforge.net/

[14] Kevin Brothaler : *OpenGL ES 2 for Android,* datum objave 6. jul 2013| ISBN-10: 1937785343 | ISBN-13: 978-1937785345, prva edicija

[15] UPnP servis – RenderingControl, preuzeto 28.06.2014. sa internet adrese http://UPnP.org/specs/av/UPnP-av-RenderingControl-v1-Service.pdf

[16] UPnP servis – ConnectionManager, preuzeto 28.06.2014. sa internet adrese http://UPnP.org/specs/av/UPnP-av-ConnectionManager-v1-Service.pdf

[17] UPnP servis – AVTransport, preuzeto 28.06.2014. sa internet adrese http://UPnP.org/specs/av/UPnP-av-AVTransport-v1-Service.pdf

[18] RemoteUIServer servis, preuzeto 28.06.2014. sa internet adrese http://www.UPnP.org/specs/rui/UPnP-rui-RemoteUIServer-v1-Service.pdf