УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Ивана Николић

Реализација плејера за мултимедију у возилу

МАСТЕР РАД

Нови Сад, 2017

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**:		Ивана Николић	
Ментор, **МН**:		доц.др Милан Бјелица	
Наслов рада, **НР**:		Реализација плејера за мултимедију у возилу	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2017	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/65/0/7/26/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кqучне речи, **ПО**:		аутомотив, мултимедија, плејер за мултимедију, аутомобил, аудио, видео, QNX ОС, Qt	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом раду је представљено једно рјешење плејера за мултимедијалне садржаје у аутомобилима, у апликативним окружењима заснованим на Qt-у, које такође подржава и манипулацију репродукцијом саджаја покретима руке. Описано рјешење испуњава захтјеве модерног потрошача, а истовремено захтјева минималну пажњу од возача, чиме се смањује његова дистракција.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	проф.др Драган Кукољ	
	Члан:	доц.др Јелена Митровић-Симић	Потпис ментора
	Члан, ментор:	доц.др Милан Бјелица	

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**:		Ivana Nikolić	
Mentor, **MN**:		PhD Milan Bjelica	
Title, **TI**:		Multimedia Player in Vehicles	
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**:		2017	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/65/0/7/26/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S**/**KW**:		automotive, in-vehicle infotainment, multimedia, multimedia player, audio, video, QNX OS, Qt	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This paper presents one solution for multimedia players in vehicles in Qt-based application environments, which also supports content playback control by hand gestures. This solution meets the demands of the modern consumer while at the same time requires minimal attention from the driver, thus reducing his/her distraction.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Dragan Kukolj, Ph.D.	
	Member:	Jelena Mitrović-Simić, Ph.D.	Menthor's sign
	Member, Mentor:	Milan Bjelica, Ph.D.	

Zahvalnost

Zahvaljujem se mentoru doc. dr Milanu Bjelici na stručnoj pomoći i savjetima za izradu ovog rada.

Zahvaljujem se svojim roditeljima, Violeti i Zoranu, na bespogovornoj podršci tokom mog školovanja.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

ADAS – *Advanced Driver Assistance Systems*, sistemi koji pomažu vozaču u procesu vožnje

ASIL D – *Automotive Safety Integrity Level D*, šema klasifikacije rizika definisana ISO26262 standardom „Funkcionalna sigurnost za drumska vozila“. ASIL D diktira najviše zahtjeve na proizvodu

AVRCP – *Audio/Video Remote Control Profile*, ovaj profil obezbjeđuje standardni interfejs koji omogućava daljinskom upravljaču (ili drugom uređaju) da kontroliše svu audio/video opremu kojoj korisnik ima pristup

BSP – *Boart Support Package*, sloj softvera koji sadrži hardverske upravljačke programe i druge rutine koje omogućavaju određenom operativnom sistema da funkcioniše u određenom hardverskom okruženju integrisanom sa samim *RTOS*-om

CID – *Central Information Display*, centralni ekran *IVI* sistema na kom se prikazuje mapa, i svi dostupni korisnički dijalozi (audio, video, telefon, podešavanja, navigacija, itd.)

DLNA – *Digital Living Network Alliance*, standard za dijeljenje podataka između uređaja

DVD – *Digital Versatile Disk,* format za skladištenje digitalnih optičkih diskova koji su 1995. godine izmislili i razvili *Panasonic*, *Philips*, *Soni* i *Toshiba*. Može da skladišti sve vrste digitalnih podataka i široko se koristi za softver i druge računarske datoteke, kao i video programe koji se gledaju pomoću *DVD* plejera. *DVD*-ovi nude veći kapacitet skladištenja od kompaktnih diskova dok imaju iste dimenzije

GPS – *The Global Positioning System*, satelitski navigacioni sistem

GPU – *Graphics Processing Unit*, specijalizovano elektronsko kolo dizajnirano da brzo manipuliše i mijenja memoriju kako bi ubrzalo stvaranje slika u baferu namijenjenih za izlaz na uređaj za prikazivanje

HTML5 – *Hypertext Markup Language*, poslednja verzija jezika za opis internet stranica

HUD – *Head Unit Display*, ekran koji se u automobilima obično nalazi ispred vozača, iznad kontrolne table (*Cluster* ekrana). Na ovom ekranu su prikazane informacije kao što je trenutna brzina kretanja, vrijeme, razdaljina koju je moguće preći sa trenutnom količinom goriva, itd.

IPC – *Inter-process Communication*, mehanizam koji omogućava procesima (koji se izvršavaju unutar istog operativnog sistema) da upravljaju zajedničkim podacima

IVC – mehanizam koji omogućava procesima (koji se izvršavaju u različitim operativnim sistemima) da upravljaju zajedničkim podacima

IVI – *In-Vehicle Infotainment*, termin iz automobilske industrije koji se odnosi na integrisani skup hardvera i softvera u automobilu koji vozačima i putnicima dostavlja informacije i pruža sadržaj zabavnog karaktera

NPAPI – *Netscape Plugin Application Programming Interface*, okruženje za programiranje aplikacija, koje omogućava razvoj dodataka za internet pregledače

Qt5 – razvojni okvir koji se koristi za razvoj aplikacija, namijenjenih različitim softverskim i hardverskim platformama, i grafičkih korisničkih okruženja

RTOS – *Real Time Operating System*, operativni sistem koji obrađuje podatke u realnom vremenu

RVI – *Remote Visual Inspection*, daljinska vizuelna kontrola video snimaka. Predstavlja oblik vizuelne kontrole koja koristi vizuelna pomagala, uključujući video tehnologiju.

SDK – *Software Development Kit*, skup alata za razvoj softvera koji omogućava kreiranje aplikacija za određeni softverski paket, softverski okvir za razvoj aplikacija, hardversku platformu, računarski sistem, konzolu za video igrice, operativni sistem, itd.

SQL – *Structured Query Language*, domenski jezik u programiranju dizajniran za upravljanje podacima koji su dio relacijskih baza podataka, ili za obradu toka podataka u sistemu za upravljanje tokovima relacionih podataka

UPnP – *Universal Plug and Play,* set mrežnih protokola koji omogućavaju umreženim uređajima, kao što su lični računari, štampači, *Wi-Fi* pristupne tačke i mobilni uređaji, da lako otkrivaju prisustvo drugih uređaja na mreži i uspostavljaju funkcionalne mrežne usluge za razmjenu podataka, komunikacije i zabavu

USB – *Universal Serial Bus,* industrijski standard koji definiše kablove, konektore i komunikacijske protokole za povezivanje, komunikaciju i napajanje između računara i uređaja

1. Uvod

Ranije je medijski sadržaj bilo moguće reprodukovati samo na računarima opšte namjene, zbog zahtjeva za skladištenjem i obradom, međutim napredak u računarskoj tehnologiji omogućio je širokom nizu potrošačkih elektronskih uređaja da inkorporiraju funkcionalnost za čuvanje i reprodukciju multimedijalnog sadržaja. Samim tim, u korak sa razvojem novih tehnologija za dobavljanje i reprodukciju medijskog sadržaja, raste i popularnost zabavnih sadržaja kao što su muzika, video zapisi, filmovi i slično. Danas, razvoj u oblasti hardvera i softvera za multimediju je takav da, skoro da ne postoji ograničenje u načinu na koji se može konfigurisati multimedijalni sistem u automobilu. Današnji automobili se ne proizvode bez podrazumijevanih *IVI* sistema opremljenih sa nekoliko ekrana (uključujući i one osjetljive na dodir), prenosnih *DVD* uređaja, mogućnošću upravljanja određenim dijelovima sistema samo pokretom ruke ili glasovnim komandama, itd. Razvoj bogato opremljenih multimedijalnih sistema ima vrlo značajnu ulogu u razvoju automobila, jer način komunikacije i interakcije vozača sa automobilom postaju odlučujući faktori.

Najveći izazov pri razvoju *IVI* sistema jeste obezbijediti sigurnost i udobnost vožnje, a istovremeno ispuniti zahtjeve korisnika. Multimedijalni sistem u automobilu treba da bude dizajniran tako da, za upravljanje različitim funkcijama sistema, ne zahtijeva previše pažnje.

Rješenje opisano u ovom radu predstavlja jedan način realizacije *player*-a za multimediju u automobilima. Kako je jedan od najvećih izazova pri razvoju *IVI* sistema postizanje sigurnosti vožnje (u smislu količine distrakcije vozača – *IVI* sistemi treba što manje da odvlače pažnju), neke akcije vezane za manipulaciju reprodukcijom su realizovane tako da ih je, osim klika na određeno dugme, moguće izvršiti i pokretom ruke.

Ovaj rad je sačinjen od 7 poglavlja.

U drugom poglavlju su objašnjeni pojam i značaj multimedije, i predstavljeno je nekoliko trenutno najznačajnijih tehnologija za razvoj multimedijalnih sistema za automobil.

U trećem poglavlju je dat dijagram rješenja, kao i opis arhitekture sistema.

Četvrto poglavlje sadrži detaljan opis programske realizacije audio *player*-a i korisničkog dijaloga za reprodukciju audio sadržaja.

U petom poglavlju je opisano ispitivanje performansi rješenja i verifikacija funkcionalnih zahtjeva.

Šesto poglavlje daje osvrt na ono što je do sada urađeno, i dalji pravac kretanja razvoja.

U sedmom poglavlju je naveden spisak literature korištene prilikom pisanja rada.

2. Teorijske osnove

Objašnjenje *player-a*, kao jednog sastavnog dijela multimedijalnog sistema o kom će se detaljnije govoriti u narednim poglavljima ovog rada, zahtijeva predznanje o multimediji i njenoj širokoj upotrebi u različitim oblastima. Zbog toga, naredno poglavlje govori o multimediji onoliko koliko je potrebno za dalje razumijevanje ovog rada. Nakon pojma multimedije će biti predstavljeno nekoliko trenutno popularnih tehnologija za predstavljanje multimedijalnog sadržaja u automobilima, a u poslednjem poglavlju ove cjeline (i na kraju predstavljanja svake od tehnologija) će biti data diskusija izbora tehnologija u smislu rješenja predstavljenog u ovom radu.

2.1 Multimedija

Ideja gledanja filmova ili igranja video igrica u automobilima nije postala popularna sve do početka 21. vijeka, a čak i tada je multimedijalni sadržaj u automobilima u velikoj mjeri bio ograničen na skupe *VCR* (*Videocassette Recorder* – video rekorder) ili *DVD* (*Digital Video Disk* – digitalni video disk) sisteme [1]. Međutim, zahvaljujući novom digitalizovanom načinu života, sve većem razvoju 'uvijek povezani' (eng. *always connected* – u smislu pristupa internetu) zajednica i sve višim zahtjevima korisnika, u poslednjih nekoliko godina dostupnost multimedijalnih sadržaja u automobilu postaje jednako važna za kupce kao i ‘ono što se nalazi ispod haube’. U skladu sa tim, i u cilju obezbjeđivanja jedinstvenog iskustva vožnje, raste dostupnost različitih oblika multimedijalnih sadržaja u automobilima, kao i broj njihovih funkcionalnosti.

Pojam multimedije se odnosi na sadržaje koji nastaju kompjuterskom integracijom teksta, grafike, slika, videa, animacija, audio sadržaja i svih drugih medija u kojima se svaka vrsta

informacija može predstaviti, sačuvati, prenijeti i obraditi digitalno. Multimedija se može snimati, reprodukovati i prikazivati, a takođe je moguća i interakcija sa multimedijalnim sadržajima, kao i pristup multimediji od strane uređaja za obradu informacija, kao što su kompjuterizovani i elektronski medijski uređaji [2].

Koliko je multimedija danas značajna govori i činjenica da multimedijalni sadržaji pronalaze svoju primjenu u različitim oblastima, uključujući, ali ne ograničavajući se na reklame, umjetnost, obrazovanje, zabavu, inženjering, medicinu, matematiku, biznis, naučna istraživanja, itd [2].

U narednom potpoglavlju će ukratko biti predstavljeno nekoliko već postojećih rješenja multimedijalnih *player*-a u automobilskoj industriji.

2.1.1 Dostupna finalna rješenja multimedija player-a u industriji

ICS kompanija [3], sa sjedištem u Valthamu Masačusets (eng. *Waltham Massachusetts*) i kancelarijama u Kaliforniji, Kanadi i Evropi, se bavi dizajniranjem i implementacijom *IVI* sistema za putničke automobile, sistema za zabavu (eng. *inflight entertainmnet – IFE*) za međunarodne aviokompanije i povezanih upravljačkih sistema za komercijalna i poljoprivredna vozila, viljuškare, itd. Pored navedenog, ova kompanija nudi integrisani prilagođeni razvoj softvera i dizajn korisničkog okruženja za ekran osjetljiv na dodir, mobilne, medicinske, ugrađene i desktop aplikacije za vodeće kompanije širom svijeta.

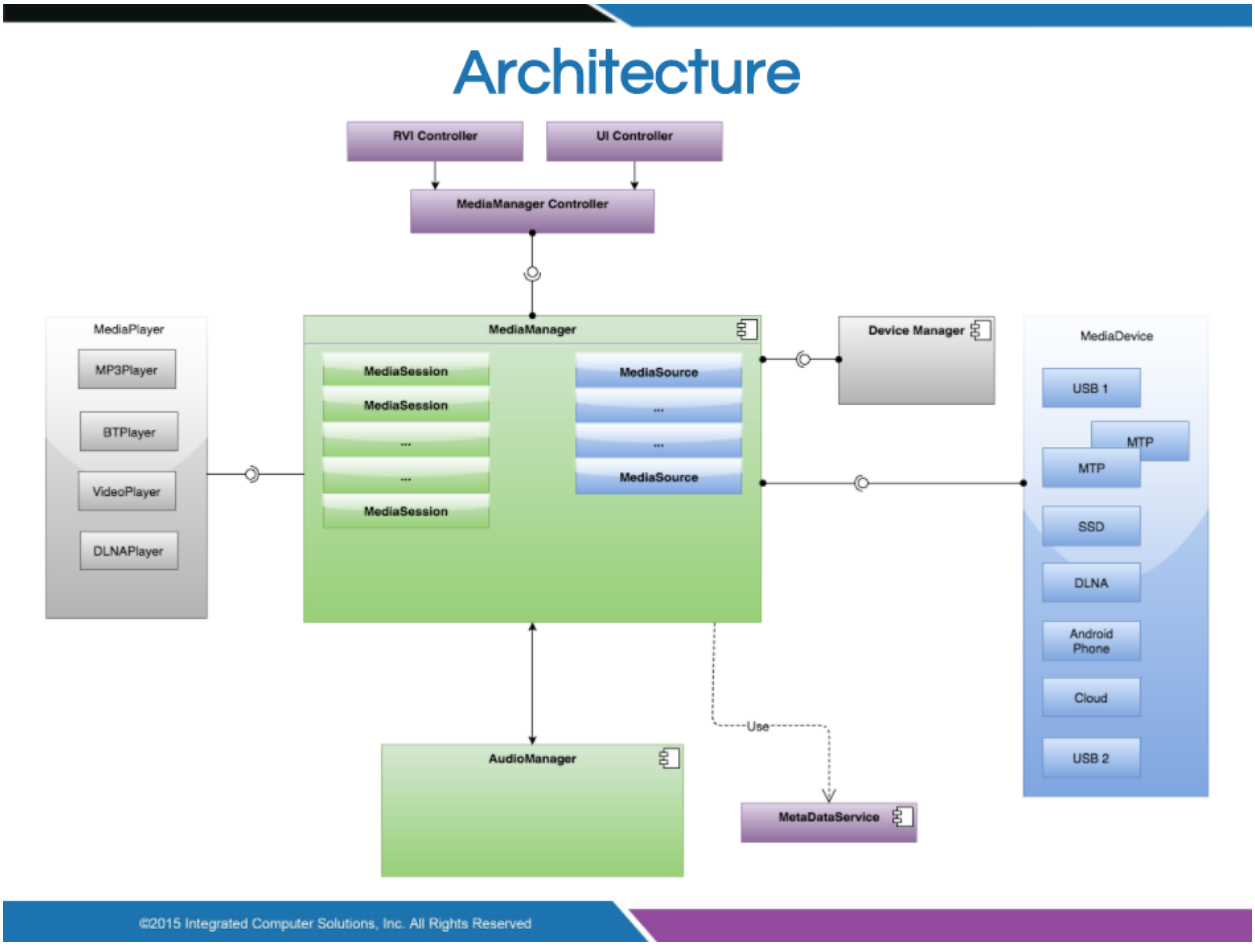
U nastavku teksta će biti predstavljen medija menadžer (eng. *media manager*) za putničke automobile, razvijen od strane pomenute kompanije.

Arhitektura medija menadžera je osmišljena sa ciljem prevazilaženja sledećih izazova [4]:

1. Inteligentno rukovanje dodavanjem i uklanjanjem medijskih sadržaja, na primjer upravljanje medijskim sadržajima sa telefona, tableta, udaljenih servera, *USB* uređaja, itd,
2. Obezbjeđivanje svim izvorima medijskih sadržaja, navedenih u prethodnoj tački, da budu dostupni kroz jedan jedinstveni interfejs,
3. Omogućavanje putnicima pretraživanje, sortiranje i filtriranje medijskog sadržaja u liste dok su u automobilu, kao što to rade na svojim telefonima, tabletima ili pametnim uređajima kod kuće,
4. Omogućavanje proizvođačima automobila implementaciju inteligentnih rješenja za automatsko kreiranje liste pjesama, i
5. Obezbjeđivanje direktnog medijskog izlaza za određene putnike u automobilu.

Na slici 1 je prikazana arhitektura menadžera medijskih sadržaja i interakcija njegovih glavnih komponenti:

1. *MediaSource*,
2. *MediaSourcePlaylists*,
3. *MediaSession*,
4. *MediaPlayer*,
5. *MediaManager*.



Slika 1 Arhitektura medija menadžera razvijenog od strane *ICS* kompanije

MediaSource obezbjeđuje interfejse za uređaje (uređaji su fizički mediji) kao što su telefoni, *USB* uređaji, *DLNA*, *Bluetooth*, udaljeni server ili bilo koji izvor koji se može indeksirati (desni blok sa slike – *MediaDevice*).

Svaki izvor daje menadžeru jednu ili više listi medijskog sadržaja. *MediaManager* preuzima ove liste i dodaje ih u odgovarajuće medijske sesije (eng. *MediaSessions*). Za svaku vrstu medija postoji posebna sesija. Dakle, svaka medijska sesija sadrži listu medijskog sadržaja sa zapisima specifičnim za vrstu medija, na primjer *mp3* datoteke, video datoteke, *Bluetooth* strimove, itd.

Na primjer, video liste za reprodukciju se prosleđuju sesiji koja je povezana sa video *player*-om, dok se *Bluetooth* liste prosleđuju *Bluetooth player*-u koji kontroliše *Bluetooth* uređaj preko *AVRCP* protokola.

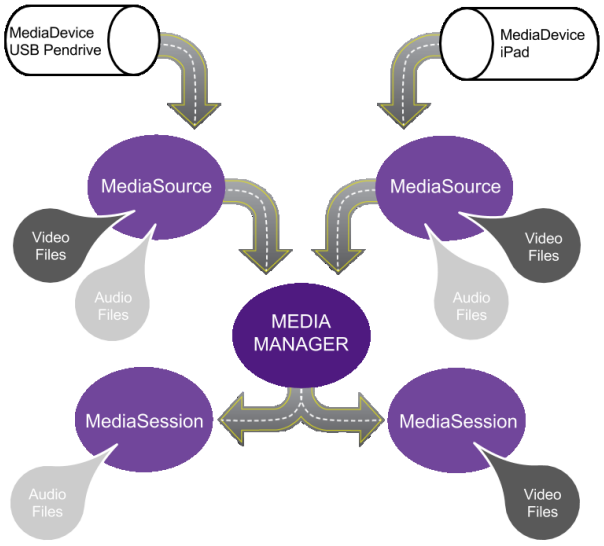
Media player-i kontrolišu izlaz medijskog sadržaja. U njima su realizovane osnovne funkcije za reprodukciju sadržaja:

- Reprodukcija,
- Pauziranje,
- Zaustavljanje,
- Prelazak na sledeći/prethodni sadržaj iz liste, itd.

Media player-i takođe mogu davati direktan izlaz na određene kanale preko audio menadžer komponente.

MediaManager se povezuje sa audio menadžerom zbog povezivanja na različite audio kanale, i menadžerom uređaja kako bi dobio obavještenja od uređaja. Takođe, *MediaManager* obezbjeđuje interfejs kontrolera (eng. *MediaManager Controller*) koji omogućava implementaciju korisničkog interfejsa pomoću izbornog alata, kao i daljinsko upravljanje putem *RVI* ili internet interfejsa (eng. *web interface*).

Na slici 2 je prikazan tok podataka od umetnutog uređaja, ili uređaja na kom je sačuvana lista medijskog sadržaja (udaljeni server), do formiranja liste u medija menadžeru.



Slika 2 Tok podataka od uređaja na kom je sačuvan medijski sadržaj do formiranja liste

Kada korisnik priključi uređaj, na kom se nalaze medijski sadržaji, na sistem (na primjer *USB* uređaj, *iPad*, *mp3 player*, telefon, itd.) *MediaSource* komponenta (za svaki priključen uređaj postoji poseban interfejs) predaje te sadržaje *MediaManager*-u koji ih prosleđuje u odgovarajuće

sesije (npr. audio sadržaj se predaje sesiji za audio sadržaje, a video sadržaji se predaju sesiji za video sadržaje).

ICS-ov *Media Manager* se bavi posebnim problemima sa kojima se suočavaju svi koji se bave razvojem savremenih *IVI* sistema, i predstavlja dobro rješenje srednjeg sloja, u smislu arhitekture, koje može poslužiti kao osnova nekog *IVI* sistema.

Cinemo GmbH kompanija [5], osnovana krajem 2008. godine sa sjedištem u Njemačkoj, je globalni lider u razvoju softverskih rješenja namijenjenih *IVI* sistemima. Fokus ove kompanije je na dizajniranju i optimizovanju softverskih rješenja, ali i visokokvalitetnoj korisničkoj podršci i uslugama. Sa višegodišnjim iskustvom u razvoju multimedijalnih sistema, cilj ove kompanije je da zadovolji visoka očekivanja partnera u automobilskoj industriji, koji se bave ugrađenim sistemima (eng. *embedded systems*), u razvoju modularne i hardverski nezavisne softverske arhitekture i isporučivanju visokokvalitetne tehnološke platforme sa univerzalnom podrškom za fajlove, diskove i metapodatke.

Jedinstvena rješenja srednjeg sloja ove kompanije dekoduju, reprodukuju, renderuju, strimuju, upravljaju i indeksiraju praktično bilo koji fajl, disk, povezani uređaj, format striminga i sadržaj sa udaljenih servera. Dizajnirana i optimizovana tako da zadovolje visokokvalitetne zahtjeve u automobilskoj industriji, *Cinemo*-ova rješenja se mogu vrlo lako integrisati u glavne jedinice IVI sistema ili jedinice sistema koje se nalaze na zadnjim sjedištima [6].

Ova kompanija je 2013. godine predstavila svoj *Multimedia Management Engine* dizajniran da omogući trenutni pristup svim medijskim sadržajima koji su dostupni u automobilu bez obzira na lokaciju ili uređaj za skladištenje. Ovaj *Media Management Engine* podržava [7]:

- Ultra-brze pretrage,
- Složene upite,
- Pametnu sinhronizaciju,
- Alfabetsko generisanje indeksa,
- Kreiranje video zapisa,
- Dinamičko pretraživanje na zahtjev, kao i
- Značajne inteligentne algoritme i
- Opsežne optimizacije koje značajno poboljšavaju način i brzinu kontrole *IVI* digitalnih medija.

Ovaj softver koristi *UPnP* tehnologiju koja omogućava pretraživanje medijskog sadržaja sa lokalnih ili udaljenih uređaja. Pretraživač sadržaja, *ScanFast*, koji je dio *Media Management Engine*-a, je inteligentan i brz indeksni mehanizam koji može upravljati uređajima za skladištenje sa stotinama hiljada multimedijalnih datoteka. Njegov višestruki pristup za

indeksiranje omogućava korisnicima da pretražuju i reprodukuju multimedijalni sadržaj gotovo odmah nakon umetanja u sistem, mnogo prije nego što se proces otkrivanja sadržaja završi.

Zahvaljujući klijent/server arhitekturi, *Multimedia Management Engine* se može pokrenuti ili na jednom sistemu, na primjer na glavnoj jedinici *IVI* sistema (eng. *head unit*), ili preko bežične mreže može da bude povezan na više uređaja (na primjer sa jedinicama koje se nalaze na pozadini sjedišta u automobilu), a arhitektura zasnovana na dodacima (eng. *plugins*) daje mogućnost proširivanja *Media Management*-a novim izvorima medijskog sadržaja.

2.1.2 Zahtjevi korisnika

U narednoj tabeli će biti navedeni neki od zahtjeva korisnika koje *player* jednog savremenog multimedijalnog sistema treba da ispuni:

Naziv funkcionalnosti	Kratak opis
Početni dijalog	Prikazuje informacije o pjesmi koja se trenutno reprodukuje (naziv pjesme, ime izvođača, ime albuma) i sadrži potrebne funkcionalnosti (dalje u tabeli)
Lista medijskog sadržaja	Dijalog sa listom pjesama
Next/Previous	Mogućnost prebacivanja na sledeći/prethodni medijski sadržaj iz liste (na početnom dijalogu)
Play/Pause	Mogućnost zaustavljanja/ponovnog puštanja pjesme
Play/Total time	Prikaz trenutnog/ukupnog vremena trajanja reprodukovanog sadržaja
Shuffle	Opcija nasumičnog izbora sadržaja koji se reprodukuje
Repeat	Opcija koja omogućava reprodukciju istog sadržaja više puta
Volume up/down	Mogućnost pojačavanja/smanjivanja jačine zvuka
Mute/Demute	Opcija isključivanja/uključivanja zvuka

Mala brzina odziva	Nakon tzv. hladnog starta (startovanje motora – paljenje sistema) prihvatljiva brzina odziva varira između 5 i 10 sekundi (inače 3 sekunde)
Promjena izvora reprodukcije	Reprodukcija sa *USB*-a, *CD*-a, telefona, itd.
Reakcija na određeni pokret rukom	Promjena pesme na mahanje ulijevo ili udesno, u zavisnosti da li korisnik želi da pusti prethodnu ili sledeću pjesmu

Tabela 1 Audio *player* – zahtjevi korisnika

Pored zahtjeva navedenih u tabeli 1, postoji još nekoliko ključnih stvari, koje značajno poboljšavaju korisničko iskustvo, na koje treba posebno obratiti pažnju prilikom projektovanja arhitekture *player*-a za *IVI* sistem:

- Dobavljanje različitih formata multimedijalnih sadržaja sa različitih medijskih izvora – današnji korisnici posjeduju različite formate medijskih sadržaja (npr. video, audio, audio knjige, televizija) na različitim medijskim izvorima (na primjer, *USB* uređaj, telefon, udaljeni server, *iTunes* biblioteka, računar, itd.) i žele da budu u mogućnosti da bez problema uživaju u svojim sadržajima bez obzira na kom uređaju ili serveru su oni sačuvani,
- Obezbjeđivanje medijskog sadržaja određenim putnicima – kad u automobilu ima više putnika svaki pojedinac bi trebalo da bude u mogućnosti da uživa u sopstvenim medijskim sadržajima, što dodatno usložnjava potencijalne mogućnosti jednog *player*-a,
- Katalogizacija i indeksiranje medijskog sadržaja – predstavljaju ključ za obezbjeđivanje brzog pristupa sadržaju . Korisnici žele da budu u mogućnosti da brzo pronađu ono što žele, što dalje implicira da je organizacija medijskog sadržaja jedna od ključnih stavki,
- Privlačan izgled korisničkog okruženja.

Prilikom razvoja arhitekture multimedijalnog *player*-a za *IVI* sistem treba obratiti pažnju na zadovoljavanje zahtjeva navedenih u ovom poglavlju, jer kvalitet bilo kog proizvoda na tržištu zavisi isključivo od korisničkog iskustva. Dakle, korisnicima treba pružiti mogućnost reprodukovanja sadržaja sa bilo kog izvora, brz pristup multimedijalnim sadržajima, brze pretrage, mogućnost reprodukovanja različitih sadržaja na više različitih izlaza (kada je više putnika u automobilu), privlačan izgled aplikacije, i obezbijediti što više funkcionalnosti u smislu manipulisanja reprodukcijom.

2.2 Gestovna kontrola

Gestovna kontrola (kontrola određenih funkcionalnosti u automobilu pokretom ruke) se, u kombinaciji sa dugmadima, ekranom osjetljivim na dodir i reagovanjem na glasovne komande, sve više primjenjuje u automobilskoj industriji sa ciljem pojednostavljivanja kontrole sistema, kako bi se smanjila distrakcija vozača izazvana eventualnom interakcijom sa *IVI* sistemom [8].

Za prepoznavanje gesta se koristi senzor blizine (može i kamera u kombinaciji sa senzorom – zavisi koliko kompleksne pokrete proizvođač želi da podrži) koji prepoznaje određene pokrete rukom za unaprijed programirane funkcije. Na primjer, rotiranjem prsta u smjeru kazaljke na satu se može povećati jačina zvuka, ili se određenim pokretom ruke može prihvatiti ili odbiti dolazni poziv. Na ovaj način se, pored olakšavanja interakcije sa sistemom i smanjivanja distrakcije vozača, takođe pojednostavljuje unutrašnji dizajn i oslobađa prostor za opcije skladištenja [9].

Među automobilima koji podržavaju kontrolu pokreta je serija *BMW 7* koja prepoznaje šest različitih pokreta mahanja rukom ispred središnje konzole. Ostali modeli automobila koji takođe podržavaju ovu naprednu funkcionalnost su *Subaru WXR*, *Seat Leon* i *VW Golf GTI*. Sistem koji se koristi na *Golf*-u, na primjer, prepoznaje poteze za premještanje horizontalno uređenih konzolnih stavki menija ulijevo ili udesno, omogućavajući vozaču da mijenja radio stanice, lista liste za reprodukciju, itd.

Izvan automobila, kontrola pokreta se takođe koristi za otvaranje i zatvaranje prtljažnika bez upotrebe ruku. Modeli automobila koji pružaju ovakvu kontrolu pokreta su *Jaguar F-Pace*, *Land Rover Discovery*, *Ford Kuga* i *Audi S5*.

Automatizovani automobili su trenutno najinteresantnija tema u automobilskoj industriji. Pored gestovne kontrole, neki oblici automatizovanih automobilskih tehnologija, kao što je *ADAS* tehnologija, se već uvode u proizvodnju vozila. Međutim, iako neki proizvođači automobila uveliko najavljuju svoje planove o postepenom uvođenju automatizovanih mogućnosti, tek između 2025. i 2035.godine se očekuje uvođenje autonomnih automobila u upotrebu [10].

2.3 Tehnologije za predstavljanje multimedije u automobilima

U ovom poglavlju će ukratko biti predstavljene trenutno najznačajnije tehnologije za predstavljanje multimedije u automobilima. Prve tri predstavljaju platforme za razvoj *IVI* sistema u vozilima – *GENIVI*, *Microsoft Embedded Automotive* i *QNX*, dok poslednje dvije (*Android Auto* i *CarPlay*) predstavljaju jedan od načina prikaza multimedijalnog sadržaja.Na kraju svake cjeline jedan pasus je posvećen kratkom osvrtu na opisivane tehnologije u smislu poređenja sa

tehnologijom koja je korištena pri izradi rješenja opisanog u ovom radu, a poslednja cjelina ovog poglavlja posvećena je pregledu zastupljenosti vodećih tehnologija za razvoj *IVI* sistema.

2.3.1 GENIVI

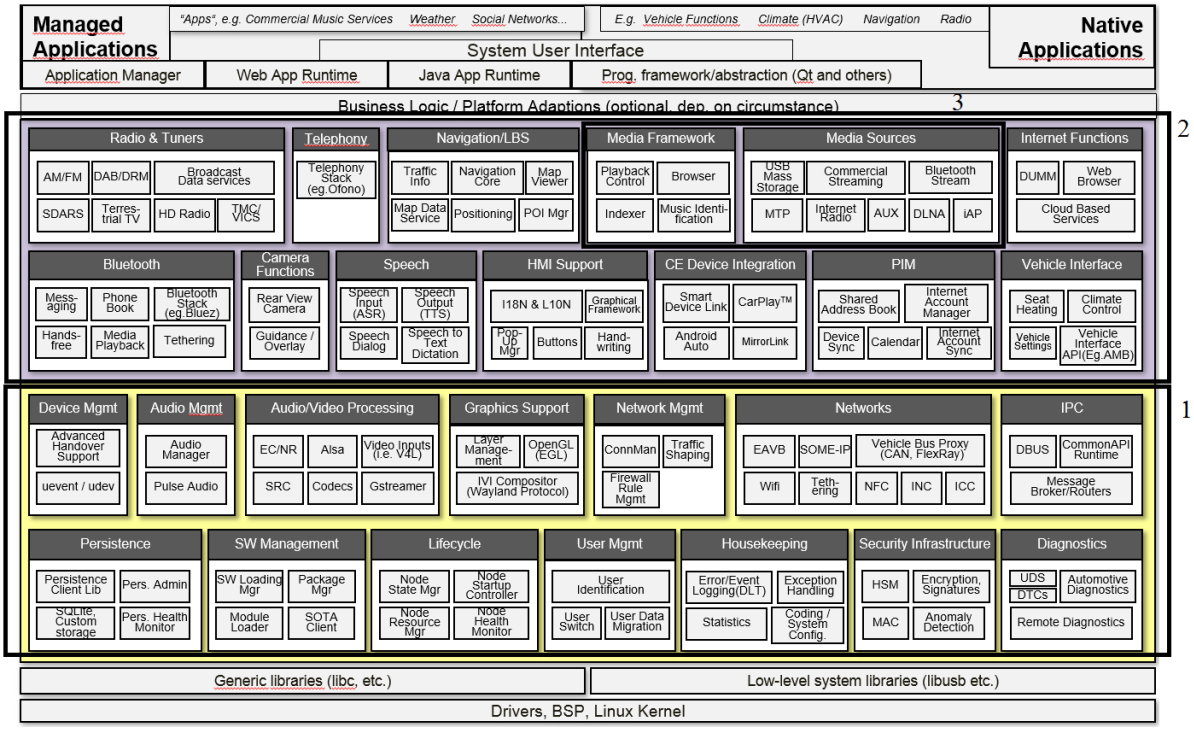
GENIVI alijansa (eng. *GENIVI Alliance*) je neprofitna alijansa u automobilskoj industriji, koja je posvećena razvijanju otvorene i globalno konzistentne platforme zasnovane na *Linux* operativnom sistemu, koju će koristiti cijela automobilska industrija za razvoj *IVI* sistema [11].

Razvojna platforma *GENIVI* (eng. *The GENIVI Development Platform – GDP*) je standardna platforma za razvoj *IVI* sistema i koncepta 'povezanih automobila', demonstracije i inovativne projekte koji koriste otvoren kod (eng. *Open Source*) i operativni sistem *Linux* [12]. *GDP* takođe predstavlja podrazumijevanu polaznu tačku za razvoj brojnih *IVI* proizvoda baziranih na *Linux-u*. Sadrži kompatibilnu *GENIVI middleware* platformu koju koristi veliki broj proizvođača originalne opreme (eng. *Original Equipment Manufacturer – OEM*) i tzv. *Tier 1* kompanija (*Tier 1* kompanija je prva u lancu kompanija koje snabdijevaju proizvođače originalne opreme. Postoje i *Tier 2*, *Tier 3,* itd. kompanije, što zavisi od kompleksnosti lanca proizvodnje.).

GDP ima tri glavna cilja:

- Obezbijediti sistem koji programeri mogu koristiti za kreiranje softverskih komponenti za automobilske sisteme,
- Obezbijediti platformu sa fokusom na *automotive* (odnosi se na automobilsku industriju) slučajeve upotrebe kako bi se programerima omogućilo da kreiraju *automotive* aplikacije i *demo-e* (probni program/sistem koji služi za demonstraciju određenih funkcionalnosti),
- Obezbijediti set alata koji se može skinuti i prilagoditi različitim hardverima za automobilsku industriju.

Na Slici 3 je prikazana arhitektura *GENIVI IVI* platforme [13].



Slika 3 Arhitektura *GENIVI IVI* platforme

Fokus ove platforme je na *middleware-u* (dio softvera koji predstavlja most između donjih i gornjih slojeva softvera) koji je uokviren pravougaonicima 1 i 2 sa slike. Sloj arhitekture uokviren pravougaonikom 1 se bavi ključnim domenima infrastrukture. Manji blokovi ovog sloja predstavljaju pojedinačne softverske komponente, koje su dostupne zajednici otvorenog koda, a neke od njih su razvijene u okviru *GENIVI* projekata otvorenog koda (eng. *open source projects*). U sloju iznad, uokvirenom pravougaonikom 2, manji blokovi predstavljaju funkcionalne domene kojima je možda potrebno dodati nekoliko softverskih komponenti u komercijalnim projektima. Unutar bloka broj 2 je označen dio softvera koji predstavlja *Media Manager* (blok 3).

Media Manager [14] je jedan od *GENIVI* projekata otvorenog koda (eng. *open source*) koji čini jedan od dio arhitekture *GENIVI* platforme.

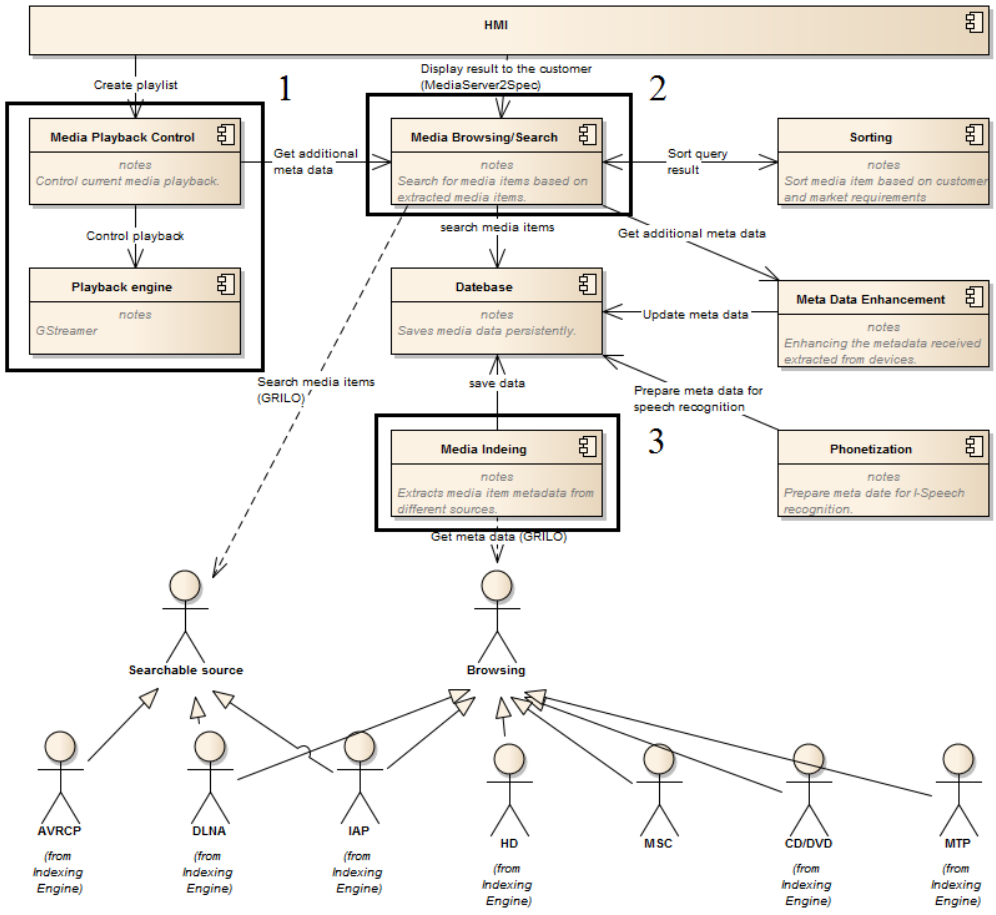
Svrha *Media Manager-a*, odnosno jednog zajedničkog aplikativnog programskog interfejsa za rukovođenje multimedijom, je da aplikacijama obezbijedi način da kontrolišu reprodukciju sadržaja tzv. povezanih *CE* (*Consumer Electronics*) uređaja na uniforman način. Na taj način aplikacije mogu podržati širok spektar raspoloživih uređaja/medija bez primjene posebnih rješenja za svaki od njih.

Media Manager je sačinjen od tri komponente:

- Medija kontrolera (eng. *Media Control*) pomoću kog se kontroliše reprodukcija sadržaja (na primjer, *play/pause* opcije),

- Medija indeksera, koji iz različitih izvora (udaljenih servera, *CD/DVD*) dobavlja metapodatke (npr. metapodaci audio sadržaja su sledeći: naziv pjesme, naziv izvođača, naziv albuma kom pjesma pripada, godina objavljivanja, fotografija, fotografija izvođača, žanr, tekst, itd.), koji se kasnije čuvaju u bazi podataka,
- Medija pretraživača koji vrši pretragu medijskog sadržaja.

Na slici 4 je prikazana arhitektura *Media Manager*-a, gdje je pravougaonikom broj 1 uokviren media kontroler, pravougaonikom broj 2 media pretraživač, a pravougaonikom broj 3 je istaknut media indekser.



Slika 4 Arhitektura *GENIVI Media Manager*-a

GENIVI projekat je još uvijek u procesu razvoja i usavršavanja, što znači da bi za izgradnju *IVI* sistema i prilagođavanje softvera, u cilju zadovoljavanja specifikacije originalnih proizvođača opreme na ovoj platformi, bilo potrebno daleko više napora i vremena u odnosu na neku od vodećih platformi na tržištu (*QNX* i *Windows Embedded Automotive*). Osim toga, korištenjem platforme otvorenog koda (eng. *open source*), proizvođači originalne opreme ostavljaju svoj proizvod dostupan konkureciji, što im svakako nije u cilju.

Imajući u vidu prethodno navedeno, kao i činjenicu da važnu ulogu pri odabiru platforme na kojoj će biti zasnovan *IVI* sistem igra cijena, kao i vrijeme koje je potrebno utrošiti na razvoj sistema, *GENIVI* ne predstavlja najbolji izbor.

2.3.2 Microsoft Windows Embedded Automotive

Windows Embedded Automotive [15] je operativni sistem namijenjen korištenju na računarskim sistemima u automobilima. Ovaj operativni sistem je prvi put isporučen u januaru 1998.godine, sa *AutoPC*-em (auto računar), koji su zajednički razvili *Microsoft* i *Clarion*.

Najznačajniji sistem zasnovan na *Windows Embedded Automotive* operativnom sistemu je *Ford Sync*. *Ford Sync* je fabrički instaliran integrisani sistem za komunikaciju i zabavu i vozilu, koji korisnicima omogućava upotrebu takozvanih *hands-free* (bez upotrebe ruku) telefonskih poziva, kontrolu muzike i obavljanje drugih funkcija upotrebom glasovnih komandi [16]. Sistem se sastoji od aplikacija i korisničkih interfejsa razvijenih od strane *Ford*-a. Prve dvije generacije ovog sistema (*Ford Sync* i *MiFord Touch*) kao osnovu koriste operativni sistem *Windows Embedded Automotive*, međutim treća generacija (*Sync 3*) se pokreće na *QNX* platformi.

Saradnja *Ford*-a i *Microsoft*-a datira još iz 2007. godine, kada je prvi put i predstavljen *Ford Sync* sistem, međutim 2014. godine dolazi do prekida saradnje zbog raznih problema (eng. *bugs*) koji su se javljali na sistemu čija je osnova *Windows Embedded Automotive*, a koji su značajno uticali na kvalitet *Ford* vozila [17]. Zbog toga *Ford* odlučuje da *Windows*-ov operativni sistem zamijeni *QNX*-om.

QNX sistem omogućava *Ford*-u da na jednostavniji način kreira *IVI* sistem sa krupnijim tekstom (fontovima) i ekranom osjetljivim na dodir koji omogućava kretanje kroz korisnički interfejs pokretima nadole, nagore ili poprijeko, kao i uvećavanje i smanjivanje prikazanog sadržaja. Krajnji rezultat je sistem koji je lakši i intuitivniji za korištenje, a koji istovremeno, zbog tradicionalista, zadržava i dugmiće.

U nastavku teksta su navedene samo neke osnovne prednosti koje *QNX* donosi *Sync*-u:

- *Sync 3* se bolje integriše sa *Apple*-ovim *iPhone*-om, što u prethodnim verzijama ovog sistema, zasnovanim na *Windows Embedded Automotive* operativnom sistemu, nije uvijek dobro funkcionisalo,
- Daleko bolje razumijevanje glasovnih komandi u odnosu na sistem zasnovan na *Windows*-ovom operativnom sistemu [18],
- Prelaskom na QNX platformu, Ford dobija zajednicu programera za punu podršku i ažuriranje softvera,
- *QNX* podržava *HTML5* i druge alate za korisničke interfejse.

2.3.3 QNX Car

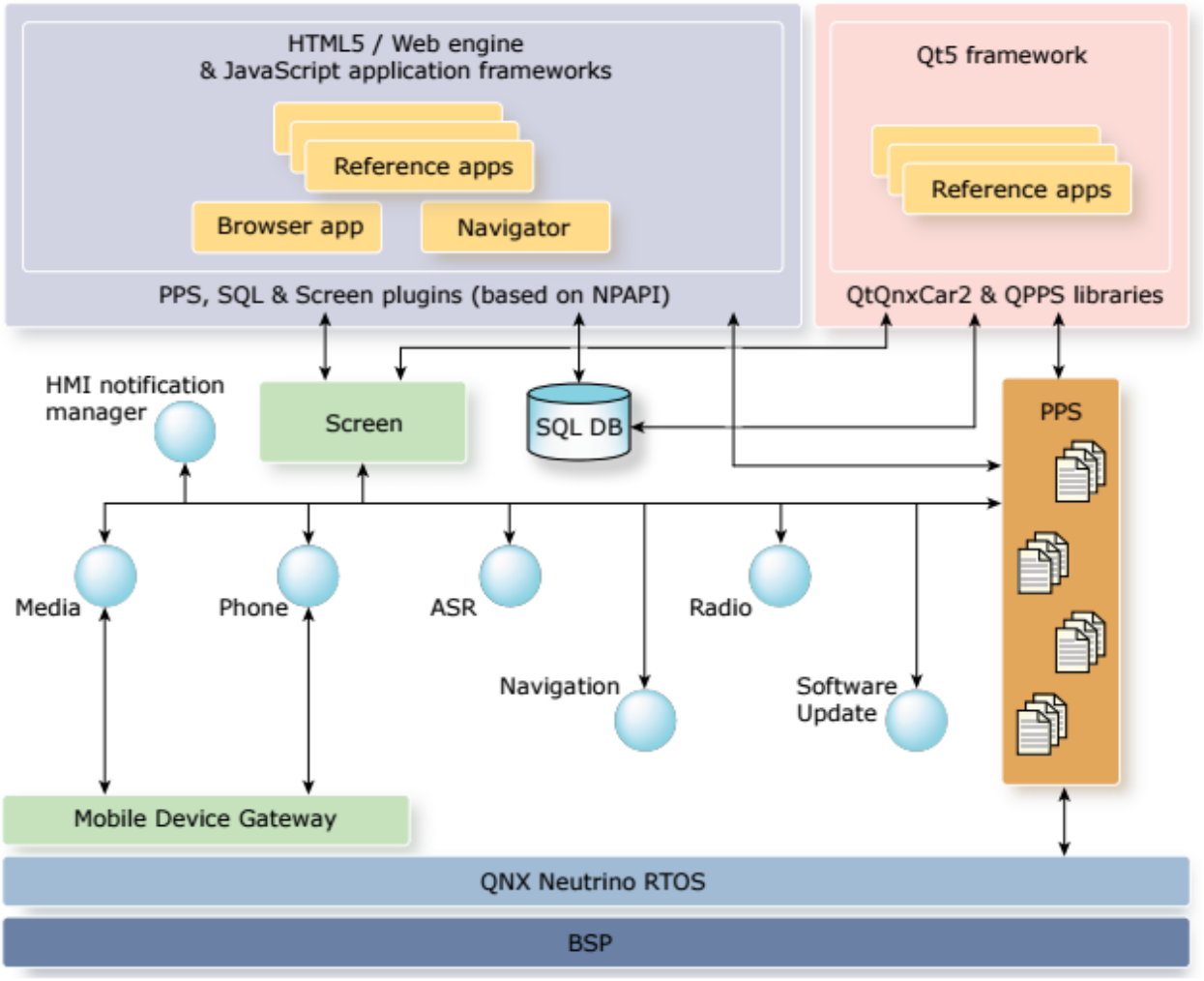
QNX Car (*QNX* auto) je specijalizovana platforma za *IVI* sisteme, sačinjena od skupa unaprijed integrisanih i optimizovanih tehnologija od strane *QNX Software Systems* kompanije i desetine njihovih partnera [19]. Ova platforma, dizajnirana sa ciljem da omogući fleksibilan razvoj *IVI* sistema, pruža razvojnim timovima raznovrsne mogućnosti za izgradnju pouzdanih *IVI* sistema vrhunskog kvaliteta, koji održavaju korak sa napretkom automobilske industrije i mobilnih uređaja na tržištu.

QNX Car platforma obuhvata širok spektar *QNX* tehnologija [20]:

- *QNX Neutrino realtime* operativni sistem,
- Moćnu strukturu za ostvarivanje konekcije (eng. *mobile connectivity framework*),
- Najkvalitetniju biblioteku za otklanjanje eha (eng. *ecoustic echo cancelation*) i smanjenje buke (eng. *noise reduction*),
- Okruženje za konfigurisanje i kontrolu multimedijalnih aplikacija (eng. *multimedia engine*),
- Okruženje za ostvarivanje veze čovjek-mašina (u daljem tekstu *HMI – Human Machine Interface*),
- Aplikacioni okvir koji podrzava *Qt*, *HTML5* i druge alate i aplikacije za razvoj korisničkog okruženja

Arhitektura *QNX Car* platforme, prikazana na slici 5, se sastoji iz tri sloja:

- Platforma,
- *Middleware* (srednji sloj),
- *HMI.*



Slika 5 Arhitektura *QNX Car* platforme

Prvi sloj, platformu, čine tri komponente:

- *QNX Neutrino RTOS,*
- *BSP*,
- Komponentu koja je zaduzena za upravljanje sistemskim nadogradnjama (eng. *Software Update component*).

Srednji sloj omogućava aplikacijama (*HTML5* ili *Qt5* aplikacije) da pristupe *IVI* servisima kao što su multimedija, *Bluetooth*, navigacija i radio. U daljem tekstu će, u nekoliko rečenica, biti predstavljeni neke komponente srednjeg sloja koje se mogu vidjeti na Slici 8.

Automatsko prepoznavanje govora (eng. *ASR – Automatic Speech Recognition*) pruža usluge ne samo za sloj iznad srednjeg sloja, već i za komponente ovog sloja. Na primjer, *ASR* se može koristiti za uključivanje radija, pokretanje *Bluetooth* telefonskog poziva ili reprodukciju video zapisa kojim upravlja multimedija.

Mobile Device Gateway komponenta omogućava povezivanje mobilnih uređaja, kroz *Bluetooth*, *DLNA*, *Wi-Fi* i druge mehanizme, na *QNX Car* platformu.

HMI Notification Manager, ažurirajući *HMI* na osnovu konfiguracione datoteke, reaguje na sistemske događaje kao što su ažuriranje (eng. *update*) navigacije ili dolazne tekstualne poruke.

Poslednji sloj arhitekture, *HMI*, se sastoji od nekoliko korisničkih aplikacija koje obezbjeđuju kontrolu podsistema kao što su *media player*, navigacioni servis i kontroler klime.

Ova verzija *QNX Car-a* sadrži i *Qt5* i *HTML5* verziju *HMI-a*. Podrazumijevano se koristi *Qt5 HMI*, ali moguće je konfigurisati *QNX Car* sistem da pri pokretanju podigne *HTML5 HMI*.

Browser engine (softver koji je obično ugrađen u Internet pregledač (eng. *web_browser*) i ima ulogu da prihvata sadržaj napisan u jezicima za obilježavanje (poput *HTML* ili *XML*) i zatim taj sadržaj prikazuje na ekranu ili štampaču), zasnovan na *Torch* internet pretraživaču, predstavlja okruženje za izvršavanje *HTML5* i *Qt5* aplikacija, dok im *NPAPI* dodaci (eng. *NPAPI plugins*) internet pretraživačima, učitani od strane *browser engine-a*, obezbjeđuju pristup servisima iz srednjeg sloja (na primjer, *SQL* dodatak pristupa bazama podataka koje čuvaju kontakt informacije). *HTML5* i *Qt5* aplikacije na različite načine pristupaju pomenutim dodacima. *HTML5* aplikacije za pristup *NPAPI* dodacima koriste *Apache Cordova* dodatke, implementirane u *JavaScript-u* (u daljem teksu *JS*), dok *Qt5* aplikacije mogu koristiti i *QtQnxCar2* i *QPPS* biblioteke. Prva biblioteka, *QtQnxCar2*, izlaže niz *C++* klasa koje obezbjeđuju kontrole za pristupanje servisima, i obezbjeđuje pristup bazama podataka pomoću *SQL* podrške. Druga biblioteka, *QPPS*, omogućava aplikacijama da koriste *QT-ovske* mehanizme signala i slotova. Aplikacije *Qt5 HMI-a* koriste obje biblioteke, osim *Media Player-a*, koji koristi još i *QPlayer* biblioteku za pretraživanje i reprodukciju medijskog sadržaja.

2.3.4 Android Auto

Android Auto [21] je infrastruktura koju je razvio *Google* sa ciljem da se pametnim telefonima sa *Android* operativnim sistemom (verzija 5.0 "*Lollipop*" i više), koji koriste *Android Auto* aplikaciju za mobilne uređaje, omogući da se povežu sa automobilom, uz korisnički interfejs prilagođen automobilu, i da prebaci aplikacije i sav sadržaj sa telefona na ekran na kontrolnoj tabli automobila koji podržava *Android Auto*. Ovaj standard omogućava vozačima da kontrolišu:

- *GPS* (eng. *Global Positioning System*; mape i navigaciju),
- Reprodukciju muzike – zvuk se šalje putem *USB*-a kako ne bi došlo do gubitka kvaliteta zvuka (za razliku od *Bluetooth*-a),
- *SMS* poruke (eng. *Short Message Service*),

- Telefoniju – telefonski pozivi se obrađuju preko *Bluetooth hands free* (bez upotrebe ruku) profila,
- Internet pretraživanje, itd.

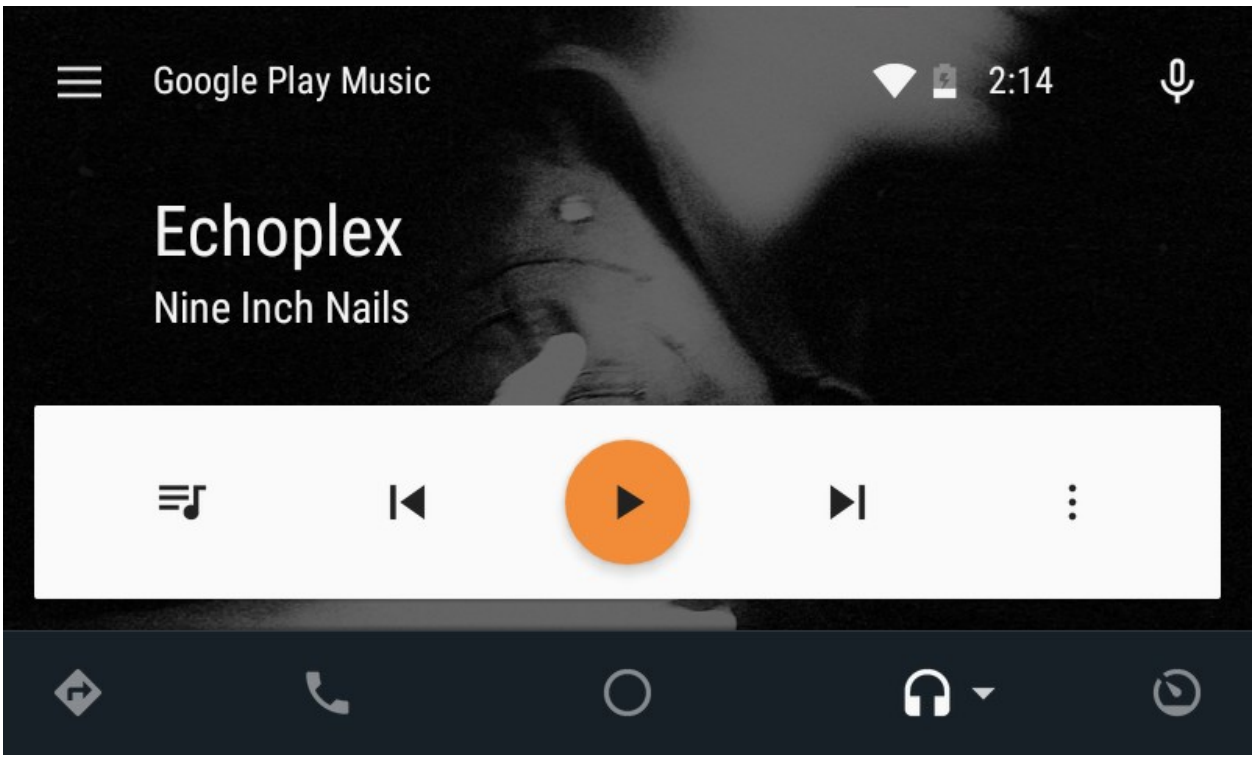
Cilj *Android Auto* tehnologije je proširivanje funkcionalnosti telefona na automobile, tako da glavni ekran u automobilu bude spoljni ekran (eng. external *display*) za telefon, s tim da prikaz sadržaja sa telefona bude prilagođen korisničkom okruženju tipičnom za automobile. Neke od aplikacija koje podržava *Android Auto* su sledeće [22]: *Google Maps*, *Google Play Music*, *Skype*, *Spotify*, *Amazon Music*, *WhatsApp Messenger*, *Facebook Messenger*, itd.

Sa *Android Auto* aplikacijom, mobilni uređaj vozača može pristupiti sledećim automobilskim ulazima i senzorima:

- *GPS* senzorima i visoko kvalitetnim GPS antenama,
- Kontrolama na volanu,
- Ozvučenju,
- Zvučnicima,
- Mikrofonu,
- Brzini vozila,
- Kompasu,
- Mobilnim antenama,
- Podacima automobila (u procesu razvijanja).

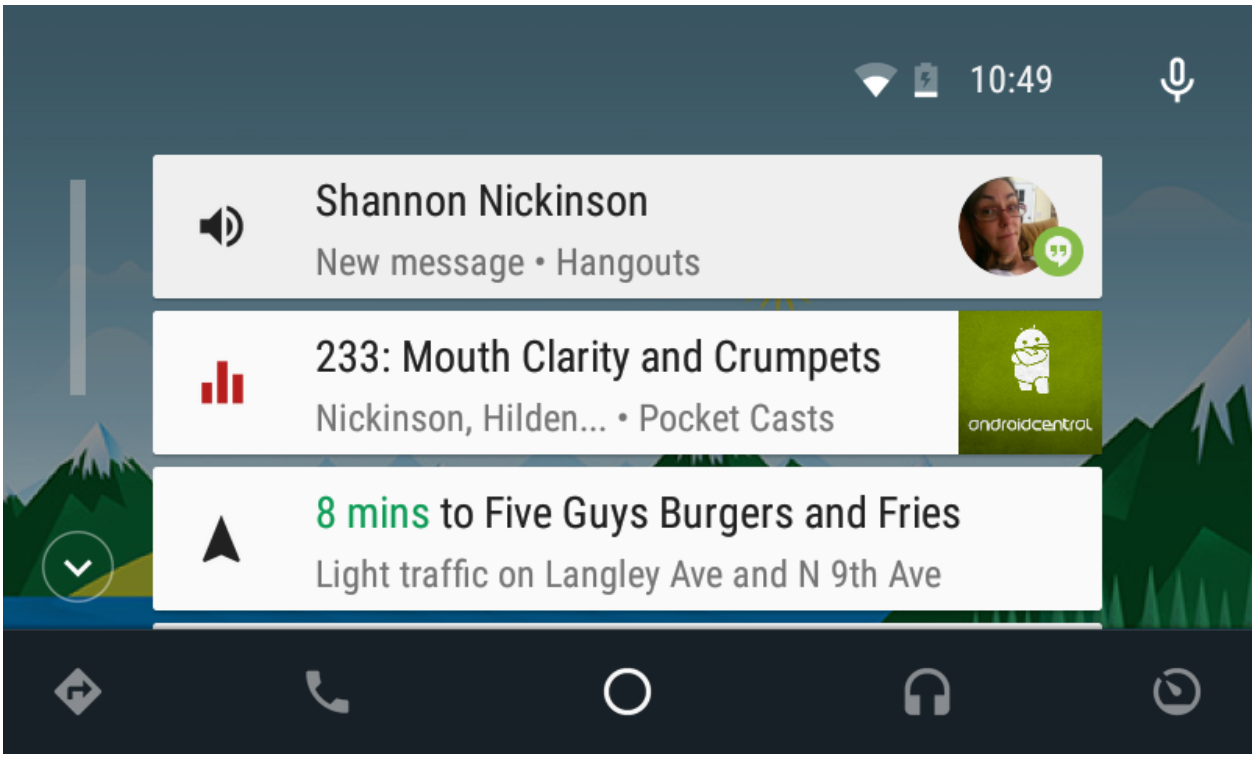
Neki od automobilskih brendova čiji modeli automobila podržavaju ili će uskoro podržavati *Android Auto* [23] su: *Alfa Romeo*, *Audi*, *Bentley*, *Cadillac*, *Chevrolet*, *Chrysler*, *Citroen*, *Fiat*, *Ford*, *Honda*, *Hyundai*, *Jeep*, *KIA*, *Lada*, *Mazda*, *Mercedes-Benz*, *Volkswagen*, itd.

Na slikama 6, 7 i 8 je prikazan izgled Android Auto aplikacije. Na slici 4 vidimo *Google Play Music* aplikaciju koja prikazuje naziv pjesme i izvođača, i nekoliko osnovnih dugmića za kontrolu.



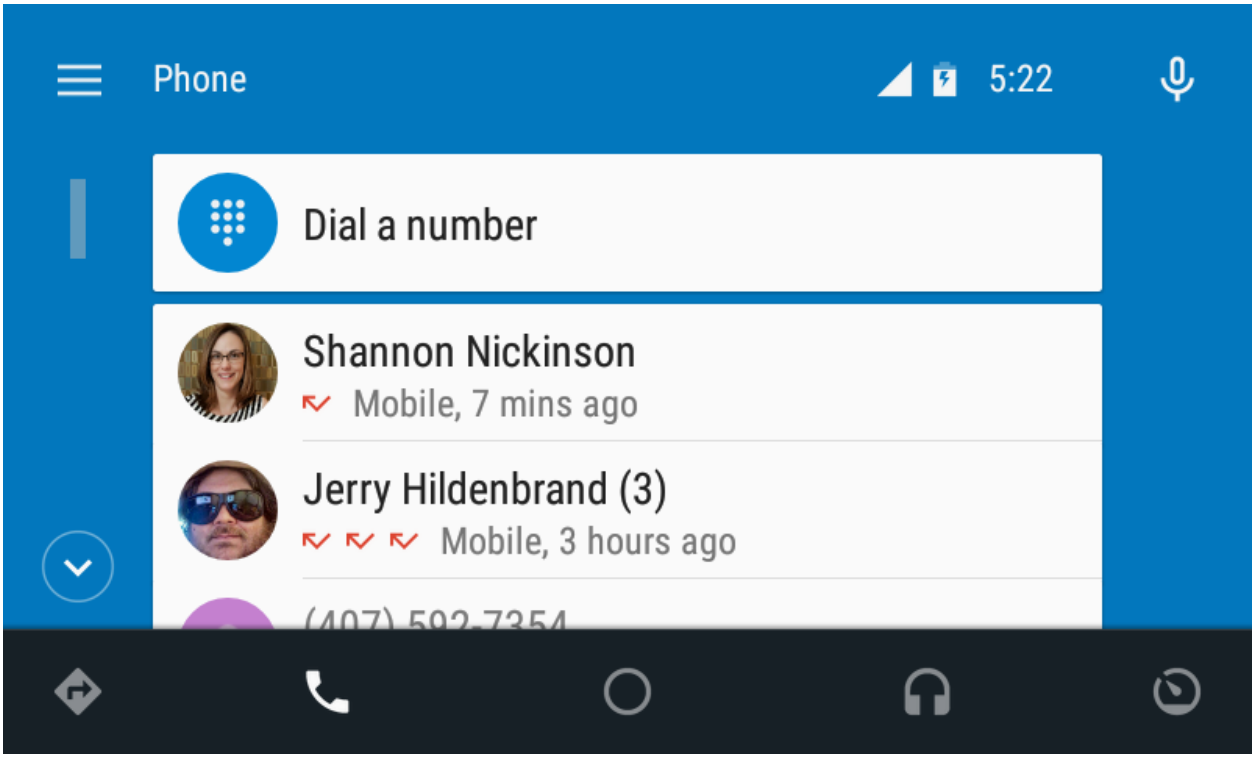
Slika 6 *Google Play Music* aplikacija

Slika 7 prikazuje univerzalnu pozadinu na kojoj se nalaze tri kartice koje prikazuju različit sadržaj.



Slika 7 Izgled univerzalne pozadine *Android Auto* aplikacije

Na slici 8 je prikazan izgled dnevnika poziva (eng. *call log*).



Slika 8 Izgled dnevnika poziva *Android Auto* aplikacije

Na prethodnim slikama se vidi da je dizajn *Android Auto* aplikacije prilično nezahtjevan. Razlog tome je jednostavan – *Google* želi da vozači većinu stvari obavljaju koristeći glasovne komande, kako ne bi skretali pogled sa puta.

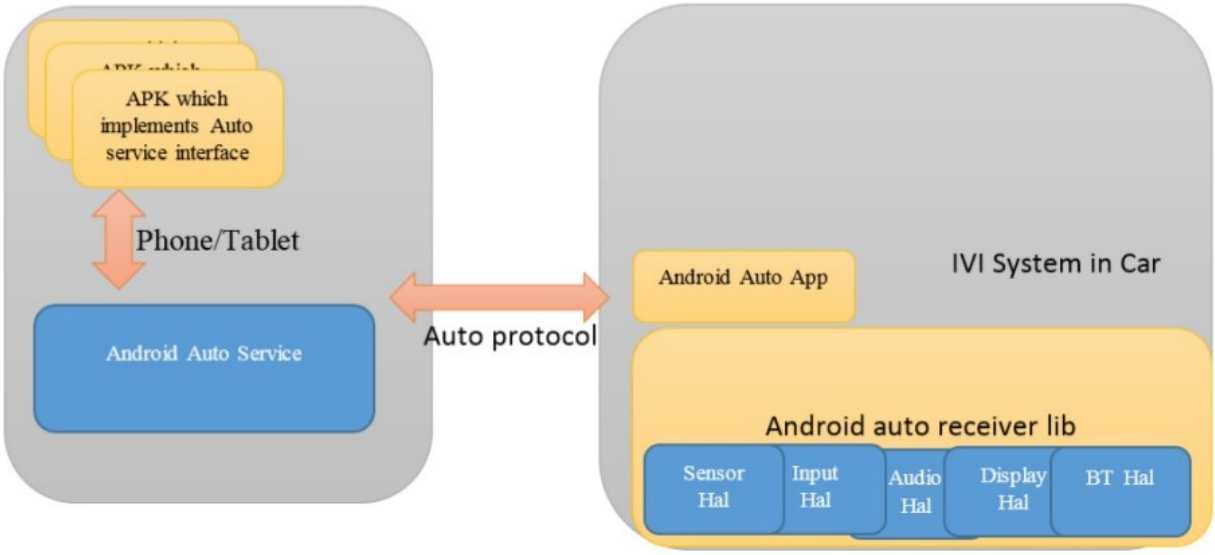
Na Slici 9 je prikazana arhitektura *Android Auto* softvera [24]. Lijevi blok sa slike predstavlja *Android* telefon/tablet, na kome se nalaze:

- *Android Auto Service*,
- *Google Play Service* (nije prikazano na slici ali ulazi u sastav softvera),
- Aplikacije koje koriste funkcionalnosti *Android Auto Service* komponente,
- *Android Auto SDK* (takođe nije prikazano na slici).

Desni blok sa slike predstavlja *IVI* sistem u automobilima, u kom se nalazi *Android Auto Service* biblioteka u kojoj su implementirane sledeće *hal* (*Hardware Abstraction Layer*) komponente:

- Ulaz (eng. input),
- *Audio*,
- Senzor podaci,
- *Bluetooth*,
- Ekran.

Ova dva bloka povezuje *Auto protocol* koji je sačinjen od pet različitih kanala preko kojih telefon komunicira sa gore navedenim aspektima automobila.



Slika 9 Arhitektura *Android Auto* aplikacije

2.3.5 Car Play

Apple CarPlay [25] proizvođačima automobila nudi mogućnost da komplikovane *IVI* sisteme zamijene displejem koji komunicira sa *iPhone* telefonom. Ovo rješenje nije sistem u automobilu na kom se pokreće *iOS* (operativni sistem) ili *iOS* aplikacije, ovo rješenje samo prebacuje *iOS* okruženje na displej *IVI* sistema, što omogućava korisnicima da kontrolišu ne samo aplikacije već i uređaj, kako preko displeja tako i pomoću glasovnih komandi. Ideja *CarPlay* aplikacije jeste da se omogući da se poznato *iOS* korisničko okruženje prenese na *IVI* ekrane, kao i da se omogući njegova kontrola koristeći sve dostupne kontrole automobila, tako da korisnicima bude omogućeno da koriste sve *Apple* aplikacije (na primjer da slušaju *Apple iTune*, da koriste *Apple* navigaciju *Apple Mapps*, sve funkcionalnosti telefona itd.) Ostale aplikacije, kao što su *Pandora*, *iHeartRadio*, *NPR News*, su takođe dostupne kroz *CarPlay* [26].

Među proizvođačima automobila koji podržavaju ovu tehnologiju su *Audi*, *Honda*, *Hundai*, *Kia*, *Volkswagen*, najnoviji modeli *Mercedes* automobila, itd.

Na Slici 10 je prikazan izgled *CarPlay* aplikacije u automobilu. Prikaz na ekranu kontrolne table automobila je prilagođen tipičnom *Apple* okruženju.

Slika 10 *Car Play* aplikacija

Android Auto i *CarPlay* su zapravo konkurencija jedan drugom, i nikako ne mogu u trku sa *QNX*-om, *Windows Embedded Automotive*-om i *GENIVI*-em, ili nekom drugom platformom za *IVI* sisteme, ali s obzirom da predstavljaju jedan način prikazivanja multimedijalnih sadržaja u automobilima, važno ih je pomenuti, i odrediti njihovo mjesto u odnosu na prethodno navedene platforme (zapravo samo u odnosu na *QNX*, jer se rješenje opisano u ovom radu oslanja na ovaj operativni sistem).

QNX je zapravo osnova *Apple*-ovog *CarPlay*-a, a što se tiče *Google*-ovog *Android Auto*-a, koji je prilagođen uređajima koji koriste *Google*-ov *Android*, *QNX* podržava *Android* aplikacije, pa samim tim i *Android Auto*.

2.4 Upotreba hipervizora u navedenim tehnologijama za predstavljanje multimedije u automobilima

U ovom potpoglavlju će biti predstavljena uloga hipervizora (eng. *hypervisor*), a takođe će biti izložena i tabela njegove zastupljenosti u navedenim tehnologijama za predstavljanje multimedije u automobilima.

Uloga hipervizora je sledeća – hipervizor izoluje operativne sisteme i aplikacije od osnovnog hardvera. Ova izolacija omogućava hardveru da samostalno upravlja jednom ili više virtuelnih mašina (svaka od ovih virtuelnih mašina ili operativnih sistema može da pokrene sopstvene programe), omogućavajući im na taj način da efikasno dijele fizičke resurse sistema

(hipervizor im dodjeljuje resurse), kao što su procesorski ciklusi, memorijski prostor, mrežni propusni opseg, itd. Dakle, hipervizor omogućava postojanje nekoliko virtuelnih mašina koje optimalno rade na jednom računarskom hardveru, čime se značajno poboljšava upotreba tog hardvera [27].

U tabeli 2 je dat pregled podrške za hipervizor u operativnim sistemima koji se koriste u pomenutim tehnologijama za predstavljanje multimedije u vozilima [28][29].

Operativni sistem	Podrška za hipervizor
QNX	
Windows Embedded Automotive	
Linux	

Tabela 2 Pregled podrške za hipervizor

Podrškom za hipervizor, proizvođačima sistema za automobile, se omogućava da na jednom računarskom hardveru integrišu *IVI* sistem i aplikaciju za kontrolnu tablu (da nema hipervizora *IVI* sistem i aplikacija za kontrolnu tablu ne bi mogli da se integrišu na jedan računarski hardver, jer se u *ISO26262* standardu navodi da se na istom operativnom sistemu ne smiju razviti aplikacije/sistemi koji podliježu *ASIL* klasifikaciji i oni koji ne podliježu [30]), i na taj način smanje složenost hardvera i cijenu, a istovremeno pruže bogato korisničko iskustvo koje današnji korisnici zahtijevaju.

2.5 Zaključak

Softver koji se koristi u *IVI* sistemima predstavlja najjače područje konkurencije među proizvođačima originalne opreme. Efikasan razvoj *IVI* aplikacija i njihovog korisničkog okruženja je od ključnog značaja za komercijalni uspijeh platforme specijalizovane za razvoj *IVI* sistema u automobilima.

U tabeli 2 je prikazana zastupljenost platformi na tržištu [31].

Platforma	Zastupljenost u *IVI* sistemima [%]
QNX	44
Microsoft	17
Linux	4
Drugi	35

Tabela 3 Zastupljenost platformi u *IVI* sistemima

U konstantnoj borbi za prevlast u svijetu *IVI* sistema i automobilske industrije, *QNX* vodi glavnu riječ, sa oko 44 procenata svjetskog tržišta (oko 60 miliona *IVI* sistema u automobilima je zasnovano na *QNX*-u). Nakon *QNX*-a slijedi *Microsoft Windows Embedded Automotive* platforma koja drži oko 17 procenata tržišta, a tu je i nekoliko *Linux* platformi od kojih je najznačajniji *GENIVI* (4 procenta tržišta). Ostatak, oko 35 procenata svjetskog tržišta pripada svim ostalim platformama za razvoj *IVI* sistema.

Uzimajući u obzir sve što je navedeno u prethodnim potpoglavljima, za razvoj rješenja koje će biti predstavljeno u nastavku teksta je izabran *QNX* operativni sistem, koji predstavlja osnovu na kojoj će biti izgrađen srednji sloj sistema i *HUD* i *Cluster* aplikacije (biće pominjane u narednom poglavlju). Pored *QNX*-a, u sklopu rješenja će biti i *Linux*, na kom će u finalnom rješenju (u radu je opisano stanje sistema nakon prve faze razvoja) biti samo *CID* aplikacija. Dakle, sva logika i suština sistema će biti na sigurnijem operativnom sistemu (*QNX*-u) odakle će se slati statusne poruke *Linux*-u na osnovu kojih će se izvršavati određene akcije na toj strani. Ovakvom arhitekturom se postiže da dizajn bude usklađen sa *ISO26262* standardom, koji navodi da softver *Cluster* aplikacije (*ASIL B*) ne smije zavisiti od softvera koji ne podliježe *ASIL* klasifikaciji.

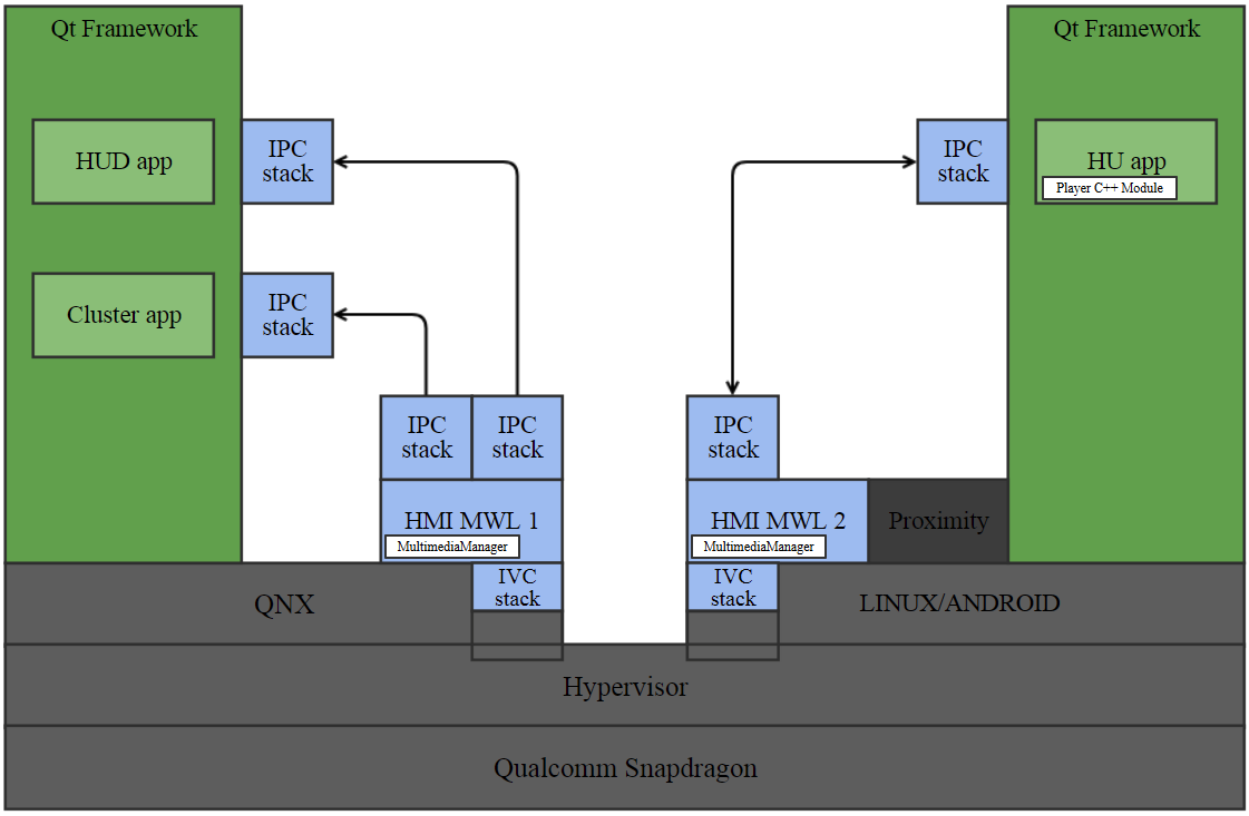
Hipervizor razvijen za *Qualcomm*-omov *Snapdragon* čip će se koristiti za kontrolu ova dva operativna sistema i razmjenu informacija između njih.

3. Koncept rješenja

Da bi koncept i implementacija *multimedia player*-a bili jasni, potrebno je proći kroz osnovne elemente sistema u kom je *player* realizovan. U ovom poglavlju biće predstavljeni elementi koji čine arhitekturu sistema u kom je realizovan *player*, kao i osnovni koncept multimedijalne aplikacije opisane u ovom radu.

3.1 Arhitektura sistema

Na Slici 11 je predstavljena arhitektura sistema. Bijelom bojom na slici, radi bolje uočljivosti, su predstavljene komponente sistema koje se odnose na *player* koji je tema ovog rada (*player C++ module* i dva *MultimediaManager*-a (jedan na *Linux* strani, a drugi na *QNX*)).



Slika 11 Arhitektura sistema opisanog u ovom radu

HU aplikacija (*Head Unit App*) je jedna od tri polazne tačke u smislu interakcije sa korisnikom (druga je senzor za blizinu, a treća je tastatura). Ova aplikacija prima korisničke akcije preko ekrana osjetljivog na dodir, i šalje informacije preko *IPC* steka na srednji sloj (u nastavku će biti objašnjen *IPC* stek). Takođe, ova aplikacija je odgovorna za prikaz informacija korisnicima.

Player C++ Module predstavlja realizaciju *player*-a u okviru *HU* aplikacije, a njegova uloga je da vrijednosti proslijeđene od audio dijaloga (dio *HU* aplikacije relevantan za upotpunjavanje priče o *player*-u), proslijedi srednjem sloju na *Linux*-u (*HMI MWL 2*).

Cluster i *HUD* (*Head Unit Display*) aplikacije ne podržavaju akcije od strane korisnika nego se koriste samo za prikaz informacija korisniku (informacija o brzini, preostaloj količini goriva, pređenom rastojanju, itd.).

Sve tri aplikacije su razvijene u *Qt* razvojnom okviru (eng. *Qt Framework*).

HMI MWL (*Human Machine Interface Middleware*), srednji sloj softvera, je kontrolni sloj. *Middleware* dobija korisničke ulaze od *proximity* senzora (senzor za blizinu), ekrana osjetljivog na dodir (eng. *touch screen*) i tastature, donosi odluke i inicira akcije kao što je prikazivanje nekih informacija korisniku, koordiniranje pri dijeljenju ekranskih elemenata preuzimanje podataka iz simulacione baze podataka, itd.

MultimediaManager na *Linux*-u, na osnovu vrijednosti dobijenih od *player* modula, formira poruku koja se, preko *IVC* steka, šalje srednjem sloju na *QNX* operativnom sistemu. *MultimediaManager* na drugoj strani, po prijemu poruke od srednjeg sloja na *Linux*-u, izvršava zadate akcije:

- Započinje/zaustavlja reprodukciju audio/video sadržaja,
- Šalje *Cluster* aplikaciji obavještenje o trenutno reprodukovanoj pjesmi kako bi on prikazao odgovarajući poster,
- U slučaju video sadržaja, šalje *Cluster* aplikaciji poruku da izbuši rupu kako bi se reprodukcija nastavila na ekranu *Cluster*-a (po zaustavljanju reprodukcije šalje poruku o tome, kako bi *Cluster* popunio prethodno izbušenu rupu).

Komunikacioni stek se koristi za ostvarivanje komunikacije između procesa (*HMI MWL* (*Linux* i *QNX*), *Cluster*, *HUD*, *HU*). Sastoji se od dvije komponente prenosa (*IPC* i *IVC*) i poruka koje se razmjenjuju. *IPC* stek (*Inter-process Communication Stack*) je grupa modula koji implementiraju transformaciju podataka (pakovanje objekata poruke u niz bajtova i obrnuto, raspakivanje bajtnih nizova u objekte poruke) i prenos podataka između različitih procesa koji se pokreću unutar istog operativnog sistema (*HMI MWL 1* komunicira preko *IPC*-a sa *Cluster* i *HUD* aplikacijom, a *HMI MWL 2* komunicira preko *IPC* steka sa *HU* aplikacijom). *IVC* stek je sličan *IPC* steku, ali je namijenjen za komunikaciju između različitih procesa koji se pokreću unutar različitih operativnih sistema (srednji slojevi međusobno komuniciraju preko *IVC* steka).

Poruke su klase koje imaju mogućnost da se transformišu u niz bajtova, a takođe se kreiraju iz niza bajtova. Koriste se za komunikaciju između različitih procesa koristeći *IPC* ili *IVC* za prenos podataka i za komunikaciju između različitih komponenti unutar istog procesa korištenjem povratne mehanike (eng. *callback*) ili redova (eng. *message queues*)za prenos podataka.

QNX je *ISO26262* (međunarodni standard za funkcionalnu sigurnost električnih i/ili elektronskih sistema u proizvodnim automobilima, koji je 2011.godine definisala Međunarodna organizacija za standardizaciju – *ISO*) *ASIL D* sertifikovani *RTOS*, sa *ASIL B* sertifikovanim grafičkim stekom.

Linux operativni sistem se koristi kao manje siguran operativni sistem za svrhe *HU* aplikacije.

Hipervizor (eng. *hypervisor*) razvijen za *Qualcomm SnapDragon* čip upravlja operativnim sistemima (*QNX* i *Linux*) i razmjenom podataka između njih.

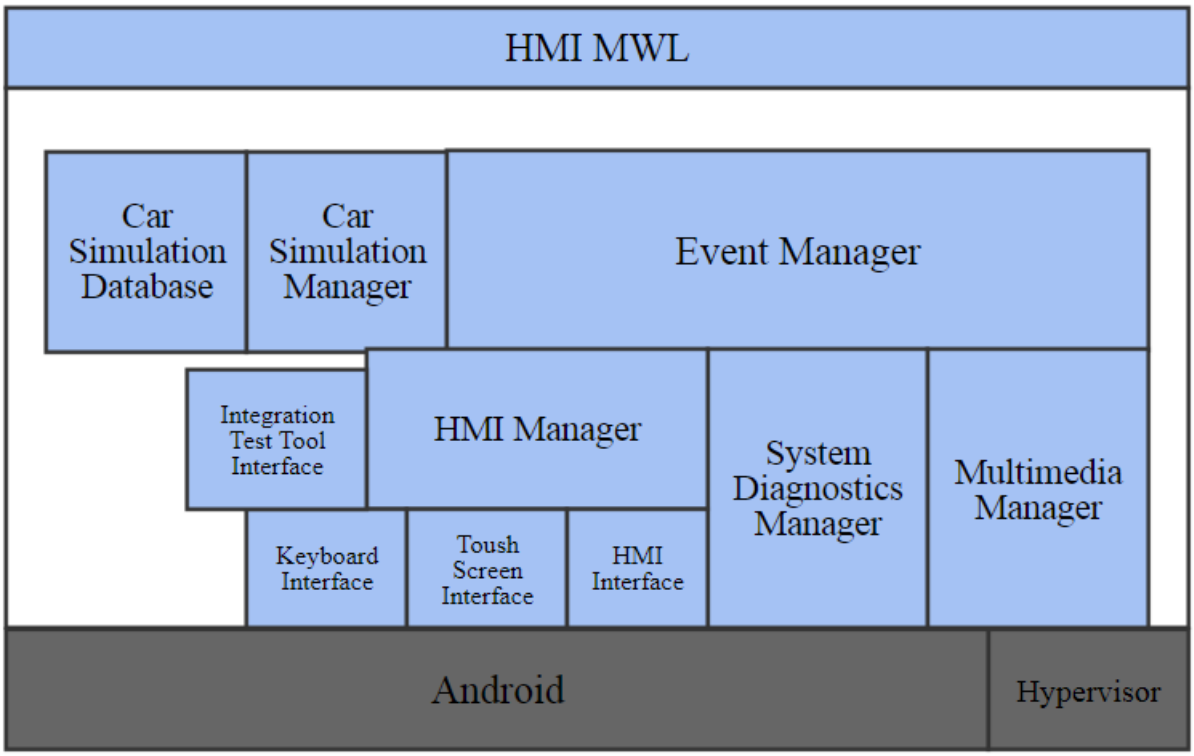
3.2 Srednji sloj

Srednji sloj je odgovoran za rukovanje ulazima sa više uređaja (trenutno podržava ekran osjetljiv na dodir, senzor blizine i tastaturu), iniciranje akcije za prikazivanje informacija korisniku i koordiniranje pri dijeljenju ekranskih elemenata. Za komunikaciju sa drugim procesima se oslanja na *IPC* i *IVC* stek.

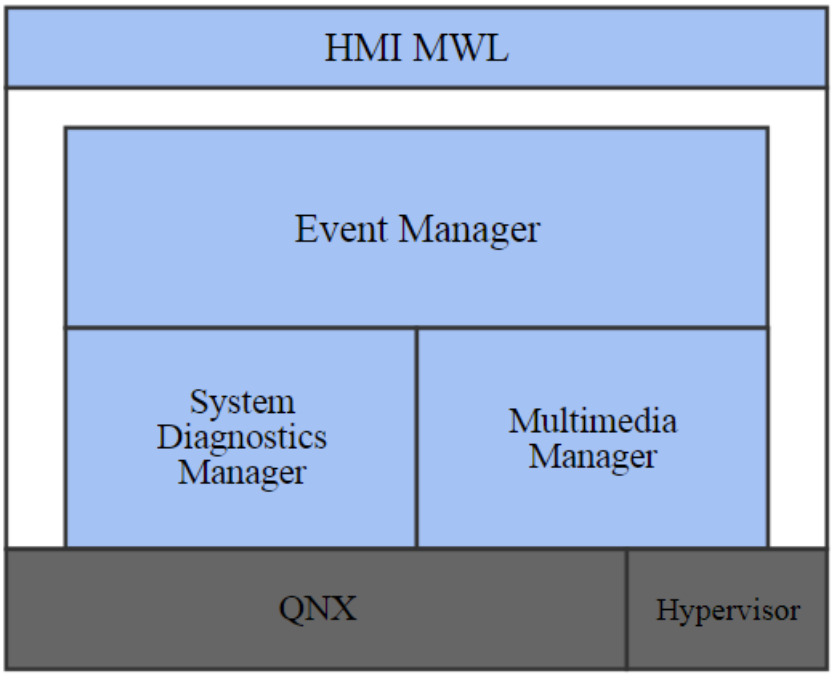
Srednji sloj je samostalan proces koji predstavlja jezgro softvera koji se opisuje u ovom radu. U nastavku poglavlja će ukratko biti predstavljeni arhitektura i dizajn, koji se sastoji od nekoliko modula:

- *Event Manager* – kontrolni modul srednjeg sloja. Ovaj modul uzima ulazne podatke od drugih modula i koristi ih za delegiranje posla za različite procese,
- *Multimedia Manager* – koristi se za reprodukciju audio i video sadržaja, i za dijeljenje ekrana između *HU* i *HUD*/*Cluster* aplikacija. Ovaj modul se u velikoj mjeri oslanja na funkcionalnosti hipervizora,
- *HMI Manager* – koristi se za preuzimanje informacija od ulaznih uređaja (ekran osjetljiv na dodir, senzor blizine, tastatura, itd.),
- *System Diagnostics Manager* – koristi se za dobijanje sistemskih informacija (iskorištenost procesora i operativne memorije),
- *Car Simulation Manager and Database* – se koristi za simulaciju nekih aktivnosti podržanih od strane srednjeg sloja.

Na slikama 12 i 13 su prikazani srednji slojevi arhitekture sistema.



Slika 12 Srednji sloj na *Android*-u



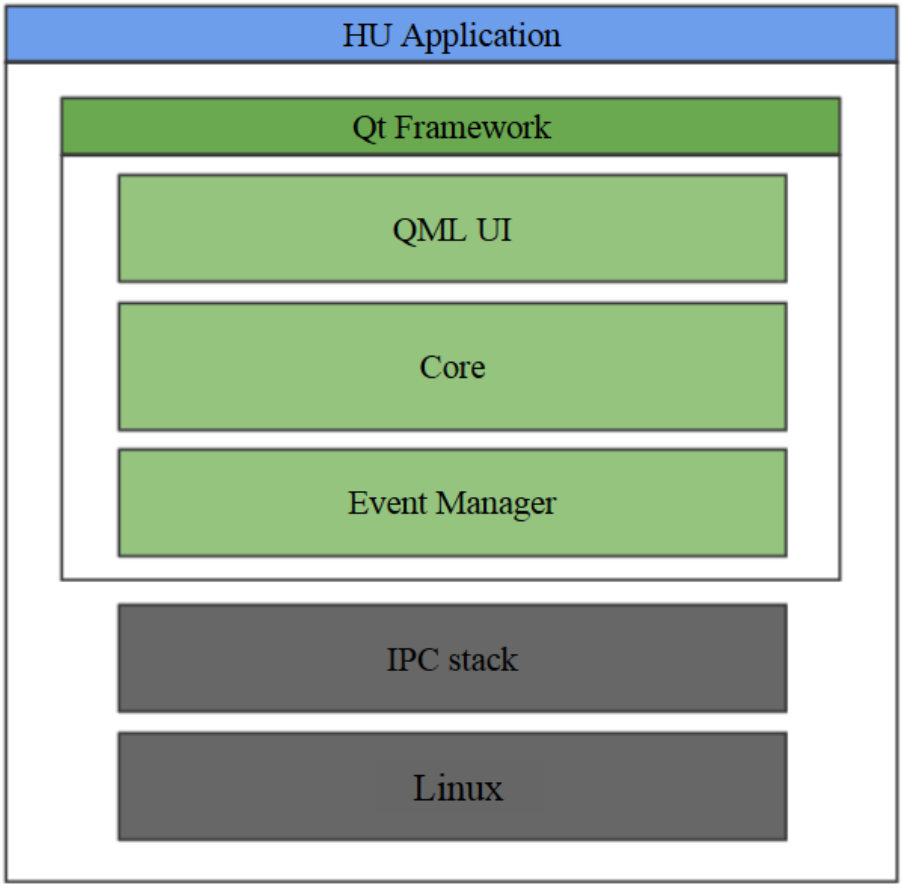
Slika 13 Srednji sloj na *QNX* operativnom sistemu

Srednji slojevi opisani u ovom poglavlju se pokreću kao samostalni procesi na bilo kom operativnom sistemu (*QNX*, *Linux*, *Android*). Implementirani su u *C++*-u, jer je *C++* standardni

način za pisanje *Android*/*Linux* sistemskih servisa/procesa (važi i za *QNX*), a takođe čini srednji sloj prenosivim na bilo koji operativni sistem.

3.3 HU aplikacija

HU aplikacija je centralna jedinica interakcije korisnika sa sistemom. Ta interakcija se ostvaruje kroz samu aplikaciju, pomoću ekrana osjetljivog na dodir, i pomoću senzora blizine koji reaguje na mahanje korisnika. Na Slici 14 je prikazan dijagram dizajna aplikacije.



Slika 14 Dijagram dizajna aplikacije

Osnovni dio aplikacije predstavljaju tri modula unutar *Qt* razvojnog (*Qt Framework*) okvira:

- *Qml*,
- *Core*,
- *Event Manager*.

Qml je deklarativni jezik koji omogućava opis korisničkih okruženja u smislu njihovih vizuelnih komponenti i načina interakcije i odnosa tih komponenti. Dakle, pri izradi *HU* aplikacije, *Qml* se koristi za implementaciju elemenata, dijaloga, kontrola i obavještenja

korisničkog okruženja. U ovom modulu su smješteni svi resursi koji izgrađuju korisničko okruženje, kao i konkretne implementacije dijaloga korisničkog okruženja koje koriste pomenute resurse. Korisnički dijalozi *HU* aplikacije su sledeći:

- Dijalog sa aplikacijama,
- Dijalog sa podešavanjima klime,
- Navigacija,
- Telefon,
- Dijalog za reprodukciju muzičkog sadržaja,
- Kamera dijalog,
- Podešavanja,
- Dijalog sa informacijama vezanim za vozilo.

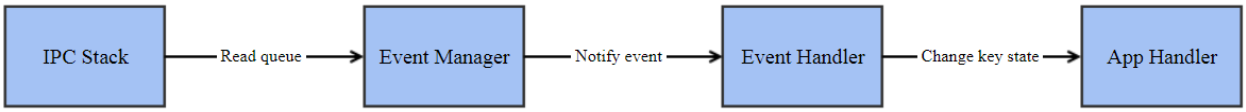
Core predstavlja glavni komunikacioni modul aplikacije. U ovom modulu je implementiran modul za slanje i prijem poruka od srednjeg sloja (komunikacija sa srednjim slojem je od ključnog značaja za reagovanje aplikacije na korisničke akcije).

Modul nazvan *Event Manager* obrađuje poruke koje aplikacija šalje *IPC* steku, kao i poruke iz suprotnog smjera, od *IPC*-a ka aplikaciji.

U nastavku će biti opisan tok podataka kroz *HU* aplikaciju u nekoliko slučajeva.

3.3.1 Primicanje ruke ekranu osjetljivom na dodir

Kada senzor blizine detektuje korisnikovu ruku, svi dugmići na *HU* aplikaciji postanu označeni. Na sici 15 je prikazan tok podataka kroz *HU* aplikaciju.

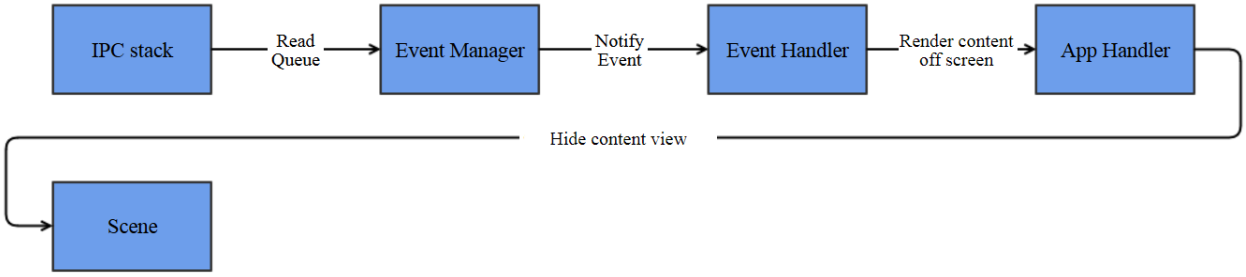


Slika 15 Prikaz postera trenutno reprodukovane pjesme na ekranu *Cluster*-a – tok podataka

Kada *IPC* stek primi poruku, *Event Manager* je pročita iz reda i šalje obavještenje *Event Handler*-u, koji tu informaciju šalje aplikaciji kao slot. Aplikacija na slot reaguje promjenom izgleda svih trenutno vidljivih dugmića.

3.3.2 Prebacivanje sadržaja sa ekrana *HU* aplikacije na ekran *Cluster* aplikacije

Kada korisnik mahne rukom ulijevo, dok se na *HU* aplikaciji prikazuje video, sadržaj koji se trenutno prikazuje se prebacuje na *Cluster* ekran (i obrnuto za mahanje udesno). Tok podataka unutar *HU* aplikacije je prikazan na slici 16.

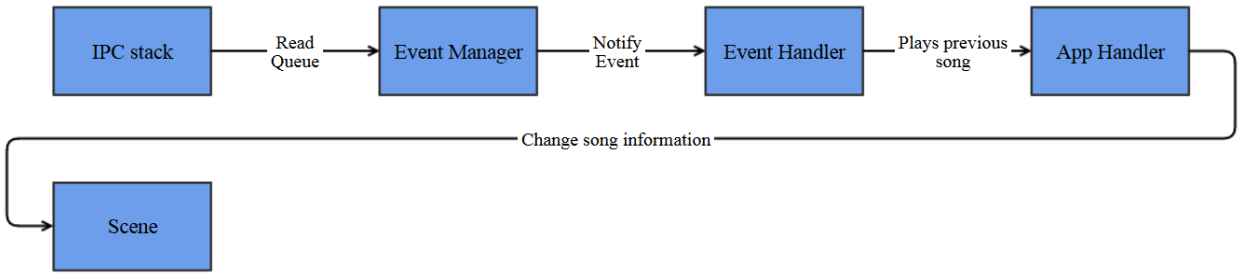


Slika 16 Prebacivanje sadržaja sa *HU* aplikacije na ekran *Cluster*-a – tok podataka

Prilikom prebacivanja sadržaja sa ekrana *HU* aplikacije na ekran *Cluster* aplikacije, *HU* aplikacija sklanja prikazivani sadržaj sa ekrana, i započinje renderovanje sadržaja u pozadini, u predefinisanoj dijeljenoj memoriji kojoj drugi sistemi mogu pristupati. Istovremeno se na *Cluster* ekranu nastavlja reprodukcija.

3.3.3 Prikazivanje postera trenutno reprodukovane pjesme na *Cluster* aplikaciji

Prilikom reprodukovanja audio sadržaja na *HU* aplikaciji, poster pjesme koja se trenutno reprodukuje se prikazuje na *Cluster* aplikaciji. Promjenom pjesme, mijenja se i poster. Na slici 17 je prikazan tok podataka, kroz *HU* aplikaciju, za navedeni slučaj.

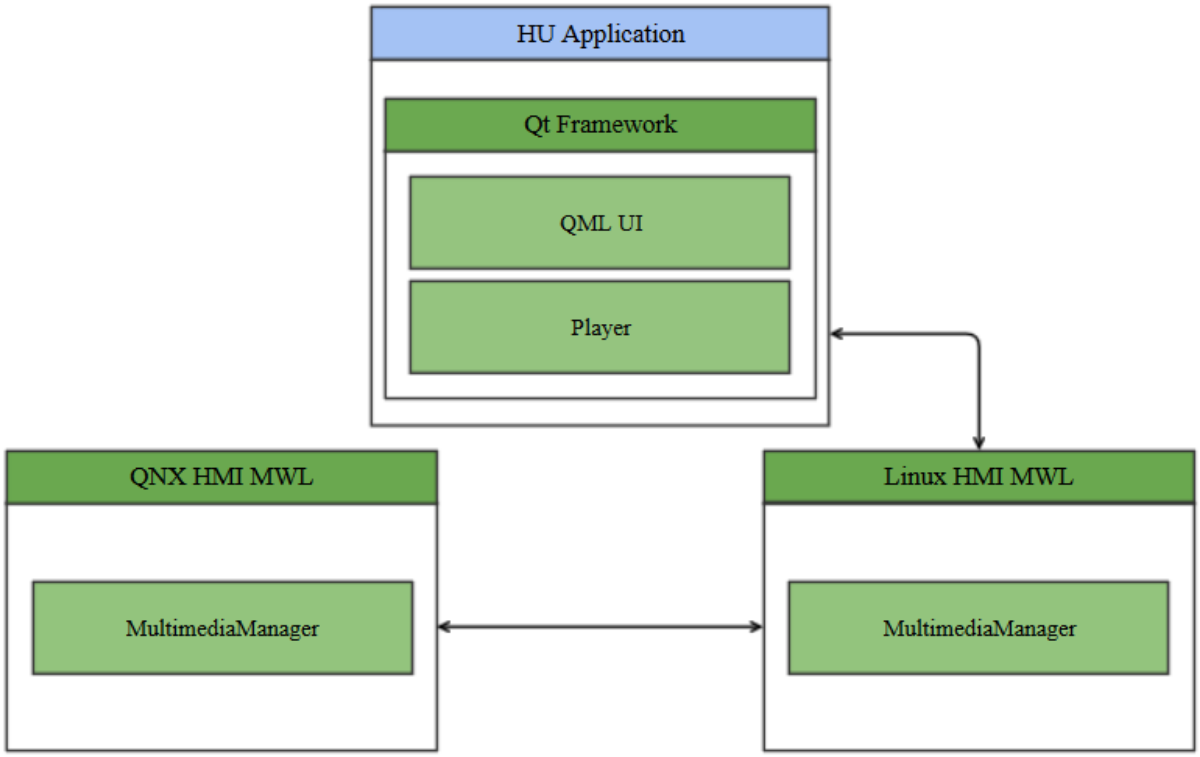


Slika 17 Prikaz postera trenutno reprodukovane pjesme na ekranu *Cluster*-a – tok podataka

Prilikom promjene pjesme, na ekranu *HU* aplikacije se mijenjaju informacije o pjesmi, i istovremeno se šalje obavještenje *Cluster* aplikaciji da prikaže odgovarajući poster.

3.4 Player

Ovo poglavlje opisuje arhitekturu realizovanog *player*-a, prikazanu na slici 18.



Slika 18 Arhitektura realizovanog *player*-a

Osnovni dijelovi player-a su sledeći:

- *Player C++* modul – razvijen u okviru *HU* aplikacije,
- *MultimediaManager* na *Linux HMI MWL* strani i
- *MultimediaManager* na *QNX HMI MWL* strani.

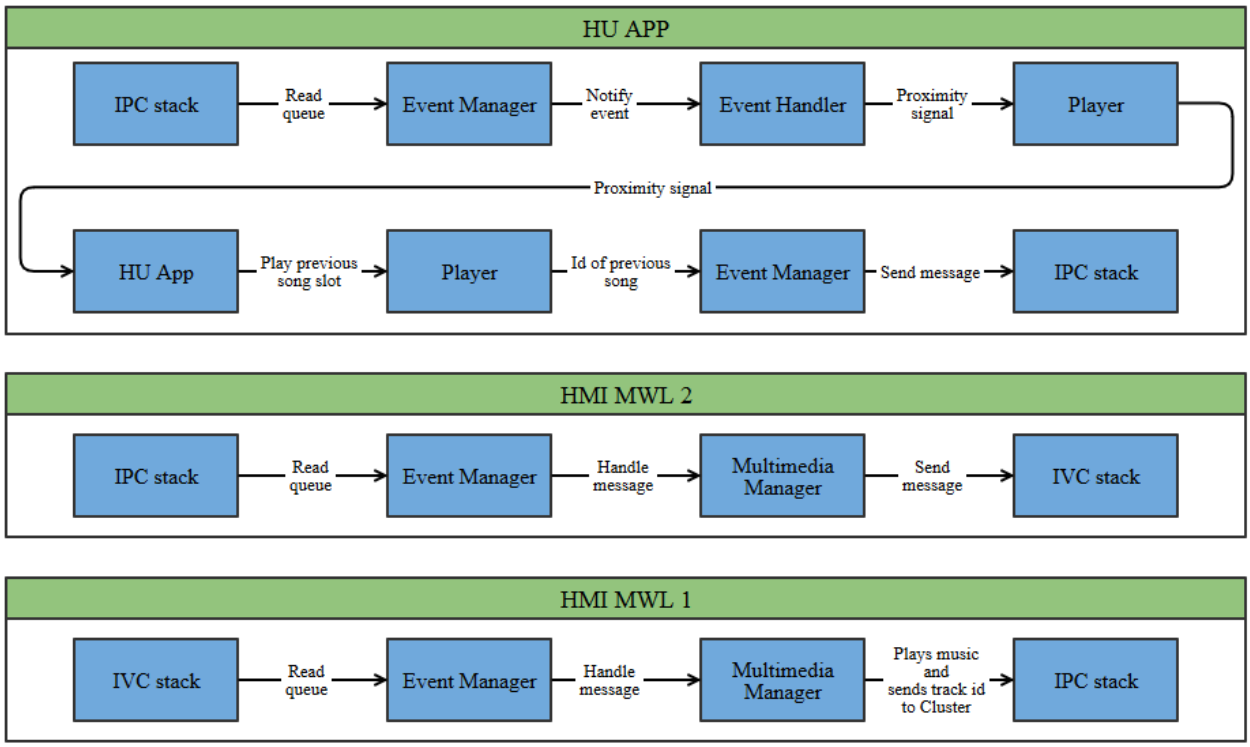
Player C++ modul prima signale od audio dijaloga kada se desi neka korisnička akcija koja izaziva manipulaciju reprodukcijom sadržaja, a zatim, preko *IPC*-a, dobijene vrijednosti prosleđuje *MultimediaManageru* na *Linux* strani.

MultimediaManager na *Linux* strani, preko *IVC*-a, prosleđuje dobijene podatke *MultimediaManager*-u na *QNX* strani, koji na osnovu dobijenih podataka započinje ili zaustavlja reprodukciju (koristi se *QPlayer*), a istovremeno, ukoliko je potrebno, šalje informacije *Cluster* i *HUD* aplikaciji o eventualnim aktivnostima (u slučaju audio sadržaja, *Cluster*-u se šalje broj

postera koji treba da prikaže, a u slučaju videa se, po potrebi, i *Cluster* i *HUD* aplikaciji šalje znak da izbuše rupu za prikaz video sadržaja).

Dakle, *player C++* modul, preko signala i slotova, komunicira sa audio dijalogom, i prosleđuje informacije potrebne za manipulaciju reprodukcije sredne sloju, jer se reprodukovanje, zbog nemogućnosti puštanja video/audio sadržaja na *Linux*-u, trenutno vrši na *QNX* strani.

U nastavku poglavlja će biti predstavljen tok informacija od *player* modula do *MultimediaManager*-a (slika 19), za slučaj prebacivanja pjesme gestovnom kontrolom, uz prikaz razmjene informacija (princip je isti i za pokretanje/zaustavljanje reprodukcije audio sadržaja).



Slika 19 Prebacivanje pjesme gestovnom kontrolom – tok podataka

Kada korisnik mahne rukom udesno, dok je reprodukcija audio sadržaja u toku, *HU* aplikacija prima *proximity* poruku koju prosleđuje player modulu. *Player* modul šalje signal audio dijalogu sa sadržajem *proximity* poruke. U ovom slučaju, audio dijalog će primiti poruku o tome da se desilo mahanje ulijevo, a zatim će vratiti *player*-u broj pjesme koja treba da se pusti, i istovremeno prilagoditi izgled dijaloga prethodnoj pjesmi iz liste. *Player* prosleđuje dobijenu informaciju srednjem sloju na *Linux*-u, koji je dalje prosleđuje srednjem sloju na *QNX* operativnom sistemu. Srednji sloj na *QNX*-u prosleđuje poruku *MultimediaManager*-u, koji će pustiti prethodnu pjesmu, i poslati redni broj trenutno reprodukovane pjesme *Cluster* aplikaciji, kako bi se promijenio poster na ekranu *Cluster* aplikacije.

4. Implementacija rješenja

U ovom poglavlju će se govoriti o realizovanim scenama audio dijaloga, realizaciji komunikacije između audio dijaloga i *player*-a, i na kraju poglavlja je dat uvid u realizaciju samog *player*-a. Na ovaj način se daje uvid u sve komponente audio plejera predstavljenog u ovom radu, i zaokružuje priča o plejeru za multimediju u vozilima. Važno je napomenuti da se *player* opisivan u ovom radu koristi i za reprodukovanje video sadržaja, ali je kao referentan primjer uzet audio dijalog, jer je njegova realizacija u ovom projektu nešto kompleksnija.

4.1 Scene

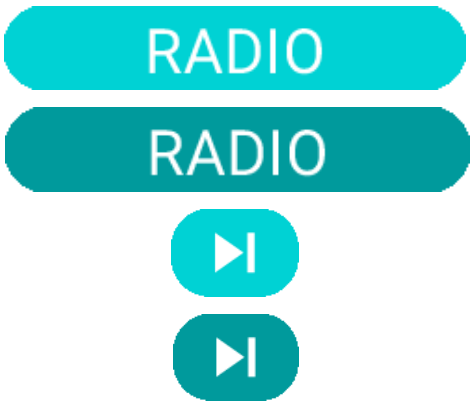
Na klik dugmeta za aktivaciju audio dijaloga u *HU* aplikaciji, podiže se prva scena audio dijaloga i započinje reprodukcija sadržaja.

Kompletan izgled audio dijaloga je realizovan u okviru jednog *QML* fajla, u kom su smješteni svi elementi koji grade cjelokupan izgled dijaloga sa koordinatama i definisanim načinom reagovanja na određene akcije, kao i dio logike za realizaciju funkcionalnosti dijaloga. S obzirom da je ukupno osam scena realizovano unutar istog fajla (četiri scene kada senzor blizine ne detektuje korisnikovu ruku, i četiri scene kada korisnik približi ruku ekranu *HU* aplikacije), bilo je potrebno osmisliti logiku koja će omogućiti prikazivanje elemenata scene koja je trenutno otvorena i, istovremeno, skrivanje svih elemenata koji nisu dio te scene. Da bi se ovo postiglo, korištena su stanja. Dakle, audio dijalog se, s obzirom na broj scena, u jednom trenutku može naći u jednom od osam stanja, u zavisnosti od toga koja scena se prikazuje i da li je senzor za blizinu detektovao korisnikovu ruku ili ne. Ta stanja, odnosno scene, su sledeća:

1. ***MEDIA_MAIN_SCENE_NORMAL*** – osnovna scena audio dijaloga gdje se vrši reprodukcija i kontrola reprodukcije, od strane korisnika, audio sadržaja koji se nalazi u arhivi iz koje se vrši reprodukcija (izvori reprodukovanja sadržaja mogu biti različiti, a biće navedeni u tabeli BROJ),
2. ***MEDIA_MAIN_SCENE_NEAR*** – izgled osnovne scene audio dijaloga kada korisnik približi ruku ekranu *HU* aplikacije,
3. ***MEDIA_EXPANDED_SCENE_NORMAL*** – proširena osnovna audio scena, u kojoj se ispod osnovne scene nalazi lista svih dostupnih pjesama,
4. ***MEDIA_EXPANDED_SCENE_NEAR*** – proširena osnovna scena audio dijaloga, kada senzor blizine detektuje korisnikovu ruku,
5. ***RADIO_MAIN_SCENE_NORMAL*** – radio scena,
6. ***RADIO_MAIN_SCENE_NEAR*** – izgled radio scene kada korisnik približi ruku ekranu,
7. ***RADIO_EXPANDED_SCENE_NORMAL*** – proširena radio scena, gdje se, ispod radio scene, nalazi lista radio stanica,
8. ***RADIO_EXPANDED_SCENE_NEAR*** – proširena radio scena kada je detektovan prilazak ruke ekranu.

Dodatak *normal*/*near* na kraju naziva stanja se odnosi na stanje u kom se nalaze dugmad scene. Dakle, *normal* scene (kada senzor ne detektuje prilazak) se u odnosu na *near* scene (kada senzor detektuje prilazak ekranu) razlikuju samo u izgledu dugmadi scene. Na slikama 20 i 21 je prikazan izgled nekoliko dugmadi u *normal* stanju, i izgled istih u *near* stanju (prvo dugme – *normal* stanje, drugo dugme – *near* stanje).

Slika 20 Izgled dugmadi u *normal* i *near* stanju



Slika 21 Izgled dugmadi u *normal* i *near* stanju

Kako bi bilo jasno na koji način funkcionišu i kako se definišu stanja u jednoj *Qt* aplikaciji, u nastavku će biti predstavljena realizacija *normal* i *near* stanja audio dijaloga, na primjeru osnovne, *MEDIA_MAIN*, audio scene. Analiziranjem ovog primjera postaće jasno na koji način je izvršena manipulacija elementima dijaloga. Na slici 22 je prikazana realizacija stanja osnovne audio scene kada senzor blizine ništa ne detektuje, a na slici 23 je dat prikaz realizacije stanja iste scene, ali kada senzor detektuje prilazak ruke ekranu *HU* aplikacije (senzor se nalazi ispod ekrana osjetljivog na dodir, pa je prilazak ruke senzoru zapravo prilazak ruke ekranu).

```
State {
    name: "MEDIA_MAIN_SCENE_NORMAL"
    PropertyChanges { target: mediaBase;            state:   "ACTIVE"; y: mediaExpandedBase.height/2 - 106 }
    PropertyChanges { target: mediaExpandedBase;    state:   "INACTIVE" }
    PropertyChanges { target: positionTime;         state:   "ACTIVE"; y: 430 + mediaBase.height - 200 }
    PropertyChanges { target: durationTime;         state:   "ACTIVE" }
    PropertyChanges { target: progressBar;          state:   "ACTIVE"; y: 470 + mediaBase.height - 200 }
    PropertyChanges { target: expandArrowButton;    state:   "NORMAL"; y: mediaBase.height - 200 }
    PropertyChanges { target: compressArrowButton;  visible: false }
    PropertyChanges { target: radioButton;          state:   "NORMAL"; y: 80 + mediaBase.height - 200 }
    PropertyChanges { target: mediaButton;          state:   "NORMAL"; y: 80 + mediaBase.height - 200 }
    PropertyChanges { target: appsButton;           state:   "NORMAL"; y: 80 + mediaBase.height - 200 }
    PropertyChanges { target: discButton;           state:   "NORMAL" }
    PropertyChanges { target: usbButton;            state:   "NORMAL" }
    PropertyChanges { target: iPodButton;           state:   "NORMAL" }
    PropertyChanges { target: bluetoothButton;      state:   "NORMAL" }
    PropertyChanges { target: auxButton;            state:   "NORMAL" }
    PropertyChanges { target: allTrackButton;       state:   "NORMAL" }
    PropertyChanges { target: folderButton;         state:   "NORMAL" }
    PropertyChanges { target: albumButton;          state:   "NORMAL" }
    PropertyChanges { target: artistButton;         state:   "NORMAL" }
    PropertyChanges { target: artistsImage;         state:   "ACTIVE"; source: ipodImages[currentMediaId]
                                                                       y: height - 60 + mediaBase.height - 200 }
    PropertyChanges { target: menuButton;           state:   "NORMAL"; y: mediaBase.height - 350 + mediaBase.height - 200 }
    PropertyChanges { target: albumNameInfo;        state:   "ACTIVE"; x: 280 ; y: 230 + mediaBase.height - 200 }
    PropertyChanges { target: dummy1Image;          state:   "ACTIVE"; x: albumNameInfo.x; y: albumNameInfo.y + albumNameInfo.height + 20 }
    PropertyChanges { target: songsNameInfo;        state:   "ACTIVE"; x: dummy1Image.x + dummy1Image.width + 20; y: dummy1Image.y + 8 }
    PropertyChanges { target: dummy2Image;          state:   "ACTIVE"; x: albumNameInfo.x; y: dummy1Image.y + dummy1Image.height + 20 }
    PropertyChanges { target: artistsNameInfo;      state:   "ACTIVE"; }
    PropertyChanges { target: previousSongButton;   state:   "NORMAL"; x: folderButton.x; y: folderButton.y - height - 30 }
    PropertyChanges { target: playButton;           state:   "NORMAL"; imageSource: playPauseSource() }
    PropertyChanges { target: nextSongButton;       state:   "NORMAL" }
    PropertyChanges { target: fmButton;             visible: false }
    PropertyChanges { target: amButton;             visible: false }
    PropertyChanges { target: sxmButton;            visible: false }
    PropertyChanges { target: ahaButton;            visible: false }
    PropertyChanges { target: pandoraButton;        visible: false }
    PropertyChanges { target: presetListButton;     visible: false }
    PropertyChanges { target: stationListButton;    visible: false }
    PropertyChanges { target: timeShiftButton;      visible: false }
    PropertyChanges { target: bookmarkButton;       visible: false }
    PropertyChanges { target: radioStationInfo;     state:   "INACTIVE" }
    PropertyChanges { target: frequencyInfo;        state:   "INACTIVE" }
    PropertyChanges { target: dummy3Image;          state:   "INACTIVE" }
    PropertyChanges { target: numbers;              state:   "INACTIVE" }
    PropertyChanges { target: trackList;            state:   "INACTIVE" }
    PropertyChanges { target: radioGrid;            state:   "INACTIVE" }
    PropertyChanges { target: leftButton;           visible: false }
    PropertyChanges { target: rightButton;          visible: false }
},
```

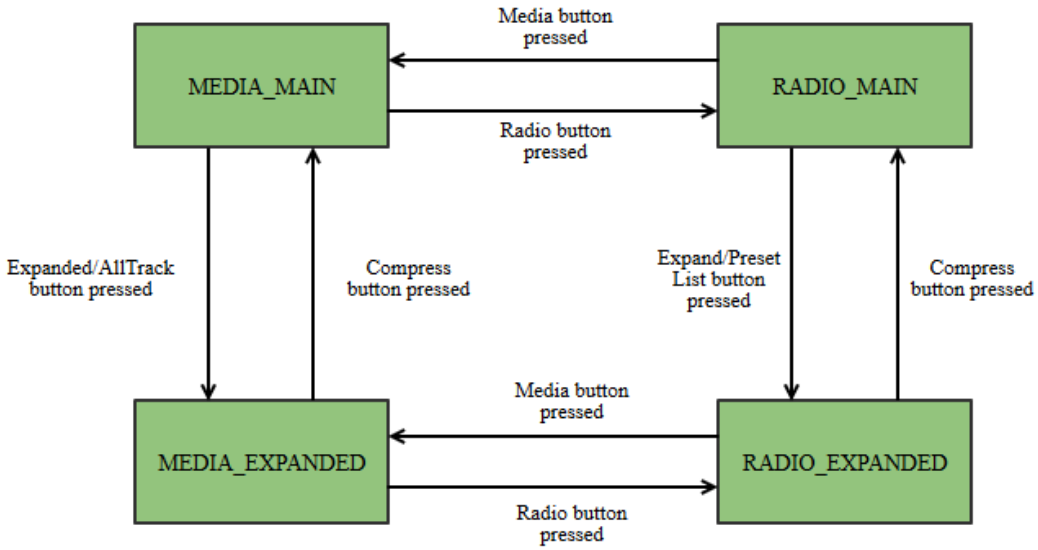
Slika 22 Realizacija *normal* stanja osnovne audio scene

```
State {
    name: "MEDIA_MAIN_SCENE_NEAR"
    PropertyChanges { target: mediaBase;            state:   "ACTIVE"; y: mediaExpandedBase.height/2 - 106 }
    PropertyChanges { target: mediaExpandedBase;    state:   "INACTIVE" }
    PropertyChanges { target: positionTime;         state:   "ACTIVE"; y: 430 + mediaBase.height - 200 }
    PropertyChanges { target: durationTime;         state:   "ACTIVE" }
    PropertyChanges { target: progressBar;          state:   "ACTIVE"; y: 470 + mediaBase.height - 200 }
    PropertyChanges { target: expandArrowButton;    state:   "NEAR";   y: mediaBase.height - 200 }
    PropertyChanges { target: compressArrowButton;  visible: false }
    PropertyChanges { target: radioButton;          state:   "NEAR";   y: 80 + mediaBase.height - 200 }
    PropertyChanges { target: mediaButton;          state:   "NEAR";   y: 80 + mediaBase.height - 200 }
    PropertyChanges { target: appsButton;           state:   "NEAR";   y: 80 + mediaBase.height - 200 }
    PropertyChanges { target: discButton;           state:   "NEAR" }
    PropertyChanges { target: usbButton;            state:   "NEAR" }
    PropertyChanges { target: iPodButton;           state:   "NEAR" }
    PropertyChanges { target: bluetoothButton;      state:   "NEAR" }
    PropertyChanges { target: auxButton;            state:   "NEAR" }
    PropertyChanges { target: allTrackButton;       state:   "NEAR" }
    PropertyChanges { target: folderButton;         state:   "NEAR" }
    PropertyChanges { target: albumButton;          state:   "NEAR" }
    PropertyChanges { target: artistButton;         state:   "NEAR" }
    PropertyChanges { target: artistsImage;         state:   "ACTIVE"; source: ipodImages[currentMediaId]
                                                                       y: height - 60 + mediaBase.height - 200 }
    PropertyChanges { target: menuButton;           state:   "NEAR";   y: mediaBase.height - 350 + mediaBase.height - 200 }
    PropertyChanges { target: albumNameInfo;        state:   "ACTIVE"; x: 280 ; y: 230 + mediaBase.height - 200 }
    PropertyChanges { target: dummy1Image;          state:   "ACTIVE"; x: albumNameInfo.x; y: albumNameInfo.y + albumNameInfo.height + 20 }
    PropertyChanges { target: songsNameInfo;        state:   "ACTIVE"; x: dummy1Image.x + dummy1Image.width + 20; y: dummy1Image.y + 8 }
    PropertyChanges { target: dummy2Image;          state:   "ACTIVE"; x: albumNameInfo.x; y: dummy1Image.y + dummy1Image.height + 20 }
    PropertyChanges { target: artistsNameInfo;      state:   "ACTIVE"}
    PropertyChanges { target: previousSongButton;   state:   "NEAR";   x: folderButton.x; y: folderButton.y - height - 30 }
    PropertyChanges { target: playButton;           state:   "NEAR";   imageSource: playPauseSource() }
    PropertyChanges { target: nextSongButton;       state:   "NEAR" }
    PropertyChanges { target: fmButton;             visible: false }
    PropertyChanges { target: amButton;             visible: false }
    PropertyChanges { target: sxmButton;            visible: false }
    PropertyChanges { target: ahaButton;            visible: false }
    PropertyChanges { target: pandoraButton;        visible: false }
    PropertyChanges { target: presetListButton;     visible: false }
    PropertyChanges { target: stationListButton;    visible: false }
    PropertyChanges { target: timeShiftButton;      visible: false }
    PropertyChanges { target: bookmarkButton;       visible: false }
    PropertyChanges { target: radioStationInfo;     state:   "INACTIVE" }
    PropertyChanges { target: frequencyInfo;        state:   "INACTIVE" }
    PropertyChanges { target: dummy3Image;          state:   "INACTIVE" }
    PropertyChanges { target: numbers;              state:   "INACTIVE" }
    PropertyChanges { target: trackList;            state:   "INACTIVE" }
    PropertyChanges { target: radioGrid;            state:   "INACTIVE" }
    PropertyChanges { target: leftButton;           visible: false }
    PropertyChanges { target: rightButton;          visible: false }
},
```

Slika 23 Realizacija *near* stanja osnovne audio scene

Analiziranjem slika 22 i 23 se vidi da se prelaskom iz *normal* scene u *near*, i obrnuto, mijenja stanje dugmadi, takođe iz *normal* u *near* stanje, i obrnuto, pri čemu svi elementi zadržavaju svoje pozicije, vidljivost i resurse koji su im dodijeljeni (moguće je upravljati vidljivošću, pozicijom, resursima, stanjima, tekstom (kad je tekstualni element u pitanju) elemenata, ali to se u ovom slučaju ne radi jer je u pitanju potpuno isti izgled dijaloga, ali sa različitim stanjima dugmadi – *near* i *normal*).

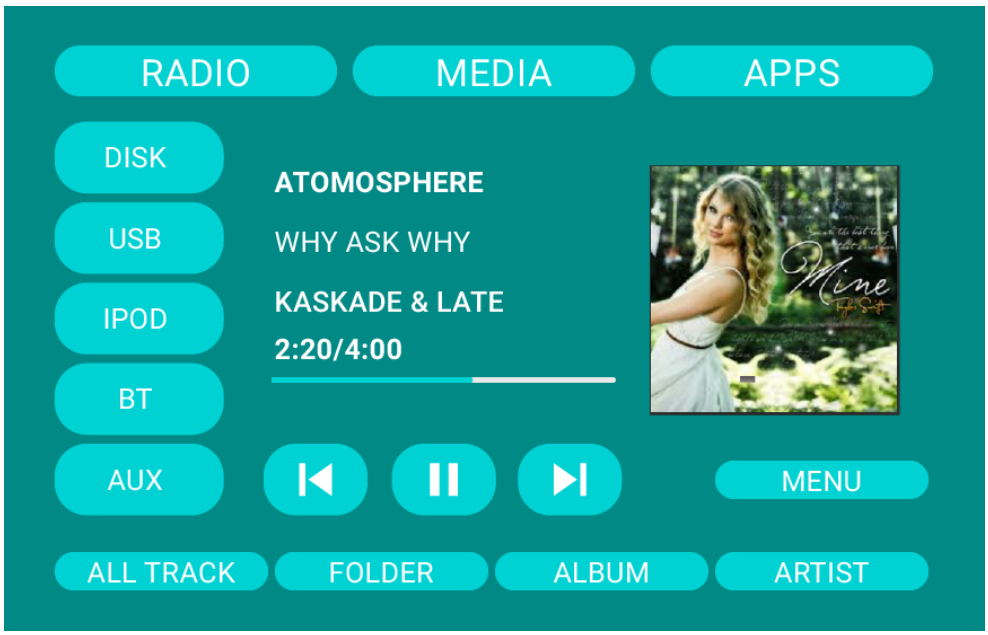
Na slici 24 je predstavljen tok prelaska iz scene u scenu. U primjeru na slici su prikazana četiri *normal* stanja (podrazumijevano je da se detektovanjem prilaska ekranu prelazi u trenutno stanje, ali sa *near* dugmadima).



Slika 24 Prelazak iz scene u scenu

Iz osnovne audio scene se, pritiskom dugmeta u gornjem lijevom uglu scene ili dugmeta „*All Track*“, prelazi u proširenu audio scenu koja, pored osnovnih infromacija o trenutno reprodukovanom sadržaju, prikazuje i listu svih pjesama na trenutnom izvoru reprodukcije. Iz ove scene se, pritiskom „*Radio*“ dugmeta, prelazi u proširenu radio scenu, odakle se pritiskom dugmeta u gornjem lijevom uglu te scene prelazi u osnovnu radio scenu. Iz osnovne radio scene se prelazi u osnovnu audio scenu na pritisak „*Media*“ dugmeta. Što se tiče suprotnog smjera (unutrašnje strelice sa slike), pritiskom „*Radio*“ dugmeta se, iz osnovne audio scene, prelazi u osnovnu radio scenu. Iz osnovne radio scene se, pritiskom dugmeta u gornjem lijevom uglu scene ili dugmeta „*Preset List*“, prelazi u proširenu radio scenu, odakle se, pritiskom „*Media*“ dugmeta, prelazi u proširenu osnovnu audio scenu. I na kraju, iz proširene osnovne audio scene se, pritiskom dugmeta u gornjem lijevom uglu te scene, prelazi na osnovnu audio scenu.

Na slici 25 je prikazan izgled osnovne audio scene.



Slika 25 Izgled osnovne scene audio dijaloga

4.2 Signali i slotovi – tok informacija

Kako bi funkcionalnosti audio dijaloga, na primjer zaustavljanje i ponovno pokretanje reprodukcije audio sadržaja, bile realizovane, implementiran je poseban modul u *C++*-u, nazvan *player* (ovaj *player* je takođe korišten za reprodukciju video sadržaja). U ovom modulu su definisane funkcije, tzv. slotovi, koje reaguju na signale emitovane od strane audio dijaloga kao reakcija na neki događaj. U ovom poglavlju će najprije biti opisan redoslijed operacija od trenutka kada se pritisne neko dugme audio dijaloga, do trenutka kada se desi definisana reakcija, a zatim će biti predstavljene i detaljnije objašnjene sve funkcije *player*-a. Kako bi bilo moguće predstaviti tok operacija, kao reprezentativan primjer će biti iskorišten pritisak dugmeta za reprodukciju naredne pjesme u listi. Dakle, redoslijed operacija za pomenutu akciju je sledeći:

1. Korisnik pritisne dugme za reprodukciju sledeće pjesme (*next*),
2. Iz audio dijaloga se emituje signal koji je povezan sa određenim slotom definisanim u *C++*-u,
3. Slot reaguje na očekivani signal tako što provjerava informaciju koju mu je signal proslijedio (signali mogu i ne moraju da proslijede neku vrijednost slotovima) – ukoliko mu je proslijeđen redni broj pjesme, slot šalje srednjem sloju informaciju o tome,
4. Srednji sloj na *Linux*-u prosleđuje informaciju srednjem sloju na *QNX*-u,

5. Srednji sloj na *QNX*-u zaustavlja trenutnu reprodukciju i započinje reprodukciju naredne pjesme u nizu

Kako bi bilo jasno kako se ostvaruje veza između signala i slotova, u nastavku će biti objašnjeno na koji način se definišu i povezuju signali i slotovi koji se koriste za ostvarivanje redoslijeda operacija navedenog u prethodnom pasusu. Signali se u *QML*-u definišu na sledeći način:

```
Item
{
    id: audioStatusManager
    objectName: "onAudioStatusChanged"

    signal currentSongId(int songId)
},
```

gdje ***id*** predstavlja identifikaciju objekta na osnovu koje mu drugi objekti mogu pristupiti, ***objectName*** predstavlja jedinstveno ime objekta koje se koristi prilikom povezivanja signala i slotova u *player*-u, a ***currentSongId*** jeste signal koji se emituje. Definisani signal se emituje na sledeći način:

```
audioStatusManager.currentSongId(currentMediaId),
```

gdje ***currentMediaId*** predstavlja redni broj pjesme koja treba da se pusti (parametar koji se prosleđuje slotu).

Povezivanje signala sa određenim slotom se vrši u *player*-u i to na sledeći način:

- Najprije je potrebno pristupiti objektu u kom je definisan signal (pomoću imena objekta):

```
QObject *audioStatusManager =
         window→findChild<QObject*>("onAudioStatusChanged");
```

- Zatim povezati signal ***currentSongId*** `audioStatusManager`-a sa ***audioStatusSlot*** slotom, definisanim u *player*-u:

```
QObject::connect(audioStatusManager, SIGNAL(currentSongId(int)),
                  this, SLOT(audioStatusSlot(int)));
```

4.3 Realizacija *player*-a

S obzirom da je detaljno opisan redoslijed operacija i protok informacija, koje su od značaja za realizaciju audio dijaloga, kao i ostvarivanje komunikacije između dijaloga i samog *player*-a, da bi priča bila potpuna, potrebno je dati uvid i u funkcije *player*-a:

audioActiveSlot – ova funkcija se aktivira ukoliko se desi otvaranje ili zatvaranje audio dijaloga, a prima jednu od dvije vrijednosti – 0 kada se audio dijalog aktivira, a 1 po zatvaranju dijaloga. U prvom slučaju se započinje reprodukcija audio sadržaja, a ukoliko se ovoj funkciji proslijedi 1, reprodukcija se zaustavlja.

```
audioActiveSlot(int audioTabActivated);
```

playPauseButtonSlot – aktivira se kada se desi pritisak dugmeta za zaustavljanje, odnosno pokretanje reprodukcije audio sadržaja – *play*/*pause* dugme. Vrijednost koju ova funkcija prima je tipa *QString* – ukoliko primi string "*play*" nastaviće se prethodno zaustavljena reprodukcija, a ako primi string "*stop*" reprodukcija će se zaustaviti.

```
playPauseButtonSlot(QString playPause);
```

audioStatusSlot – ovaj slot se aktivira klikom na dugme za sledeću ili prethodnu pjesmu, ali i po završetku svake pjesme (kad se reprodukcija pjesme završi, audio dijalog emituje signal o tome, kako bi obavijestio *player* da treba da se pusti naredna pjesma). Parametar ove funckije je broj koji predstavlja redni broj pjesme čija reprodukcija treba da započne.

```
audioStatusSlot(int songId);
```

5. Ispitivanje

Performanse rješenja opisanog u ovom radu su mjerene u intervalima od 10-15 minuta, u ukupnom trajanju od dva sata, pri čemu je aplikacija sve vrijeme bila aktivna. Vrijednosti su mjerene i tokom "mirovanja" aplikacije (aplikacija je pokrenuta, ali nema korisničkih aktivnosti) i tokom njenog intenzivnog korištenja, što podrazumijeva prolazak kroz cijelu listu pjesama pokretima ruke, zaustavljanje i ponovno puštanje reprodukcije, prelazak iz jedne scene u drugu, zatvaranje i ponovno otvaranje dijaloga, mijenjanje pjesama na dugme *next*/*previous*, itd. Na ovaj način su upoređene vrijednosti u stanju mirovanja aplikacije i tokom njenog korištenja. Mjereno je zauzeće radne memorije, iskorištenost centralnog procesora i broj slika u sekundi, a ispitivane vrijednosti su analizirane u nastavku poglavlja.

5.1 Brzina odziva

U tabeli 4 se mogu vidjeti brzine odziva realizovanog *player*-a u zavisnosti od upotrebljenih uređaja.

HMI uređaj	Test	Vrijeme odziva [ms]
Ekran osjetljiv na dodir	Mijenjanje pjesme pritiskom dugmeta za reprodukciju prethodne ili sledeće pjesme u listi	160
	Zaustavljanje/pokretanje reprodukcije pritiskom dugmeta *stop*/*play*	160
Senzor blizine	Mijenjanje pjesme pritiskom dugmeta za reprodukciju prethodne ili sledeće pjesme u listi	160

Tabela 4 Brzine odziva realizovanog *player*-a

Kako bi se sumirali rezultati navedeni u tabeli iznad, koristiće se zadovoljavajuća vremena odziva u potrošačkoj elektronici [32]:

- Vrijeme odziva ispod 2 sekunde – maksimalno dozvoljeno vrijeme čekanja koje neće izazvati nezadovoljstvo korištenjem sistema,
- Vrijeme odziva ispod 500 milisekundi – maksimalno dozvoljeno vrijeme čekanja promjene stanja izazvane svjesnom akcijom,
- Vrijeme odziva ispod 200 milisekundi – maksimalno dozvoljeno vrijeme čekanja u realizaciji korisničkih sprega u realnom vremenu.

Na osnovu prethodno navedenog u tabeli 4 može se zaključiti da se brzina odziva korištenjem različitih *HMI* uređaja za kontrolu reprodukcije ne mijenja, a takođe ispunjava i sva tri kriterijuma navedena iznad.

5.2 Broj slika u sekundi

Broj slika u sekundi (eng. *Frames Per Second – FPS*) predstavlja relevantnu vrijednost koja utiče na fluentnost cijele aplikacije, i značajno utiče na korisničko iskustvo. Što je ova vrijednost manja, aplikacija ima tendenciju da radi nestabilno, uz povremena usporenja. Takođe, performanse aplikacije nisu zadovoljavajuće i grafički interfejs ne odgovara toliko hitro na korisničku akciju.

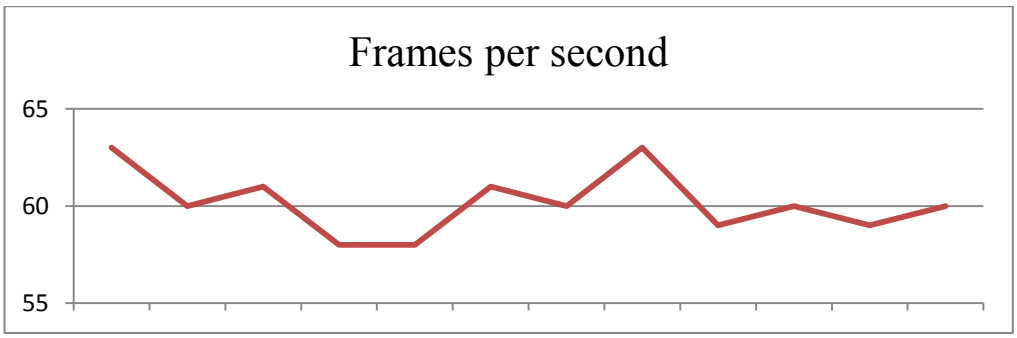
U tabeli 5 je predstavljena zavisnost izgleda jedne animacije u odnosu na broj slika u sekundi.

Broj slika u sekundi	Ponašanje animacije
10-12	Formira se jasan pokret, ali se mogu razaznati pojedinačne slike.
24	Zahvaljujući tehnikama kao što je zamućenje pokreta, ljudsko oko je u mogućnosti da vidi kretanje tečnosti.
30	Sasvim dovoljno, ali nije živopisno.
60	Ovakve animacije većina vidi kao visokokvalitetno glatko kretanje.

Tabela 5 Zavisnost izgleda animacije u odnosu na broj slika u sekundi

Na osnovu tabele se može zaključiti da 10-12 slika u sekundi nikako nije dovoljno za dobijanje kvalitetne, živopisne animacije jer se jasno mogu razaznati pojedinačne slike. 24 slike u sekundi je standardan *FPS* za filmove, dok televizija često prikazuje i do 30 slika u sekundi (zahvaljujući tehnici zamućivanja pokreta postiže se glatkoća pokreta, pa su ove vrijednosti *FPS*-a dovoljne). U svijetu igrica, standard za broj slika u sekundi je 30 ili 60 (uglavnom 30 zbog performansi) [33] [34]. Nešto je slična situacija i sa aplikacijama. Idealna vrijednost za aplikacije je oko 60 FPS. Minimalna vrijednost, mjereno po korisničkom iskustvu, je 24 FPS. Sve ispod ove vrijednosti će biti funkcionalno, ali će korisnički doživljaj znatno da se pogorša.

Vrijednost *FPS*-a za aplikaciju predstavljenu u ovom radu varira oko 60 (slika 26), što je, prema gore navedenom, zadovoljavajuće.



Slika 26 Grafik broja slika u sekundi (*FPS*) u trajanju od dva sata korištenja aplikacije

5.3 Iskorištenost procesorske moći i zauzeće radne memorije

Iskorištenost procesorske moći je predstavljena u tabeli 3.

Operativni sistem	Test	Iskorištenost procesora [%]	Iskorištenost radne memorije [%] (*Linux*)/ Količina slobodne memorije [M] (*QNX*)
QNX	*QNX* podignut bez *Linux*-a	6.75	760
	QNX podignut sa *Linux*-om, i sistem pokrenut	28	323
	Reprodukovanje audio sadržaja	29	320
Linux	*QNX* podignut sa *Linux*-om, i sistem pokrenut	12	13
	Reprodukovanje audio sadržaja	24.5	14.3

Tabela 6 Iskorištenost procesorske moći i zauzeće radne memorije

Iz tabele zaključujemo da za *QNX*, pri reprodukciji audio sadržaja, iskorištenje procesora poraste za 1 procenat, što je skoro beznačajno, dok se za *Linux* ta vrijednost duplo poveća.

Zauzetost memorije za *Linux* se kreće između 13% i 14.3%, što znači da nema većih oscilacija. U toku sat vremena intenzivnog korištenja aplikacije, zauzetost memorije je bila relativno nepromjenljiva, što govori o odsustvu problema „curenja memorije". Isto važi i za *QNX*, s tim da su vrijednosti izražene u megabajtima, pa iz tabele vidimo da ni za ovaj operativni sistem nema većih oscilacija u smislu zauzetosti memorije (prirodno je da se količina slobodne

memorije na *QNX*-u značajno smanji podizanjem *Linux*-a, jer *QNX* podiže ovaj operativni sistem).

5.4 Funkcionalna verifikacija

Funkcionalna verifikacija podrazumijeva prolazak kroz osnovne funkcionalnosti audio dijaloga u cilju testiranja rada dijaloga.

U tabeli 7 su navedeni svi testni slučajevi sa očekivanim rezultatima.

Testni slučaj	Opis koraka	Očekivani rezultat	Uspješnost
Otvaranje audio dijaloga	1. Pritisnuti dugme "Audio" u početnom meniju 2. Ponovo pritisnuti isto dugme kako bi se dijalog zatvorio	Otvara se audio dijalog	Uspiješno
Puštanje/zaustavljanje pjesme	1. Pritisnuti dugme "Audio" u početnom meniju 2. Pritisnuti dugme *play*/*pause* za zaustavljanje/pokretanje reprodukcije	Otvara se audio dijalog i započinje reprodukcija audio sadržaja. Pritiskom na dugme *stop* reprodukcija se zaustavlja, a pritiskom na *play* reprodukcija se ponovo pokreće.	Uspiješno
Puštanje prethodne/sledeće pjesme	1. Pritisnuti dugme *stop* 2. Pritisnuti dugme *play*	Reprodukcija se zaustavlja i ponovo pokreće pritiskom odgovarajućeg dugmeta	Uspiješno
Promjena postera na ekranu *Cluster* aplikacije	Pritisnuti dugme za reprodukciju sledeće ili prethodne pjesme (*previous*/*next*)	Započinje reprodukcija sledeće/prethodne pjesme u listi, a istovremeno se mijenja poster na ekranu *Cluster* aplikacije	Uspiješno

Testni slučaj	Opis koraka	Očekivani rezultat	Uspješnost
Promjena pjesme pokretom ruke	Mahnuti rukom ulijevo ili udesno ispred senzora za blizinu, dok je audio dijalog aktivan	Započinje reprodukcija sledeće ili prethodne pjesme u listi, u zavisnosti od smjera mahanja	Uspiješno

Tabela 7 Testni slučajevi

Sve funkcionalnosti su provjeravane više puta, i sve su bile uspiješne. Ovim testom je provjerena ispravnost rada *player*-a i audio dijaloga realizovanih u ovom radu.

6. Zaključak

Razvojem multimedijalnih sistema i njihovom potražnjom na tržištu automobilske industrije, raste i potreba za razvojem i usavršavanjem tih sistema, ali i načina ostvarivanja komunikacije između vozača i samog sistema u automobilu. U ovom radu je predstavljeno jedno rješenje realizacije *player*-a za multimediju u automobilu u aplikativnim okruženjima zasnovanim na *Qt*-u, koje takođe omogućava kontrolu funkcionalnosti, kao što je reprodukcija sledeće ili prethodne pjesme, pokretom ruke.

Ispitane su performanse rješenja, a rezultati su pokazali da je ovo rješenje primjenljivo, te da nema prekoračenja u smislu dodatnog vremena potrebnog za izvršavanje funkcija *player*-a.

Dalji pravci razvoja podrazumijevaju implementaciju onih dijelova audio dijaloga, u smislu razvoja korisničkog izgleda svih scena i proširenja funkcionalnosti dijaloga, koje su predviđene za razvoj (a ispunjavaju korisničke zahtjeve navedene u tabeli 1), ali još uvijek nisu implementirane.

Osim audio dijaloga, dalji pravci razvoja ovog sistema podrazumijevaju neke izmjene u samoj arhitekturi sistema, što će takođe izazvati određene promjene u realizaciji *player*-a (sva logika će biti smještena u *player*-u (manipulacija reprodukcijom)). Zapravo, u finalnoj verziji softvera će sva logika i suština srednjeg sloja biti na *QNX* operativnom sistemu (srednji slojevi na oba operativna sistema su dizajnirani tako da budu portabilni, pa će biti lako premjestiti logiku, koja je trenutno smještena na *Linux*-u, u *QNX*), a odatle će se samo slati statusne poruke *Linux*-u, na osnovu kojih će se izvršavati određene akcije na toj strani. Ovim se postiže da jezgro sistema bude na sigurnijem operativnom sistemu, i da dizajn bude usklađen sa *ISO26262* standardom, koji navodi da softver *Cluster* aplikacije (*ASIL B*) ne smije zavisiti od softvera koji ne podliježe *ASIL* klasifikaciji.

7. Literatura

[1] https://www.lifewire.com/car-multimedia-basics-534720 jul 2017.

[2] https://en.wikipedia.org/wiki/Multimedia oktobar 2017.

[3] https://www.ics.com/company/about

[4] https://www.ics.com/blog/media-manager-automotive-infotainment-part-1 mart 2016.

[5] http://www.cinemo.com/whycinemo/portable-solutions.html

[6] http://markets.businessinsider.com/news/stocks/Cinemos-Unified-Media-Solution-Presented-on-a-Next-Generation-In-Vehicle-Infotainment-IVI-Prototype-at-IAA-2017-455532 septembar 2017

[7] http://www.businesswire.com/news/home/20120725005692/en/Cinemo-Redefines-In-Vehicle-Infotainment-IVI-Experience-Ground jul 2012

[8] Jun Ma, Maofei Xu, Yuchun Du, „A Usability Study on In-Vehicle Gesture Control“, 14.9.2016, https://papers.sae.org

[9] Pablo Sauraz-Perez, Andrea Gil, Jasprint Singh Gill, Pierluigi Pisu, Joachim Taiber, „VoGe: A Voice and Gesture System for Interacting with Autonomous Cars“, 28.3.2017, https://papers.sae.org

[10] https://www.just-auto.com/analysis/next-wave-of-gesture-recognition-technologies_id173440.aspx novembar 2016.

[11] https://en.wikipedia.org/wiki/GENIVI_Alliance oktobar 2017.

[12] https://at.projects.genivi.org/wiki/pages/viewpage.action?pageId=2785416 avgust 2017.

[13] https://at.projects.genivi.org/wiki/display/GRK/2_Reference+Architecture+and+Compliance+Specification avgust 2017.

[14] https://at.projects.genivi.org/wiki/display/PROJ/Media+Manager oktobar 2016.

[15] https://en.wikipedia.org/wiki/Windows_Embedded_Automotive jun 2017.

[16] https://en.wikipedia.org/wiki/Ford_Sync oktobar 2017.

[17] https://www.usatoday.com/story/money/cars/2014/12/11/ford-sync-3/20234131/ decembar 2014.

[18] https://www.computerworld.com/article/2859373/ford-dumps-microsoft-for-qnx-unleashes-new-functions-in-sync-v3.html decembar 2014.

[19] QNX, „QNX CAR Platform for Infotainment “, 2014.

[20] QNX CAR™ Platform for Infotainment 2.1, „Architecture Guide“, 16.septembar 2016.

[21] https://en.wikipedia.org/wiki/Android_Auto#cite_note-10 oktobar 2017.

[22] https://play.google.com/store/apps/collection/promotion_3001303_android_auto_all 2017.

[23] https://www.android.com/auto/ 2017.

[24] Bin Yang, „2014 Google I/O overview“, https://www.slideshare.net/BinYang7/google-io-2014-overview, 4.avgust 2014.

[25] http://www.techradar.com/news/car-tech/apple-carplay-everything-you-need-to-know-about-ios-in-the-car-1230381 jun 2017.

[26] https://www.apple.com/ios/carplay/ 2017.

[27] http://searchservervirtualization.techtarget.com/definition/hypervisor decembar 2016.

[28] https://en.wikipedia.org/wiki/GENIVI_Alliance oktobar 2017.

[29] https://www.qnx.com/content/qnx/en/products/hypervisor/index.html

[30] Jörn Schneider, Tillmann Nett, “Safety Issues of Integrating IVI and ADAS functionality via running Linux and AUTOSAR in parallel on a Dual-Core-System”

[31] http://europe.autonews.com/article/20141112/ANE/141119993/supplier-competition-intensifies-for-infotainment-software novembar 2014.

[32] Milan Bjelica, doktorska disertacija: “Metode realizacije kontekstualnih platformi i kontekstualnih korisničkih sprega za primjene u uređajima potrošačke elektronike”, 2013.

[33] http://blog.logicalincrements.com/2015/04/does-fps-matter-decide-for-yourself/ april 2015.

[34] http://www.ign.com/articles/2014/11/05/understanding-frame-rate-and-its-importance novembar 2014.