УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Сретен Танацковић

Једно решење даљинског управљања симулацијом периферних уређаја на Андроид платформи

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2013

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**:		Завршни (Bachelor) рад	
Аутор, **АУ**:		Сретен Танацковић	
Ментор, **МН**:		др Јелена Ковачевић	
Наслов рада, **НР**:		Једно решење даљинског управљања симулацијом периферних уређаја на Андроид платформи	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2013	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/37/0/0/17/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кључне речи, **ПО**:		Андроид, Линукс, УПнП, УДП, ДТВ	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом раду је реализовано једно решење даљинског управљања уређаја заснованих на Линукс језгру. Циљ реализације је програмска подршка која омогућава управљање оваквим уређајима симулацијом периферних уређаја. У раду је такође приказана реализација програмске подршке којом се врши управљање, а намењена је за преносне уређаје засноване на Андроид платформи.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	проф. др Никола Теслић	
	Члан:	мр Милан Савић	Потпис ментора
	Члан, ментор:	др Јелена Ковачевић	

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

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monographic publication
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Bachelor Thesis
Author, **AU**:	Sreten Tanacković
Mentor, **MN**:	Jelena Kovačević, PhD
Title, **TI**:	One Solution of Peripheral Devices Simulation on Android Platform
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**:	2013
Publisher, **PB**:	Author's reprint
Publication place, **PP**:	Novi Sad, Dositeja Obradovica sq. 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/37/0/0/17/0/0
Scientific field, **SF**:	Electrical Engineering
Scientific discipline, **SD**:	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, **S/KW**:	Android, Linux, UPnP, UDP, DTV
UC	
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, **N**:	
Abstract, **AB**:	This paper presents one solution of remote controller designed for devices based on Linux kernel. The goal of this solution is creating a software that allows management of these devices by simulating pheripheral devices. This paper also contains the realization of controller, which is intended for handheld devices based on Android platform.
Accepted by the Scientific Board on, **ASB**:	
Defended on, **DE**:	

Defended Board, **DB**:	President:	Nikola Teslić, PhD	
	Member:	Milan Savić, MSc	Menthor's sign
	Member, Mentor:	Jelena Kovačević, PhD	

Zahvalnost

Zahvaljujem se institutu RT-RK na pruženoj mogućnosti za realizaciju ovog rada.

Takođe se zahvaljujem mentoru dr Jeleni Kovačević, stručnim saradnicima Nikoli Kuzmanoviću i Milanu Saviću, kao i kolegama iz AMUSE tima na stručnoj pomoći i savetima prilikom izrade ovog rada.

Na kraju, zahvaljujem se svojoj porodici na pruženoj podršci tokom mog školovanja.

SADRŽAJ

1. Uvod 1
2. Teorijske osnove 3
2.1 Android platforma 3
2.2 Android aplikacije 4
2.3 UDP komunikacioni protokol 6
2.4 UPnP protokol 7
2.5 uinput modul Linux jezgra 8
3. Koncept rešenja 9
3.1 Koncept aplikacija uslužioca 9
3.2 Koncept korisničke aplikacije 11
4. Programsko rešenje 13
4.1 Aplikacija uslužioca 13
4.1.1 Glavni modul uslužioca 15
4.1.2 Modul za simulaciju perifernih uređaja 16
4.1.3 Modul za komunikaciju pomoću UPnP protokola 18
4.1.4 Modul za komunikaciju pomoću izvedenog protokola 18
4.2 Klijentska aplikacija 20
4.2.1 Glavni modul klijentske aplikacije 20
4.2.2 Modul za upravljanje računarskim mišem 21
4.2.3 Modul za upravljanje tastaturom 22
4.2.4 Modul za upravljanje daljinskim upravljačem 22
4.2.5 Komunikacioni modul 23
5. Ispitivanje i verifikacija 25

5.1 Programsko ispitivanje funkcionalnosti 25
5.2 Ispitivanje od strane krajnjih korisnika 26
6. Zaključak 28
7. Literatura 29

SPISAK SLIKA

Slika 2.1. Prikaz slojevite arhitekture Android platforme 3
Slika 2.2. Format UDP datagrama 6
Slika 2.3. Umreženi UPnP uređaji 7
Slika 3.1 Prikaz algoritma rada servisa 10
Slika 3.2 Prikaz algoritma funkcionisanja izvedenog protokola.....11
Slika 3.3 UML sekvencijalni dijagrama reakcije slojeva na korisničku interakciju.....12
Slika 4.1 UML dijagram apstraktne klase RemoteServerModule 14
Slika 4.2 UML dijagram klase RemoteServer 15
Slika 4.3 UML klasni dijagram modula za simulaciju perifernih uređaja 17
Slika 4.4 UML klasni dijagram UPnP komunikacionog modula.....18
Slika 4.5 UML dijagram klasa komunikacionog modula za izvedeni protokol.....19
Slika 4.6 Grafička reprezentacija glavnog modula korisničke aplikacije 21
Slika 4.7 Grafička reprezentacija modula za rukovanje računarskim mišem 21
Slika 4.8. Prikaz izgleda grafičke sprege kojom se upravlja virtuelnom tastaturom 22
Slika 4.9 Prikaz izgleda grafičke sprege realizovanog daljinskog upravljača 23
Slika 4.10 UML klasni dijagram komunikacionih klasa.....24
Slika 4.11 Grafički prikaz dijaloga koji se prikazuje za vreme uspostavljanja veze sa uslužiocem 24

SKRAĆENICE

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

LAN - *Local Area Network,* Lokalna mreža

WLAN - *Wireless Local Area Network,* Bežična lokalna mreža

STB - *Set-Top Box,* Uređaj za prijem TV signala

UDP - *User Datagram Protocol,* Protokol za korisničke datagrame

TCP - *Transmission Control Protocol,* Protokol kontrole toka

IP - *Internet Protocol,* Internet protokol

HTTP - *Hypertext Transfer Protocol,* Protokol za prenos hiper teksta

XML - *Extensive Markup Language,* Proširivi meta jezik za označavanje tekstualnih dokumenata

SDK - *Software Developement Kit,* Skup alata za razvoj programske podrške

JNI - *Java Native Interface,* Java nativna sprega za povezivanje sa kodom pisanim u C,C++ ili asemblerskom programskom jeziku

RTP - *Real-Time Transport Protocol,* Protokol za prenos podataka u realnom vremenu

VoIP - *Voice over Internet Protocol,* Protokol za prenos govora preko Internet protokola

DNS - *Domain Name System,* Sistem za dodelu naziva računarima u mreži

1. Uvod

U ovom radu je realizovan jedno rešenje daljinskog upravljanja uređajima zasnovanih na Linux jezgru. Cilj realizacije je programska podrška koja omogućava upravljanje ovakvim uređajima simulacijom perifernih uređaja. U radu je takođe prikazana realizacija programske podrške kojom se vrši upravljanje, a namenjena je za prenosne uređaje zasnovane na Android platformi.

Android platforma se zasniva na Linux jezgru[1]. Prvenstveno je namenjena za prenosne uređaje (mobilne telefone). Razvojem novih tehnologija na polju računarstva uz povećanje procesorske moći i smanjenjem cene elektronskih komponenti hardver potrošačkih uređaja sve više podseća na personalne računare. To dovodi da je Android van svoje osnovne namene našao primenu i u raznim potrošačkim uređajima kao što su digitalni TV i STB uređaji, digitalni foto aparati, prenosni i kućni multimedijalni plejeri, navigacioni uređaji i drugi. Akcenat ovog rada je na TV uređajima [2] sa mogućnošću prijema digitalnog signala koji su zasnovani na Android platformi. Takvi uređaji pored svoje osnovne namene, gledanja televizijskog programa, imaju mogućnost umrežavanja sa drugim uređajima preko lokalne mreže, pristup raznim internet servisima, korišćenje raznih Android aplikacija, reprodukciju multimedijalnog sadržaja i mnoge druge mogućnosti. Oni standardno podržavaju interakciju sa korisnikom jedino preko daljinskog upravljača. Kompleksnost i skup mogućih funkcija takvog daljinskog upravljača varira i zavisi od proizvođača. Glavni nedostatak ovog koncepta je što skup funkcija klasičnog daljinskog upravljača ne pruža mogućnost iskorištenja svih funkcionalnosti koje nudi Android platforma. Taj nedostatak se može primetiti prilikom pristupa internetu ili igranja Android igara na ovakvom uređaju. Kako bi proizvođači nadomestili ovaj nedostatak za svoje DTV proizvode kreiraju sofisticirane daljinske upravljače koji na sebi sadrže tastaturu ili region osetljiv na dodir. Popularnost korišćenja „pametnih” telefona i tableta u svetu dovodi do toga da će u bližoj

budućnosti svi imati takav uređaj. Imajući to u vidu, cilj ovog rada je napraviti aplikaciju za prenosni uređaj (mobilni telefon ili tablet) koja omogućava upravljanje platformom zasnovanom na Android-u. Time je rešenje univerzalno za širok skup potrošačkih uređaja koji funkcionišu na sličnim principima. Jedan od načina na koji se može realizovati takvo upravljanje jeste simulacijom perifernih uređaja (računarskog miša i tastature)[9][11]. Simulacijom računarskog miša moguće je donekle nadomestiti nedostatak ekrana na dodir koji je prisutan na prenosnim, a nema ga na drugim Android baziranim uređajima. Primena računarskog miša u ovom okruženju poprilično olakšava korišćenje raznih aplikacija, pa čak i omogućava korišćenje nekih aplikacija kojima je nemoguće upravljati standardnim daljinskim upravljačem. Upravljanje koje se postiže simulacijom tastature najkorisnije je prilikom pristupa internetu. Takođe, dobra strana ovog rešenja je što se pomoću simulacije tastature može izvršiti identično upravljanje kao i sa standardnog daljinskog upravljača.

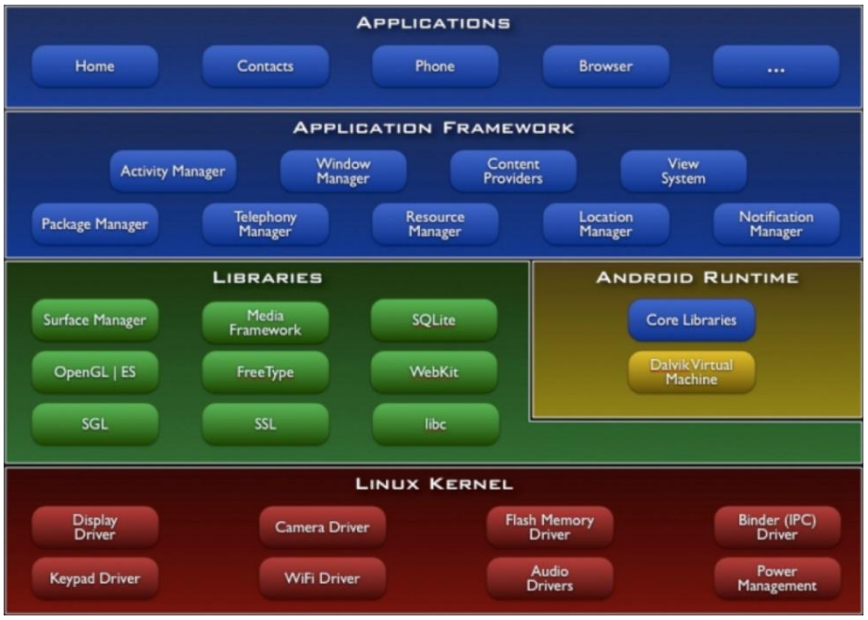
U drugom poglavlju date su teorijske osnove na kojima je rad zasnovan. Treće poglavlje opisuje koncept na osnovu koga je nastao realizovani sistem od dve Android aplikacije za upravljanje platformom zasnovanom na Android-u. Opis realizovanoh programskih modula ove dve aplikacije je dat u četvrtom poglavlju. U petom poglavlju opisani su načini ispitivanja i verifikovanja navedenog programskog rešenja. Šesto poglavlje predstavlja kratak pregled onoga što je realizovano u ovom radu. U sedmom poglavlju dat je spisak korišćene literature za izradu rada.

2. Teorijske osnove

Ovo poglavlje opisuje teorijske osnove na kojima je rad zasnovan. U njemu su pomenute osnovne informacije o Android platformi i Android aplikacijama. Ovo poglavlje obuhvata i opis korišćenih komunikacionih protokola za realizaciju rešenja. Takođe, poglavlje se sastoji i iz opisa „uinput“ modula Linux jezgra koji je neophodan za simulaciju perifernih uređaja.

2.1 Android platforma

Android platforma se zasniva na Linux jezgru[1]. Prednost Android platforme je njena mogućnost lakog prilagođenja prirodi krajnjih uređaja zbog slojevite arhitekture koja je karakteriše.



Slika 2.1. Prikaz slojevite arhitekture Android platforme

Prvobitno je Android platforma razvijena za ARM procesorsku arhitekturu. Danas je našla primenu na velikom broju različitih uređaja koje karakterišu različite arhitekture, bilo ARM, MIPS ili x86.

Softverska arhitektura sistema[3] predstavlja standardnu arhitekturu Linux-a gde su segmenti sistema razdvojeni po nivoima na kojim rade. Najniži sloj ove arhitekture se sastoji od modifikovanog monolitnog Linux jezgra zaduženog za podršku hardvera i funkcija niskog nivoa. Dalje se sastoji iz skupa biblioteka zaduženih za dodatne podrške kao što su iscrtavanje grafika, podrška za dekodovanje video snimaka, podrška za SSL enkripciju, itd. U sklopu biblioteka se nalazi i odvojeni sloj „Android Runtime“ koji sadrži osnovne, bazne, biblioteke i Dalvik virtualnu mašinu. Dalvik virtualna mašina je zadužena za pokretanje aplikacija višeg nivoa napisanih u Java programskom jeziku. Na višem nivou od biblioteka su sistemske aplikacije neophodne za upotrebu sistema od strane korisnika i tu se nalaze menadžeri prozora, resursa, instalacionih paketa, kao i aplikacije zadužene za obavljanje osnovnih funkcija vezenih za uređaj na kom je instaliran Android. Na najvišem nivou se nalaze krajnje korisničke aplikacije, odnosno aplikacije koje direktno koristi korisnik.

Za crtanje 3D grafike Android koristi biblioteku zasnovanu na OpenGL ES specifikaciji, što ovom sistemu daje mnoge napredne grafičke sposobnosti. Android poseduje i ugrađenu podršku za multitasking. Android se oslanja na Linux jezgro za osnovne usluge sistema kao što su bezbednost, upravljanje memorijom, upravljanje procesima, mrežno skladište i drajveri. Jezgro takođe spaja apstrakcije između fizičke arhitekture i ostatka programskog paketa.

2.2 Android aplikacije

Android aplikacije se pišu u Java programskom jeziku. Android SDK (engl. *Software Developement Kit*)[3] alati prevode kod, zajedno sa svim podacima i potrebnim datotekama u Android paket. Android paket predstavlja arhivsku datoteku sa ekstenzijom *apk*. Ova datoteka se šalje na Android uređaj, gde se postavlja (engl. *Install*) i omogućava pristup od strane krajnjih korisnika.

Android aplikacija radi u svom procesu, sa sopstvenom instancom Dalvik virtuelne mašine[3]. Dalvik virtuelna mašina izvršava Dalvik izvršne datoteke u *dex* formatu, koje su optimizovane za minimalnu potrošnju memorije. Virtuelna mašina pokreće klase prevedene Java programskim prevodiocem u *dex* formatu pomoću „dx“ alata. Radi osnovne funkcionalnosti kao što su paralelizam i pristup memoriji niskog nivoa virtuelna mašina se oslanja na Linux jezgro.

Svaki proces ima svoju virtualnu mašinu, tako da se kôd aplikacije pokreće izolovan od drugih aplikacija. Po pravilu svaka aplikacija se pokreće u nezavisnom Linux procesu. Proces se

pokreće kada jedna od komponenti aplikacije treba da se izvrši. Android gasi proces kada više nije potreban ili kada je potrebno oporaviti memoriju za druge aplikacije. Na ovaj način Android primenjuje princip najmanje privilegije. Odnosno svaka aplikacija ima pristup komponentama koje su joj potrebne za izvršenje svog zadatka. Ovo stvara vrlo bezbedno okruženje u kome aplikacija ne može da pristupi delovima sistema za koje nema odobrenje.

Aplikacija može da zatraži dozvolu za pristup podacima[4] kao što su kontakti korisnika, SMS poruke, kamera, i drugo. Sve dozvole programer nabraja u *AndroidManifest.xml* datoteci, a moraju biti odobrene od strane korisnika u trenutku instalacije aplikacije inače aplikacija neće biti instalirana. Pre nego što Android platforma može da pokrene neku komponentu aplikacije, mora da sazna da ta komponenta postoji čitajući *AndroidManifest.xml* datoteku aplikacije. Sve komponente aplikacije se moraju definisati u ovoj datoteci.

Aplikacione komponente [1][3] su osnovni gradivni blokovi Android aplikacije. Postoje četiri različite vrste aplikacionih komponenti, gde svaka ima jasnu svrhu i poseban životni ciklus koji definiše kako se komponenta kreira i uništava.

- **Aktivnosti** (engl. *Activity*) predstavljaju jedan prikaz na ekranu sa grafičkom korisničkom spregom. Aplikacija može da se sastoji iz više aktivnosti. Kombinacijom više aktivnosti formira se jedinstven utisak o aplikaciji kod korisnika. Ipak svaka od aktivnosti je nezavisna i može se samostalno pokrenuti u zavisnosti od situacije i potrebe.
- **Servis** (engl. *Service*) je komponenta koja radi u pozadini i obavlja dugotrajne radnje bez direktne interakcije sa korisnikom. Servis ne sadrži grafičku korisničku spregu. Druge komponente mogu pokrenuti servis ili se povezati na njega kada im je potrebno da izvrše uslugu koju on obezbedjuje. Jedan od načina korišćenja servisa je da servis može da pušta muziku u pozadini dok se korisnik nalazi u nekoj drugoj aplikacija.
- **Dobavljač sadržaja** (engl. *Content Provider*) omogućava deljenje podataka između aplikacija. Moguće je skladištiti podatke u sistemu datoteka, SQLite bazi podataka, na internetu ili bilo kojoj drugoj lokaciji za skladištenje kojoj aplikacija može da pristupi.
- **Primalac emitovanih poruka** (engl. *Broadcast Receiver*) je komponenta Android sistema koja je vezana za slanje i prijem sistemskih poruka. Ove sistemske poruke najčešće mogu da nastaju u trenutku kada je ekran isključen, baterija prazna, i drugo.

Android aplikacije napisane u Java programskom jeziku imaju pristup samo aplikativnom sloju Android platforme. Kako bi se nadomestio ovaj nedostatak Android podržava povezivanje koda aplikacije sa kodom koji nije pisan u Java programskom jeziku pomoću JNI sprege. JNI je programski okvir koji omogućava spregu između koda napisanog u Java programskom jeziku i koda napisanog u drugim jezicima, kao što su C, C ++ i asembler. Na ovaj način aplikacija može imati pristup sistemskom prostoru.

2.3 UDP komunikacioni protokol

UDP protokol[5] (engl. *User Datagram Protocol*) je jedan od osnovnih protokola koji se primenjuje u internetu. Ovaj protokol ne zahteva uspostavljanje i održavanje komunikacione sesije, već mu je samo bitna isporuka paketa destinaciji. Nije bitno da li je destinacija stvarno primila poslati paket i kojim redosledom. Ono što je bitno UDP protokolu je da to što se pošalje što pre stigne do destinacije bez velikih gubitaka vremena i mrežnih resursa na podatke koji se šalju. Neke od osobina UDP protokola mogu da se navode i kao njegove mane, nema proveru na greške, višestruko slanje datagrama, gubitak podataka, dostavljanje podataka u izmenjenom redosledu. Međutim, većina aplikacija i servisa koje koriste ovaj protokol od njega očekuju maksimalnu brzinu isporuke podataka, bez obzira na sve greške i propuste, jer bi uvođenje dodatnih opcija provere grešaka vrlo verovatno ugrozilo njihov rad, odnosno direktno usporilo isporuku podataka destinaciji. Koristi ga veliki broj aplikacija, naročito multimedijalne aplikacije poput internet telefonije i video konferencije. Koriste ga protokoli RTP, VoIP, DNS, serveri za računarske igre, i drugi. UDP obezbeđuje kontrolnu sumu podataka koji se prenose i na taj način omogućuje unutrašnju kontrolu na greške. Koristi brojeve prolaza (engl. *Port*) radi adresiranja izvornih i odredišnih datagrama.

UDP je pogodan gde provera grešaka ili oporavak od grešaka nije neophodan radi ispravnog funkcionisanja aplikacija. Vremenski senzitivne aplikacije često koriste UDP protokol ako gubljenje paketa uzrokuje manju štetu na sistem nego usporenje u radu koje može da se desi korišćenjem protokola koji garantuju isporuku zaostalih paketa kakav je TCP.

bits	0 – 7	8 – 15	16 – 23	24 – 31
0	Source address			
32	Destination address			
64	Zeros	Protocol	UDP length	
96	Source Port		Destination Port	
128	Length		Checksum	
160+	Data			

Slika 2.2. Format UDP datagrama

2.4 UPnP protokol

UPnP (engl. *Universal Plug and Play*) [6][7] je skup mrežnih protokola koji omogućavaju automatsko pronalaženje, priključivanje i korišćenje uređaja koji su priključeni na računarsku mrežu. Primeri uređaja koji mogu da koriste ovaj protokol su lični računari, štampači, bežične pristupne tačke, mobilni uređaji i drugi. UPnP omogućava da se bez problema otkrije prisustvo drugih uređaja u mreži i uspostava mrežnih usluga za deljenje podataka, komunikaciju i zabavu.

UPnP tehnologiju promoviše UPnP Forum. To je inicijativa računarske industrije namenjena omogućavanju jednostavne i robusne povezanosti samostalnih uređaja i ličnih računara različitih proizvođača. Forum se sastoji od preko osamsto proizvođača, koji pokrivaju oblasti od potrošačke elektronike do mrežnog računarstva. UPnP je konceptualni produžetak „plug-and-play" protokola, tehnologije za dinamičko povezivanje uređaja sa računarom. Mada UPnP nije direktno srodan sa ranijom „plug-and-play" tehnologijom, UPnP uređaji su „plug-and-play" u smislu da kad su povezani na mrežu oni automatski uspostavljaju radne konfiguracije sa drugim uređajima.

UPnP tehnologija je nezavisna od operativnog sistema i programskog jezika i izgrađena je na internet baziranim tehnologijama kao što su IP, TCP, UDP, HTTP i XML.

Slika 2.3. Umreženi UPnP uređaji

UPnP ne sadrži ugrađeni mehanizam za autentikaciju korisnika, dok je postojeći sigurnosni protokol veoma kompleksan, što može da otežа implementaciju autentikacije. Za prenos kratkih poruka preko mreže može biti suviše glomazan, ali je koristan prilikom prenosa veće količine podataka. Jedna od loših strana ovog protokola je otkrivena početkom 2013. godine [8]. 40-50

miliona uređaja koji podržavaju UPnP i imaju stalnu vezu sa internetom je osetljiva na napade zbog pronađenih nedostataka u protokolu.

2.5 uinput modul Linux jezgra

Sve se u Linux-u tretira kao datoteka[9], čak i hardverski uređaji kao što su serijski portovi, hard diskovi, tastature, računarski miševi, skeneri , itd. Da bi se moglo pristupiti tim uređajima, specijalna datoteka zvana čvor uređaja (engl. *Device Node*) mora da postoji. Svi čvorovi uređaja se nalaze u */dev* direktorijumu. Preko čvorova uređaja Linux jezgro komunicira sa fizičkim uređajima.

Fizički uređaji mogu biti:

- Blokovsko orijentisani uređaji (svi diskovi i fleš memorije)
- Znakovno orijentisani uređaji (računarski miševi, tastature, virtuelni terminali, i drugi)
- Mrežni uređaji

Od ključnog interesa u ovom radu su znakovno orijentisani uređaji. Svaki programski rukovalac (engl. *Driver*) ovakvog uređaja registruje se Linux jezgru sa skupom svih funkcija koje implementiraju one operacije datotečkog sistema koje rukovalac podržava. Sistem komunicira sa ovim uređajima slanjem podataka znak-po-znak. Najčešće ne koriste skladištenje podataka u memoriji (baferovanje) prilikom ulaznih i izlaznih operacija. Programski rukovalac koji rukuje njima momentalno čita znak iz datoteke i upisuje ga na uređaj.

„uinput“ [9][10] je modul Linux jezgra koji služi za upravljanje ulaznim podsistemom iz korisničkog prostora, zbog šega se i naziva korisnički unos (engl. *User Input*). Ovaj modul omogućava kreiranje i upravljanje ulaznim znakovno orijentisanim uređajima iz neke aplikacije. Takvi uređaji se registruju u specijalnu datoteku na putanji */dev/uinput* ili */dev/input/uinput* . Ova putanja zavisi od verzije Linux jezgra. Uređaj registrovan u „uinput“ modulu predstavlja virtuelnu spregu koja ne pripada ni jednoj fizičkoj komponenti sistema. Virtuelni uređaj može imati funkciju bilo kog znakovno orijentisanog uređaja ili čak kombinaciju više takvih uređaja.

3. Koncept rešenja

Rešenje simulacije periferijskih uređaja na Android platformi se sastoji iz dve Android aplikacije. Ove aplikacije su zamišljene tako da predstavljaju klijent-uslužilac arhitekturu. Obe aplikacije su zamišljene modularno, kako bi dalje unapređenje i dodavanje novih funkcija u ovo rešenje bilo jednostavno i moguće bez izmena postojećih modula.

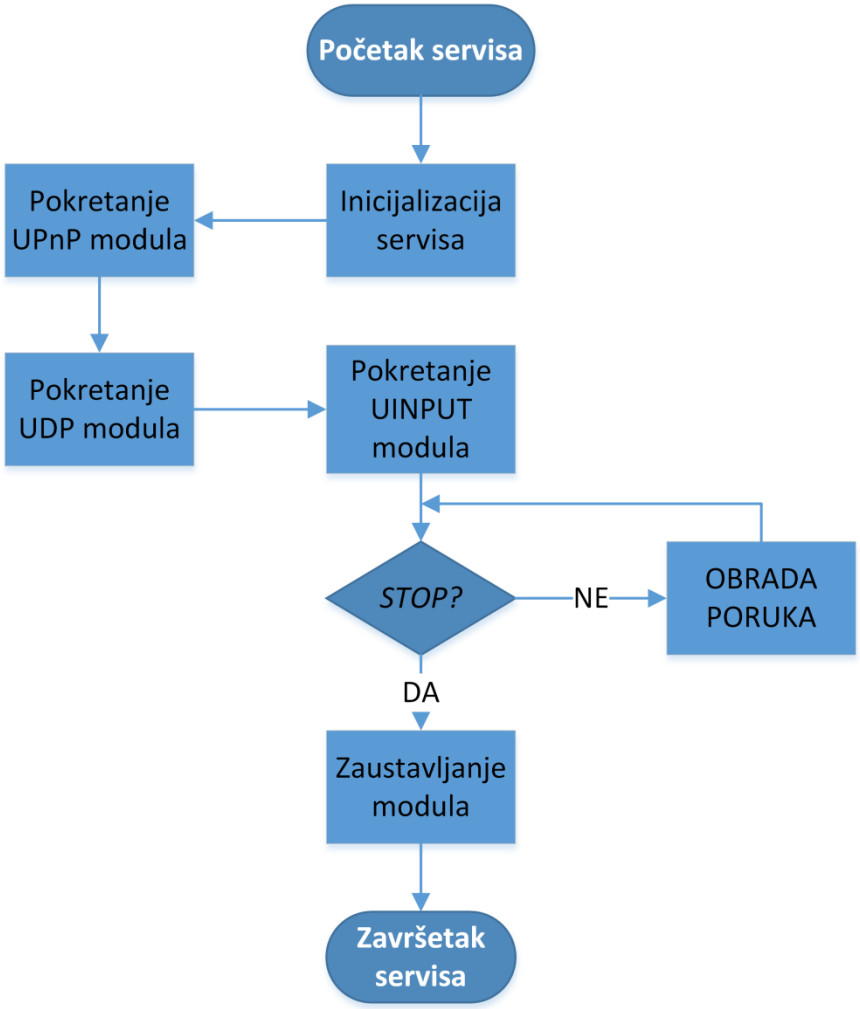
3.1 Koncept aplikacija uslužioca

Aplikacija uslužioca je zamišljena kao Android servis. To zanči da ona ne pruža grafičku spregu za korisnike. Različite funkcionalnosti su omogućene različitim modulima aplikacije.

Aplikacija uslužioca se sastoji iz:

- Glavnog modula
- Modula za simulaciju perifernih uređaja
- Komunikacionog modula zasnovanog na UPnP protokolu
- Komunikacionog modula zasnovanog na izvedenom protokolu

Glavni modul je ulazna tačka aplikacije i zadužen je da inicijalizuje ostale module po pokretanju servisa, vrši obradu poruka, kao i da pravilno deinicijalizuje ostale module pre zaustavljanja servisa. Modul za simulaciju perifernih uređaja se oslanja na „uinput“ modul Linux jezgra. Zadatak mu je da registruje virtuelni uređaj u ovom modulu koji će predstavljati spregu za upravljanje platformom. Komunikacioni moduli vrše programsku implementaciju korišćenih komunikacionih protokola.



Slika 3.1 Prikaz algoritma rada servisa

Modul za simulaciju perifernih uređaja se vrši simulaciju pomoću „uinput“ modula Linux jezgra. Virtuelni događaji koji se prosleđuju ovom modulu su identični događajima koji nastaju tokom rada stvarnih fizičkih uređaja.

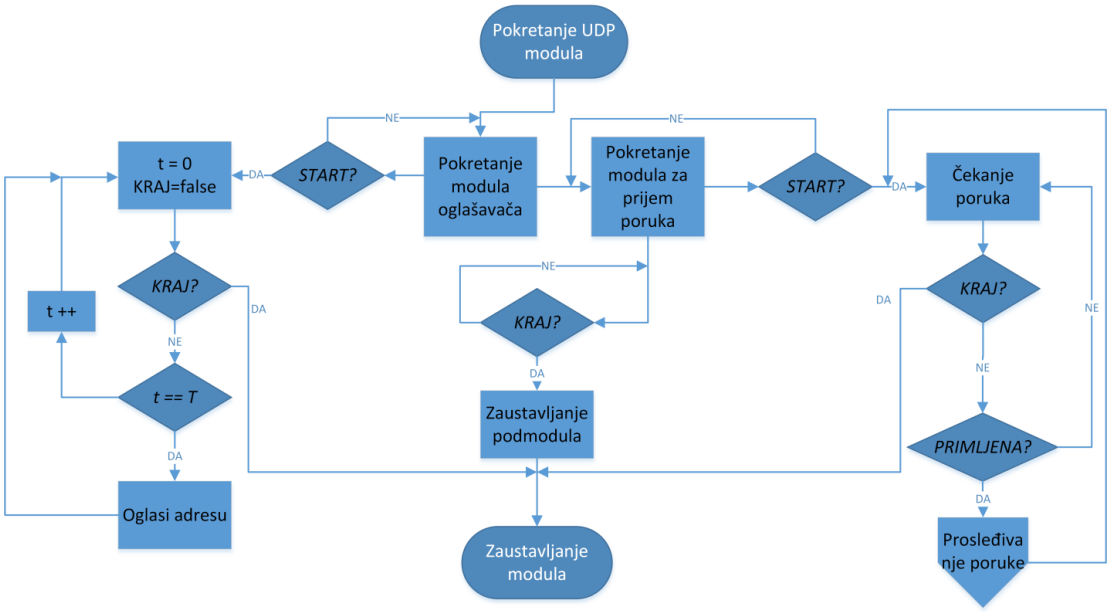
Komunikacioni modul zasnovan na UPnP protokolu se oslanja na UPnP biblioteku i implementira one funkcionalnosti koje su neophodne za realizaciju ovog rešenja, kao što su prihvat konekcije od klijenata(na strani uslužioca) i razmena poruka između aplikacija.

Komunikacioni modul zasnovan na izvedenom protokolu predstavlja jedno rešenje koje je zamišljeno da bude minimalistički protokol. Protokol je bez autentikacije korisnika i koristi minimalan broj kontrolnih informacija kako bi što bolje funkcionisao i u veoma opterećenim mrežama gde komplikovaniji protokoli mogu imati funkcionalnih problema. U slučaju gubitka poruka u takvim uslovima ne narušava se funkcionisanje sistema. Gubljenje poruka usled opterećenja komunikacionog medijuma je prihvatljivije od velikog kašnjenja koje može nastati primenom komplikovanijih protokola. To je i sa stanovišta upotrebljivosti aplikacije bolje, jer je korisnicima intuitivnije da ponove akciju ukoliko ne vide reakciju na uređaju kojim upravljaju

nego da čekaju da li će akcija stići. Sastoji se od dva pod-modula, modul za oglašavanje na mreži i modula za prijem poruka.

Podmodul za oglašavanje na mreži je zamišljen tako da omogućava klijentskoj aplikaciji da u bilo kom trenutku može da pronađe aplikaciju uslužioca i samim tim uspostavi komunikaciju sa uređajem kojim se želi upravljati. Oglašavanje se vrši slanjem datagrama koji će primiti svi uređaji u lokalnoj mreži. Ovaj datagram sadrži IP adresu uređaja i broja prolaza na koji da se šalju datagrami sa podacima upravljanja. Ova poruka se ponavlja po isteku predefinisanog vremenskog intervala i na taj način ukoliko nekoliko ciklusa klijentska aplikacija ne primi ovu poruku znaće da je servis uslužioca prekinuo sa radom.

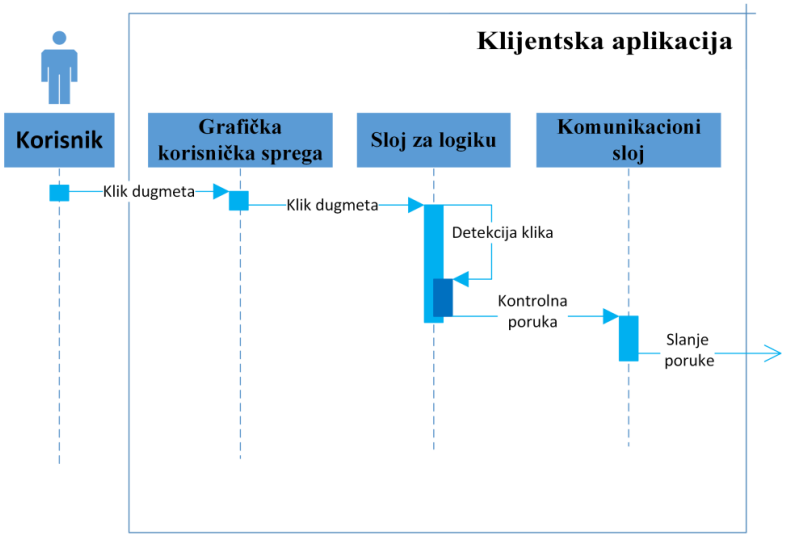
Podmodul za prijem poruka je programska petlja koja čeka na dolazne datagrame. Iz primljene poruke ovaj modul preuzima podatke i prosleđuje ih na obradu i izvršavanje glavnom modulu.



Slika 3.2 Prikaz algoritma funkcionisanja izvedenog protokola

3.2 Koncept korisničke aplikacije

Korisnička aplikacija sadrži grafičku spregu pomoću koje se vrši upravljanje DTV platforme zasnovane na Android-u. Funkcionisanje aplikacije se vrši kroz tri sloja. Sloja koji rukuje grafičkom korisničkom spregom (engl. *Graphical User Interface – GUI*), logičkim slojem (engl. *Middleware*) koji reaguje na interakciju korisnika i stvara poruku koja se prosleđuje aplikaciji uslužitelja preko komunikacionog sloja.



Slika 3.3 UML sekvencijalni dijagrama reakcije slojeva na korisničku interakciju

Aplikacija se sastoji iz nekoliko modula: glavni modul, modul za upravljanje računarskim mišem, modul za upravljanje tastaturom, modul za upravljanje daljinskim upravljačem i komunikacioni modul.

Glavni modul služi za pokretanje svih ostalih modula. Pored toga putem grafičke sprege pruža mogućnost izbora komunikacionog protokola.

Komunikacioni modul pruža transparentnu komunikaciju između dveju aplikacija. Ne proverava sadržaj komunikacionih poruka, nego ih prosleđuje logičkom sloju odgovarajućeg modula. On se izvodi na bazi UPnP ili izvedenog protokola zasnovanog na UDP protokolu u zavisnosti od izbora korisnika. Samo jedan komunikacioni modul može biti aktivan tokom jedne komunikacione sesije između klijentske i uslužilačke aplikacije. Svi ostali moduli imaju pristup komunikacionom modulu kako bi iz svakog modula bilo moguće slati poruke upravljanja aplikaciji uslužioca. Komunikacija drugih modula sa komunikacionim modulom se ne razlikuje za različite komunikacione protokole koji se koriste.

Modul za upravljanje računarskog miša je jednostavan. Zamišljen je da se oslanja se na grafički prikaz table osetljive na dodir za upravljanje ukazivača računarskog miša (engl. *Touchpad*). Sve kontrole se dalje preko logičkog sloja vezanog za ovaj modul prenose u komunikacioni modul na slanje.

Modul za upravljanje tastaturom je zamišljen da izgleda kao standardna tastatura sa „qwerty“ rasporedom tastera, bez numeričkih i funkcionih tastera, pošto za njima ne postoji potreba na uređaju baziranom na Android-u.

Modul za upravljanje daljinskim upravljačem predstavlja podskup tastera iz modula za upravljanje tastaturom organizovanih tako da preslikavaju funkcionalnost tipki sa klasičnog daljinskog upravljača.

4. Programsko rešenje

Ovo poglavlje daje prikaz programskog rešenja dveju aplikacija za upravljanje Android platformom simulacijom perifernih uređaja. Opisan je način funkcionalnosti pojedinih modula. Objašnjene su pojedine metode i prikazani su UML dijagrami klasa koje su od ključnog značaja za realizaciju ovog rada. Pored toga prikazan je i izgled aktivnosti klijentske aplikacije.

4.1 Aplikacija uslužioca

Aplikacija uslužioca je realizovana kao Android servis zbog toga što nema potrebe da sadrži korisničku spregu. Aplikacija je pisana delom Java i delom C programskom jeziku. Cilj aplikacije je da omogući upravljanje platformom na kojoj se nalazi svo vreme za vreme njenog rada. Kako bi se omogućilo automatsko pokretanje uslužioca iskorišćen je mehanizam Android platforme za automatsko pokretanje aplikacija po inicijalizaciji uređaja. To se radi traženjem specijalne dozvole u *AndroidManifest.xml* datoteci :

```
android:permission="android.permission.RECEIVE_BOOT_COMPLETED">
```

Pored toga neophodno je registrovati se sistemu za dobijanje obaveštejna o pokretanju platforme. To se se takođe dodaje u *AndroidManifest.xml* datoteku:

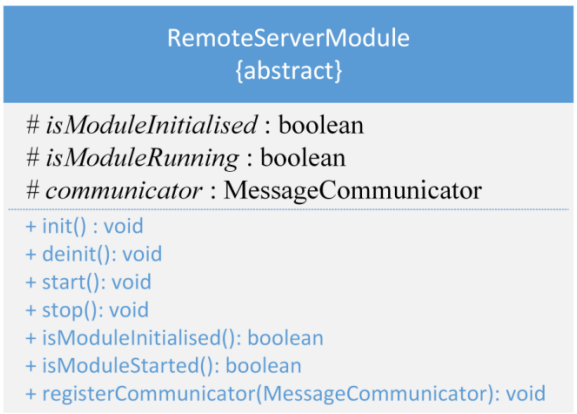
```
<intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Ovim krajnji korisnici ne vide kada se pokreće uslužilac, a pokrenut je za svo vreme rada sistema. Na taj način realizovana aplikacija daluje kao ugrađen uslužilac Android platforme.

Uslužilac se sastoji iz više modula:

- Glavni modul (realizovan klasom *RemoteServer*)
- Modul za realizaciju simulacije perifernih uređeja (realizovan klasom *ModuleUserInput*)
- Komunikacioni modul za izvedeni protokol (realizovan klasom *ModuleUDP*)
- Komunikacioni modul zasnovan na UPnP protokolu (realizovan klasom *ModuleUPnP*)

Pored ovih klasa koje reprezentuju funkcionalne module servisa, realizovana je apstraktna klasa *RemoteServerModule* koja obuhvata sve zajedničke metode i atribute koje modul uslužioca treba da obuhvata. Uvođenje novih modula u rešenje, to jest proširivanje skupa funkcionalnosti ovog rešenja se svodi na nasleđivanje ove klase i implementacije funkcionalnosti koja je karakteristična za taj modul.



Slika 4.1 UML dijagram apstraktne klase RemoteServerModule

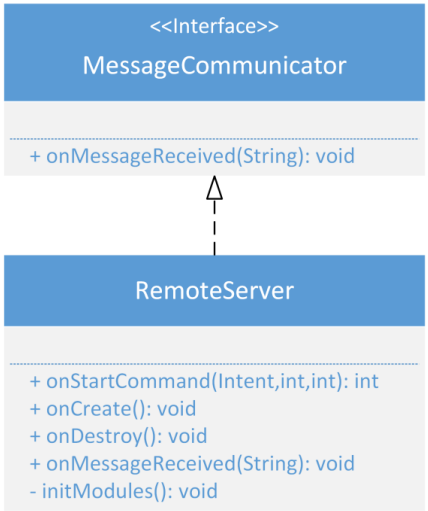
Ona se sastoji iz sledećih atributa i metoda:

- **boolean** *isModuleRunning* – atribut naznačava da li je modul pokrenut
- **boolean** *isModuleInitialised* – atribut naznačava da li je modul inicijalizovan
- **MessageCommunicator** *communicator* – atribut omogućava razmenu poruka sa glavnim modulom
- **void** *init()* – inicijalizuje modul
- **void** *deinit()* – deinicijalizuje modul
- **void** *start()* – pokreće modul
- **void** *stop()* – zaustavlja modul
- **boolean** *isModuleInitialised()* – vraća informaciju da li je modul inicijalizovan
- **boolean** *isModuleRunning()* – vraća informaciju da li je modul pokrenut

- **void** *registerCommunicator(MessageCommunicator)* – inicijalizuje objekat preko koga će se razmenjivati poruke sa drugim modulima

4.1.1 Glavni modul uslužioca

Glavni modul uslužioca je realizovan kroz klasu *RemoteServer*. Ova klasa predstavlja ulaznu tačku pri pokretanju servisa. Njen prvi zadatak je da inicijalizuje Android servis i predstavi ga sistemu. Nakon toga je zadužena za pokretanje modula i upravljanje servisom. Svaki modul pokreće u drugoj niti. Klasa kojom je realizovan ovaj modul nasleđuje interfejs *MessageCommunicator* i implementira njenu metodu *onMessageReceived(String)*. Pomoću ovog interfejsa se glavni modul registruje u ostalim modulima za primanje poruka od njih.



Slika 4.2 UML dijagram klase RemoteServer

Metode klase RemoteServer i njihova namena:

- **int** *onStartCommand(Intent,int,int)* – predstavlja ulaznu tačku u aplikaciji. Poziva je Android platforma po inicijalizaciji. Povratna vrednost ove metode govori platformi kako da se ponaša prema servisu u slučaju njegovog neregularnog zaustavljanja. U ovom rešenju ona je konstanta *START_STICKY*, što platformi govori da ponovo pokrene servis u slučaju njegovog neregularnog zaustavljanja. Druge vrednosti mogu biti: *START_CONTINUATION_MASK*, *START_NON_STICKY*, *START_REDELIVER_INTENT*, *START_STICKY_COMPATIBILITY*.
- **void** *onCreate()* – po pozivu ove metode se vrši inicijalizacija i pokretanje svih modula.

- **void** *onDestroy()* – po pozivu ove metode se vrši zaustavljanje i deinicijalizacija svih modula sa ciljem regularnog zaustavljanja servisa.
- **void** *onMessageReceived(String)* – pomoću ove metode komunikacioni moduli dostavljaju primljenu poruku glavnom modulu, gde glavni modul parsira tu poruku i vrši dalje upravljanje na osnovu njenog sadržaja.

4.1.2 Modul za simulaciju perifernih uređaja

Ovaj modul je realizovan kroz klasu *ModuleUserInput*. Zadatak ovog modula je da omogući simulaciju perifernih uređaja. Klasa se povezuje sa JNI biliotekom gde je realizovana komunikacija sa modulom „uinput“ Linux jezgra. Ovaj deo aplikacije je realizovan u programskom jeziku C, pošto se iz aplikativnog dela koji je pisan programskim jezikom Java ne može da se pristupi „uinput“ modulu.

Inicijalizacija ovog modula se ogleda u tome da se učita biblioteka u kojoj je realizovana komunikacija i registruje virtuelni uređaj u Linux jezgru. Registracija virtuelnog uređaja se prvenstveno obavlja tako što se otvori */dev/uinput* kao standardna tekstualna datoteka. Nakon toga se registruju ulazni kodovi koje virtuelni uređaj može da pošalje. Registracija kodova se vrši pomoću sistemske funkcije *ioctl*, koja predstavlja sistemski poziv za ulazno/izlazne operacije nad specifičnim uređajem. Kao ulazni parametar ova funkcija prima deskriptor datoteke uređaja i ostale parametre vezane za informaciju koju upisuje u uinput datoteku.

Registruju se tipovi događaja koji mogu da stignu od uređaja:

- *ioctl(uinput_fd, UI_SET_EVBIT, EV_KEY)* - događaj stisnute tipke
- *ioctl(uinput_fd, UI_SET_EVBIT, EV_REL)* - događaj pomeraja ukazivača na poziciju miša na ekranu

Dalje se registruju sve moguće vrednosti koje su povezane sa prethodno prijavljenim događajima. Neki od njih su:

- *ioctl(uinput_fd, UI_SET_RELBIT, REL_X)*
- *ioctl(uinput_fd, UI_SET_RELBIT, REL_Y)*
- *ioctl(uinput_fd, UI_SET_KEYBIT, BTN_LEFT)*
- *ioctl(uinput_fd, UI_SET_KEYBIT, KEY_ A)*
- *ioctl(uinput_fd, UI_SET_KEYBIT, KEY_1)*

Sledeća informacija koja je potrebna da bi se registrovao virtuelan uređaj u Linux jezgru je informacija o samom uređaju. Ona se sastoji od naziva uređaja, proizvođača, identifikacionog

broja i još nekoliko informacija koje mogu proizvoljno da se popune. Po upisu ovih informacija sistem zna koje sve događaje može uređaj da izazove. Sada je moguće kreirati taj virtuelan uređaj, što se radi pomoću funkcije:

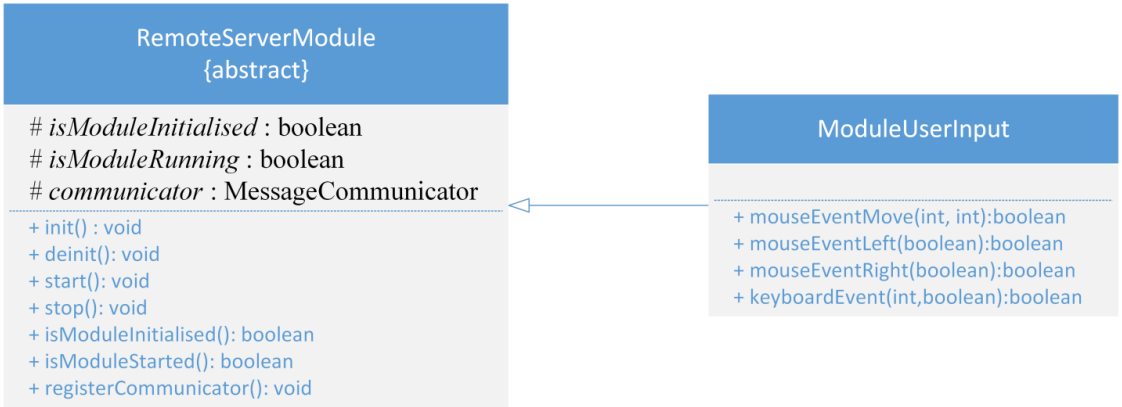
ioctl(uinput_fd, UI_DEV_CREATE, -1);

Komande primljene od ovog virtuelnog uređaja biće tretirane podjednako kao i komande koje su primljene od stvarnog, fizičkog, računarskog miša ili tastature. Svaka komanda koja se prosleđuje jezgru platforme gleda se kao ulazni događaj. Ulazni događaj u programskom smislu je realizovana kao struktura *input_event*. Polja ove strukture daju kompletan opis o događaju koji se prosleđuje sistemu. Polja strukture *input_event* su:

- *time* – vreme nastanka događaja
- *type* – tip događaja (npr. EV_REL za relativan pomeraj ukazivača računarskog miša za neku vrednost ili EV_KEY za stisnut taster bilo sa tastature ili tipke miša)
- *code* – kôd događaja (npr. KEY_A za stisnut taster A ili REL_X za pomeraj ukazivača miša po apcisi)
- *value* – vrednost događaja (npr. 1 za stisnut taster ili 0 za otpušten)

Deinicijalizacija modula se vrši deinicalizacijom uređaja, što se radi sledećim pozivom:

ioctl(uinput_fd, UI_DEV_DESTROY);

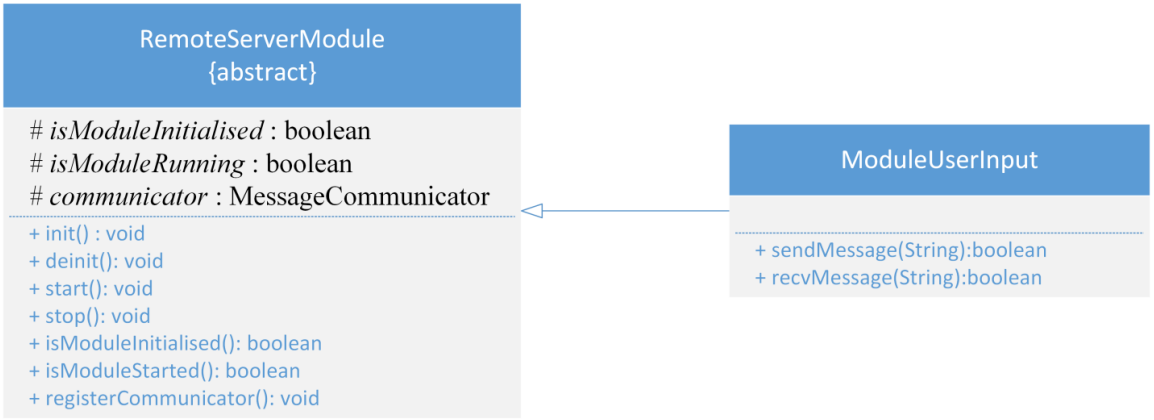


Slika 4.3 UML klasni dijagram modula za simulaciju perifernih uređaja

Metode prikazane na slici 4.3 klase *ModuleUserInput* poziva glavni modul sa ciljem prenošenja poruke upravljanja. Ovaj modul nema potrebe za registraciju objekta preko koga može da komunicira sa glavnim modulom. Povratna vrednost tih metoda predstavlja uspešnost prosleđivanja događaja perifernih uređaja koji je poslat.

4.1.3 Modul za komunikaciju pomoću UPnP protokola

Ovaj modul je realizovan pomoću klase *ModuleUPnP*. Realizacija UPnP protokola je data u UPnP biblioteci. Zahvaljujući tome nije bilo potrebno ništa posebno implementirati, nego samo u ispravnom redosledu pozivati funkcije koje su date kao sprega za korišćenje ove biblioteke. Oko UPnP biblioteke je napisana omotačka biblioteka koja vrši apstrakciju poziva potrebnih za realizaciju ove aplikacije. Omotačka biblioteka je napisana tako da omogućava JNI spregu sa Java kodom.



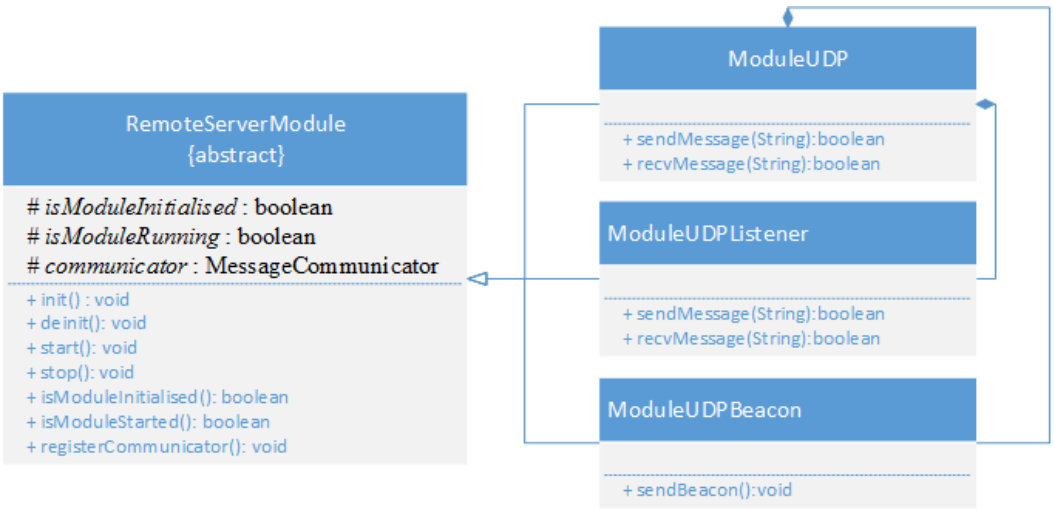
Slika 4.4 UML klasni dijagram UPnP komunikacionog modula

Sve operacije koje su vezane za protokol se izvršavaju u metodama nasleđene klase RemoteServerModule. Ostale metode:

- **boolean** *sendMessage(String)* – služi za slanje određene poruke klijentima. povratna vrednost predstavlja indikaciju o uspešnosti slanja.
- **boolean** *recvMessage(String)* – omogućava da se primljena poruka iz klijentske aplikacije prosledi do glavnog modula. Primljena poruka se prvo iz JNI sloja prenese u modul pozivom ove metode. Nakon toga se poruka u okviru ove metode prosledi glavnom modulu preko objekta klase *MessageCommunicator* koji je postavljen pri inicijalizaciji modula.

4.1.4 Modul za komunikaciju pomoću izvedenog protokola

Ovaj modul je realizovan pomoću klase *ModuleUDP*. Sastoji se iz dva podmodula, modula za prijem poruka (realizovan klasom *ModuleUDPListener*) i modula za obaveštavanje prisutnosti na mreži (realizovan klasom *ModuleUDPBeacon*). Obe klase nasleđuju *RemoteServerModule* pa se upravljanje njima obavlja kao u svakim drugim modulom. Oba modula se pokreću kao zasebne niti.



Slika 4.5 UML dijagram klasa komunikacionog modula za izvedeni protokol

Modul za obaveštavanje prisutnosti na mreži ima zadatak da tokom svog životnog veka na mrežu šalje datagrame koji sadrže informaciju o IP adresi uređaja na kome je pokrenuta aplikacija i broju prolaza na koji klijentske aplikacije treba da šalju svoje poruke. Slanje datagrama se odvija po isteku vremenskog intervala od 500ms. Ovaj vremenski interval je uzet sa obzirom kako se komunikacioni kanal ne bi suviše zatrpao datagramima, ali ipak pružio pravovremeni izveštaj o dostupnosti/nestanku na lokalnoj mreži. Svojim emitovanjem adrese klijenti uvek mogu da znaju kada se uslužilac dostupan, a kada nije. Prednost ovog emitovanja je što klijentska aplikacija može da se poveže automatski sa aplikacijom uslužioca bez ikakve interakcije sa korisnikom. Pošto neki protokoli takođe koriste ovakav sistem oglašavanja poruka koja se šalje mora da bude unikatna kako bi se razlikovala od poruka drugih poruka protokola za otkrivanje uslužioca (engl. *Server Discovery Protocol*). Format poruke:

AMUSE_REMOTE:<IP_ADRESA>:<BROJ_PROLAZA>

Modul za prijem poruka je dužan da osluškuje dolazeće datagrame koji su namenjeni aplikaciji i da primljene podatke prosledi glavnom modulu na obradu. Primljena poruka se kao i u slučaju UPnP modula prosleđuje kroz objekat klase *MessageCommunicator* koji je postavljen prilikom inicijalizacije modula. Takođe ima mogućnost slanja poruke klijentima u koliko ima potrebe za time, pružajući time dvosmernu asinhronu komunikaciju između aplikacija. Metoda za osluškivanje poruka je blokirajuća. Pravilno zaustavljanje ovog modula je omogućeno uništavanjem utičnice (engl. *Socket*) na kojoj se osluškuje za dolazeće poruke, čime se odblokira ovaj modul.

4.2 Klijentska aplikacija

Klijentska aplikacija je napravljena modularno da se lako mogu dodavati nove funkcionalnosti sa ciljem daljeg unapređenja rešenja. Moduli klijentske aplikacije su:

- Glavni modul (realizovan u okviru aktivnosti *MainActivity*)
- Modul za upravljanje računarskim mišem (realizovan kao zasebna aktivnost klasom *MouseActivity*)
- Modul za upravljanje tastaturom (realizovan aktivnosti *KeyboardActivity*)
- Modul za upravljanje daljinskim upravljačem (realizovan aktivnosti *RemoteCtrlActivity*)
- Komunikacioni modul (realizovan klasom *Client* i *ClientUPnP*)

Svaka aktivnost koja postoji u aplikaciji mora da se navede u *AndroidManifest.xml* datoteci. To se radi na sledeći način:

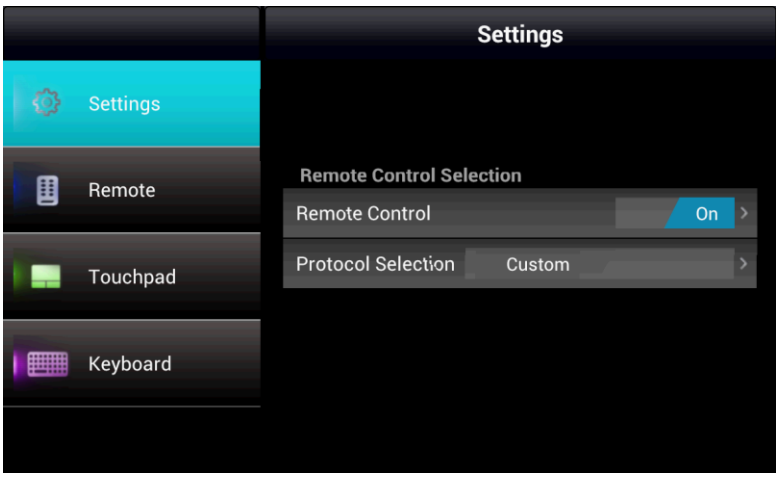
```
<activity
        android:name="NazivAktivnosti" >
</activity>
```

Pokretanje druge aktivnost iz trenutne aktivnosti se vrši pomoću „Intent“-a na sledeći način:

```
Intent intent = new Intent(<Klasa_aktivnost_iz_koje_se_pokreće>.this,
                <Klasa_aktivnost_koja_se_pokreće>.class);
startActivity(intent);
```

4.2.1 Glavni modul klijentske aplikacije

Ulazna tačka aplikacije je glavni modul. U okviru njega korisniku se pruža mogućnost da izabere komunikacioni protokol i da izabere način na koji želi daljinski da upravlja DTV platformom. Po izboru neka od tri upravljačka modula rad aplikacije prelazi na odgovarajuću aktivnost. Svaka od aktivnosti ima pristup komunikacionom modulu koji se koristi. Grafička sprega glavnog modula je realizovana pomoću standardnih Android grafičkih komponenti.



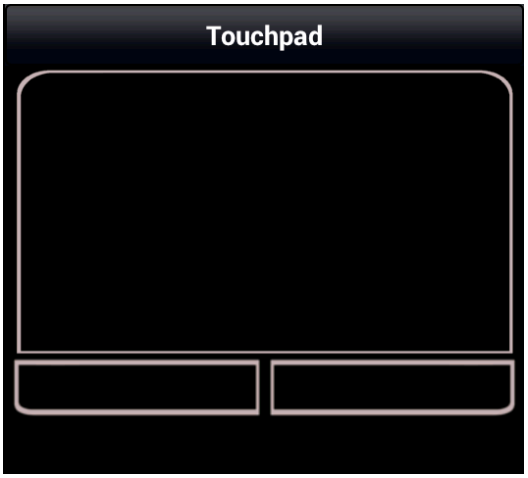
Slika 4.6 Grafička reprezentacija glavnog modula korisničke aplikacije

4.2.2 Modul za upravljanje računarskim mišem

Upravljanje računarskim mišem se vrši preko grafičke sprege koja izgleda kao tabla osetljiva na dodir za upravljanje ukazivača računarskog miša. Meri pomeranje prsta po ekranu prenosnog uređaja i šalje te informacije aplikaciji uslužioca. Pored osnovnog klika, pruža i mogućnost dugog klika koji je karakterističan za uređaje zasnovane na Android platformi. Logički sloj ovog modula se sastoji od osluškivača na dodir elemenata grafičke sprege. U okviru ovih osluškivača se formira poruka i prosleđuje komunikacionom modulu na slanje. Format poruka koje se šalju:

{MouseClick,<Left/Right>:<0/1>}

{MouseMove,<pomeraj_po_X_osi>:< pomeraj_po_Y_osi >}

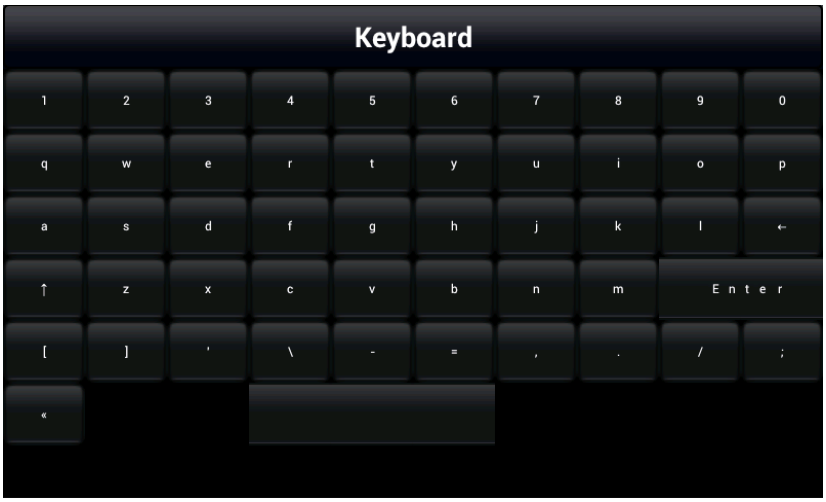


Slika 4.7 Grafička reprezentacija modula za rukovanje računarskim mišem

Grafička sprega se sastoji iz dva dugmeta (engl. *Button*), za levi i desni klik miša. Pored toga sadrži i pogled za prikaz slike (engl. *ImageView*) koji služi za praćenje pomeraja prsta sa ciljem upravljanja ukazivača računarskog miša.

4.2.3 Modul za upravljanje tastaturom

Upravljanje tastaturom vrši se preko grafičke sprege koja sadrži „qwery“ raspored tipki. Pruža mogućnost prenosa svih karaktera kao i sa standardne tastature bez numeričkih i funkcionalnih tastera koji nemaju primenu u Android-u. Logički sloj ovog modula predstavlja osluškivače koji čekaju dodir nekog dugmeta koji pretvaraju u događaj upravljanja i šalju preko komunikacionog modula uslužiocu.



Slika 4.8. Prikaz izgleda grafičke sprege kojom se upravlja virtuelnom tastaturom

Grafička sprega ovog modula se sastoji od dugmadi koji reprezentuju dugmadi sa „qwerty“ tastature. Logički sloj ovog modula je realizovan pomoću osluškivača na dodir koji su vezani sa dugmadima. U okviru ovih osluškivača se formira poruka i prosleđuje komunikacionom modulu na slanje. Format poruke je:

{KeyboardClick,<znak_koji je kliknut>}

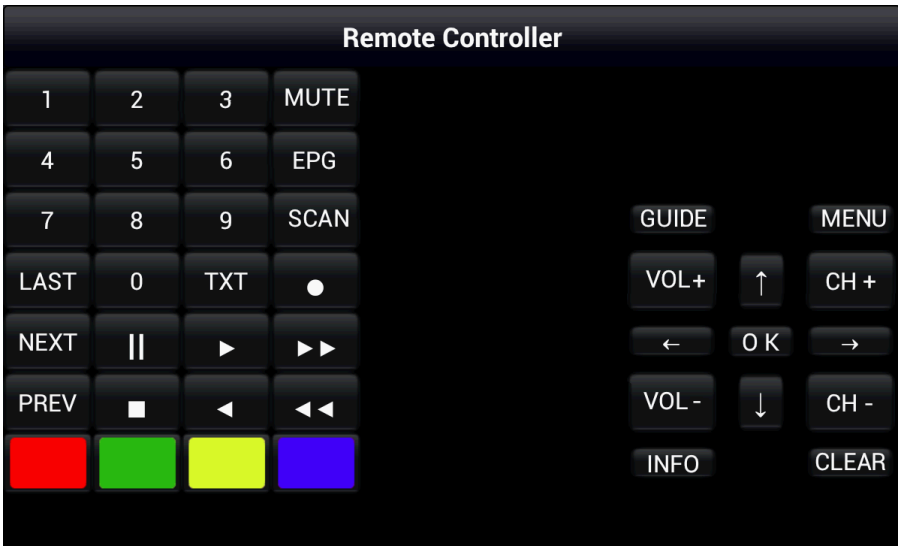
4.2.4 Modul za upravljanje daljinskim upravljačem

Modul za upravljanje daljinskim upravljačem je u stvari podskup tastera iz modula za upravljanje tastaturom. Tasteri su organizovani tako da preslikavaju funkcionalnost tipki sa klasičnog daljinskog upravljača koji se dostavlja uz DTV uređaj. Zahvaljujući tome kodovi koji se šalju za događaje tastature mogu iskoristiti i za simulaciju daljinskog upravljača. Sa aspekta platforme koja prima ove kodove reakcija na njih je ista kao i na kodove koje šalje standardni

daljinski upravljač namenjen za ovu platformu. Ovi kodovi međutim mogu biti različiti na drugim platformama. Zbog toga je realizovana klasa *KeyMap* koja u sebi sadrži enumeraciju svih mogućih kodova vezani sa njihovom funkcijom. U koliko platforma ne reaguje na neke kodove koje klijentska aplikacija šalje, moguće je podesiti kodove u ovoj klasi.

Grafička sprega ovog modula se sastoji od dugmadi koji reprezentuju dugmadi sa klasičnog daljinskog upravljača. Logički sloj ovog modula je realizovan pomoću osluškivača na dodir koji su vezani sa dugmadima. U okviru ovih osluškivača se formira poruka i prosleđuje komunikacionom modulu na slanje. Format poruke je:

{RemoteCtrlClick,<znak_koji je kliknut>}



Slika 4.9 Prikaz izgleda grafičke sprege realizovanog daljinskog upravljača

4.2.5 Komunikacioni modul

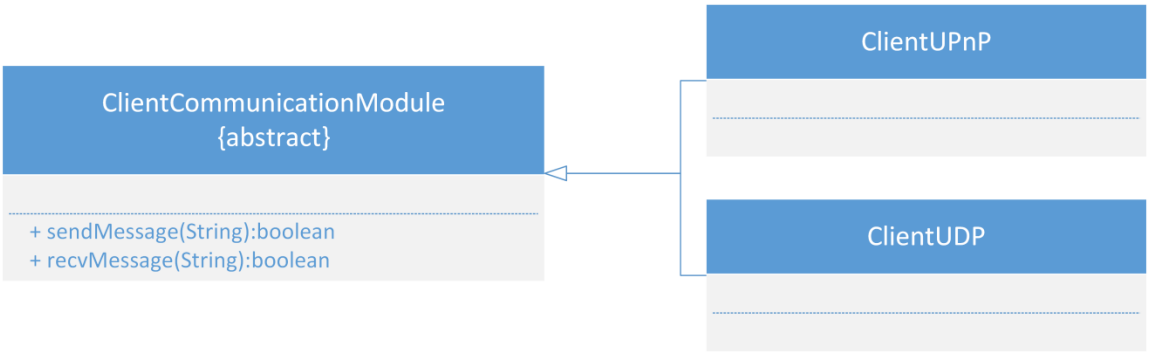
Komunikacioni modul omogućuje transparentnu komunikaciju između logičkih slojeva klijenta i uslužioca. Njegov zadatak je prenos poruke bez ikakve interpretacije. U isto vreme komunikacioni modul može da koristi samo jedan komunikacioni protokol.

U slučaju UPnP komunikacionog protokola rešenje komunikacionog sloja je jednostavno. Komuniklacija se oslanja na korišćenje poziva funkcija koje su ostavljene kao sprega za programere-korisnike UPnP biblioteke. Oko date UPnP biblioteke je napisana omotačka biblioteka koja stvara spregu sa potrebnim funkcionalnostima za ovu aplikaciju. Omotačka biblioteka je povezana pomoću JNI sprege sa ostatkom aplikacije. Realizacija komunikacionog modula koji koristi UPnP protokol je data u klasi *ClientUPnP*.

Izvedeni protokol za klijentsku aplikaciju takođe nije ništa komplikovaniji. Sastoji se iz dve niti. Jedna nit služi za osluškivanje poruka koje šalje uslužilac da oglasi svoju prisutnost na

mreži. Druga nit je zadužena za slanje poruka uslužiocu. Ona nije aktivna u koliko prva nit ne registruje prisustvo uslužioca na mreži. Takođe ni jedna funkcionalnost aplikacije nije dostupna dok nit za osluškivanje ne pronađe uslužioca na mreži. Traženje uslužioca ne počinje dok klijent to ne izabere u podešavanjima glavnog modula.

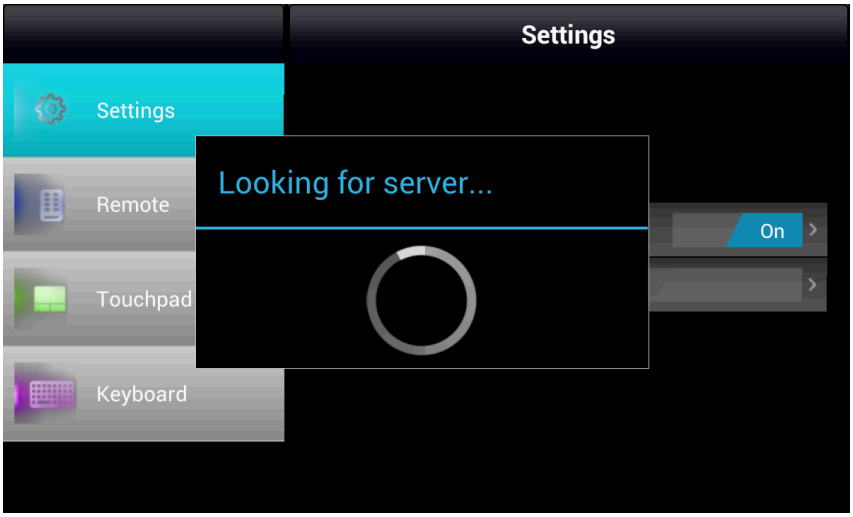
Obe ove klase se ponašaju na isti način sa gledišta ostalih modula. One se na identičan način pokreću, zaustavljaju i prosleđuju poruke. Zbog toga su zajedničke osobine izvučene u klasu *ClientCommunicationModule* koju treba da nasledi svaka klasa komunikacionog modula.



Slika 4.10 UML klasni dijagram komunikacionih klasa

Karakteristične metode za ovaj modul su *sendMessage(String)* i *recvMessage(String)*. One služe za slanje i prijem poruka ka i od uslužioca respektivno.

Tokom traženja uslužioca interakcija sa elementima aplikacije nije moguća, a dok se ne uspostavi veza sa uslužiocem aplikacija prikazuje dijalog koji obaveštava korisnika da je u toku uspostavljanje veze sa uslužiocem.



Slika 4.11 Grafički prikaz dijaloga koji se prikazuje za vreme uspostavljanja veze sa uslužiocem

5. Ispitivanje i verifikacija

Ispitivanje i verifikacija ispravnosti klijentske aplikacije je vršeno pomoću prenosnih uređaja zasnovanih na ARM i MIPS arhitekturi procesora, kao i na virtuelnoj mašini na kojoj je instaliran Android ICS (engl. *Ice Cream Sandwich*) zasnovan na x86 arhitekturi. Utvrđivanje ispravnosti aplikacije uslužioca je vršeno na razvojnim pločama BCM97241, BCM97435 VMS i BCM97435C (Arhitekture MIPS). Ispitivanje se sastoji iz dve faze. Prva faza obuhvata programsko ispitivanje funkcionalnosti, navigacije i komunikacije, koja su napisana u Java programskom jeziku, dok druga faza sadrži ispitivanje od strane krajnjih korisnika.

5.1 Programsko ispitivanje funkcionalnosti

Ispitivanje i verifikacija navigacije i interakcije sa korisnikom u Android aplikaciji se svodi na oponašanje upotrebe aplikacije od strane korisnika. Pri tome trebalo je ispitati da li se svaka komponenta, vidljiva na ekranu, može obeležiti i izabrati, da li je svaki pritisak dugmeta ispravno tretiran odgovarajućom reakcijom, odnosno kakve su posledice nasumičnog pritiskanja tastera, kao i reakciju aplikacija na iznenadan nestanak klijenta ili uslužioca i dalji nastavak rada. Za potrebe ispitivanja i verifikacije u Android aplikaciji je napravljen poseban modul koji je sačinjen od niza funkcija grupisanih po vrsti testova. Tako prepoznajemo sledeće grupe testova:

- Grupa testova koja ispituje da li se svaka stavka modula može obeležiti
- Grupa testova oponašanja pritisaka svih tastera klijentske aplikacije
- Grupa testova oponašanja nekontrolisanih pritisaka dugmića na klijentskoj aplikaciji
- Grupa testova koja ispituje rad aplikacija u slučaju iznenadnog nestanka klijenta ili uslužioca

Prva grupa testova ispituje da li se svaka stavka modula može obeležiti. Za potrebe ovog ispitivanja, sve komponente modula su povezane sa osluškivačem (eng. *Listener*) koji reaguje kada je ova komponenta obeležena. Pošto je u pitanju kontrolisan test gde se unapred znaju događaji koji će biti prosleđeni aplikaciji na tretiranje, lako se može izračunati koja komponenta će koliko puta biti obeležena. Na kraju izvršavanja ove grupe testova prikupljeni su rezultati od osluškivača povezanih sa određenim komponentama i upoređivanjem vrednosti zaključeno je da je ova grupa testova uspešno izvršena.

Druga grupa testova ima za cilj oponašanje pritiska svih tastera. Realizovani modul za ispitivanje omogućuje slanje svih komandi koje su moguće aplikaciji uslužioca. Takođe, prilikom izvršavanja ove grupe testova, pored osluškivača koji reaguju na odabir i obeležavanje određene komponente modula, od bitnog značaja je bilo postavljanje osluškivača na module svih nivoa, koji reaguju kada je određeni modul prikazan na ekranu, odnosno sklonjen. Na ovaj način ispitano je da li se pritiskom određenih tastera prikazuju određeni moduli, odosno da li su ti moduli funkcionalni. Prikupljanjem podataka od osluškivača i upoređivanjem sa događajima koji su prosleđeni aplikaciji uslužioca, zaključeno je da je ova grupa testova uspešno izvršena i da je funkcionalnost ispitanih modula ispravna.

Grupa testova koja se bavi oponašanjem nekontrolisanih pritisaka dugmića pravi nasumične događaje i prosleđuje ih aplikaciji na obradu. Na ovaj način, ispitani su zlonamerni pokušaji zbunjivanja i dovođenja aplikacije u ne konzistentno stanje. Ovom grupom testova je takođe ispitan i prelazak između modula aplikacije i vraćanje na prethodno stanje po povratku iz modula. Na ovaj način je pokazano da prilikom korišćenja različitih funkcionalnosti iz različitih modula nije moguće aplikaciju dovesti u ne konzistentno stanje.

Posledja grupa testova ima za cilj da ispita reakcije aplikacija u slučaju prestanka sa radom realizovanih aplikacija. Ispitan je normalan i nasilan prestanak rada klijentske aplikacije i ponovno povezivanje sa aplikacijom uslužioca. Isto tako je ispitan normalan i nasilan prestanak rada aplikacije uslužioca i njena mogućnost ponovnog usluživanja klijenata. Pored toga prilikom gašenja jedne aplikacije, praćena je reakcija u drugoj aplikaciji. Svo ovo ispitivanje je obavljeno korišćenjem oba komunikaciona protokola. Prikupljeni podaci od mrežnih modula obe aplikacije za svo ispitivanje iz ove grupe je potvrdilo ispravno funkcionisanje komunikacionih modula.

5.2 Ispitivanje od strane krajnjih korisnika

U okviru ovog ispitivanja okupljena je grupa ljudi različitog uzrasta sa različitim predznanjem o upravljanju Android aplikacijama.

Prilikom ispitivanja, primećeno je da je starijim ljudima bilo teže da se snađu prilikom korišćenja grafičke korisničke sprege aplikacije, odnosno trebalo im je više vremena da „nauče” da koriste aplikaciju. Za razliku od njih, mlađih ljudi koji svakodnevno koriste Android aplikacije sa lakoćom su uspevali da pronađu željenu stavku.

Nakon ispitivanja od krajnjih korisnika, većina ljudi je bila zadovoljna načinom na koji je projektovana grafička korisnička sprega klijentske aplikacije. Takođe su bili zadovoljni brzinom odziva i realizovanim upravljanjem na Android uređaju gde se nalazi aplikacija uslužioca. Korisnici su se takođe složili i da su grafičke komponente dobro predstavljene i lako prepoznatljive.

6. Zaključak

U radu je pokazan jedan način simulacije perifernih uređaja na platformi zasnovanoj na Android platformi. Rešenje omogućava iskorištenje celokupne funkcionalnosti koju nudi Android platforma. Realizovane aplikacije pružaju mogućnost upravljanja iz bilo kog dela kuće gde prenosni uređaj ima pristup lokalnoj mreži. Korišćenjem minimalističkog protokola omogućen je rad aplikacije i u veoma opterećenim uslovima komunikacionog medijuma preko koga se vrši upravljanje. Deo rešenja (konkretno biblioteke napisane u programskom jeziku C koje su realizovane za ovo rešenje) može se preneti na bilo koju platformu ili personalni računar koji se zasnovaju na Linux jezgru i time omogućiti daljinsko upravljanje ovim uređajima. Realizovano rešenje je nezavisno od fizičke arhitekture na kojoj se primenjuje.

Dato rešenje uslužioca može da se proširi funkcionalnostima karakterističnim za DTV okruženje u kome se primenjuje (npr. elektronski programski vodič, distribucija video sadržaja, teletekst, glasovno upravljanje i druge funkcije koje bi se pomoću ovog rešenja delile u lokalnoj mreži). Isto tako, realizovana klijentska aplikacija može da posluži kao osnova za prikaz različitog sadržaja vezanog za digitalnu televiziju. UPnP komunikacioni protokol omogućava osnovu za distribuciju različitog sadržaja za kojima može biti potrebe u daljem razvoju ovog rešenja.

7. Literatura

[1] Z. Mednieks, L. Dornin, G. B. Meike, M. Nakamura: „Programming Android“ , 2nd Edition, United States of America, O'Reilly Media,September 2012

[2] Marvell: „Whitepaper: Evolving Android for CE devices“, March 2013

[3] M. L. Murphy: „The Busy Coder's Guide to Android Development“, United States of America, CommonsWare, 2008

[4] J. Six, „Application Security for the Android Platform“ , 1st Edition, United States of America, O'Reilly Media, December 2011

[5] RFC768: https://tools.ietf.org/html/rfc768

[6] M. Jeronimo, J. Weast: „UPnP Design by Example, A Software Developer's Guide to Universal Plug and Play“, Intel Press, 2003

[7] UPnP standard: http://upnp.org/

[8] Rapid7: "Whitepaper: Security Flaws in Universal Plug and Play: Unplug, Don't Play.", February 2013

[9] D. P. Bovet, M. Cesati: „Understanding the Linux Kernel“ , 3rd Edition, United States of America, O'Reilly Media, 2005

[10] S. Goldt, S. van der Meer, S. Burkett, M. Welsh: „The Linux Programmer’s Guide“ , Germany, 1995

[11] S. Tanacković, D. Dejanović, M. Savić: „Jedno rešenje simulacije perifernih uređaja na platformi sa Android operativnim sistemom“, Srbija, Etran, Jun 2013