УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Александар Стефановић

Једно решење алата за верификацију програмске подршке пријемника дигиталног ТВ сигнала

МАСТЕР РАД

Нови Сад, 2014

SADRŽAJ

SPISAK SLIKA

SKRAĆENICE

CHAL - *Comedia Hardware Abstraction Layer*, Adaptacioni sloj programske podrške

DTV - *Digital Television, Digitalna televizija*

PID - *Packet Identifier*, Identifikacioni broj paketa

RTSP - *Real Time Streaming Protocol*, protokol za prenos podataka preko kontrolisane mreže

DVB-T - *Digital Video Broadcasting - Terrestrial*, standard za emitovanje digitalne televizije putem zemaljske antene

DVB-S - *Digital Video Broadcasting - Satellite*, standard za emitovanje digitalne televizije putem satelitske antene

DVB-C - *Digital Video Broadcasting - Cable*, standard za emitovanje digitalne televizije putem kabela

1. Uvod

U ovom radu je opisana realizacija alata za testiranje i verifikaciju rada osnovnih funkcionalnosti programske podrške prijemnika digitalnog TV signala. Dat je prikaz realizacije aplikacije upotrebom C programskog jezika koja je prilagođena za izvršavanje na platformi baziranoj na Android operativnom sistemu [1].

Od početka istorije računarskih sistema, testiranje je predstavljalo sastavni deo procesa razvoja softvera. Sa povećanjem složenosti programa i sistema, razvijale su se razne metode testiranja. Njihova uloga je da omoguće što efikasniji način provere rada programa, pronalaženjem njegovih funkcionalnih mana i grešaka.

Cilj realizovane aplikacije, odnosno alata za testiranje, je da omogući brz i efikasan način za verifikaciju sloja programske podrške koji se oslanja na funkcionalnosti fizičke arhitekture. Verifikovani sloj programske podrške je poznat pod imenom CHAL (Comedia Hardware Abstraction Layer). Realizovana aplikacija je pre svega korišćena u procesu prilagođavanja CHAL sloja Android platformi.

Rad je sačinjen od sedam poglavlja

Prvo poglavlje sadrži osnovne podatke o radu i opis rada.

Drugo poglavlje sadrži opis Android platforme i sloja programske podrške za čije je testiranje i verifikaciju razvijana data aplikacija. Takođe sadrži i kratak istorijat i razvoj metoda testiranja aplikacija i njihova uloga u procesu razvoja.

Treće poglavlje daje konceptualan pregled razvijanog alata za testiranje. Počevši od načina organizacije testnih slučajeva, načina njihovog izvršavanja, pa do pregleda svih podržanih komandi za njeno upravljanje.

Četvrto poglavlje sadrži opis implementacije alata za testiranje. Dat je opis organizacije testnih slučajeva, i način na koji se kontroliše njihovo izvršavanje i prikupljanje rezultata. Takođe je dat i opis funkcija koje su dostupne korisniku alata za pisanje novih testnih slučajeva.

Peto poglavlje daje kratak pregled realizovanih testnih slučajeva, pomoću kojih se verifikuje funkcionalnost programske podrške prijemnika digitalnog TV signala. Istaknute su najznačajnije grupe testnih slučajeva.

Šesto poglavlje daje kratak pregled onoga što je urađeno u ovom radi i kakvi su dalji pravci razvoja.

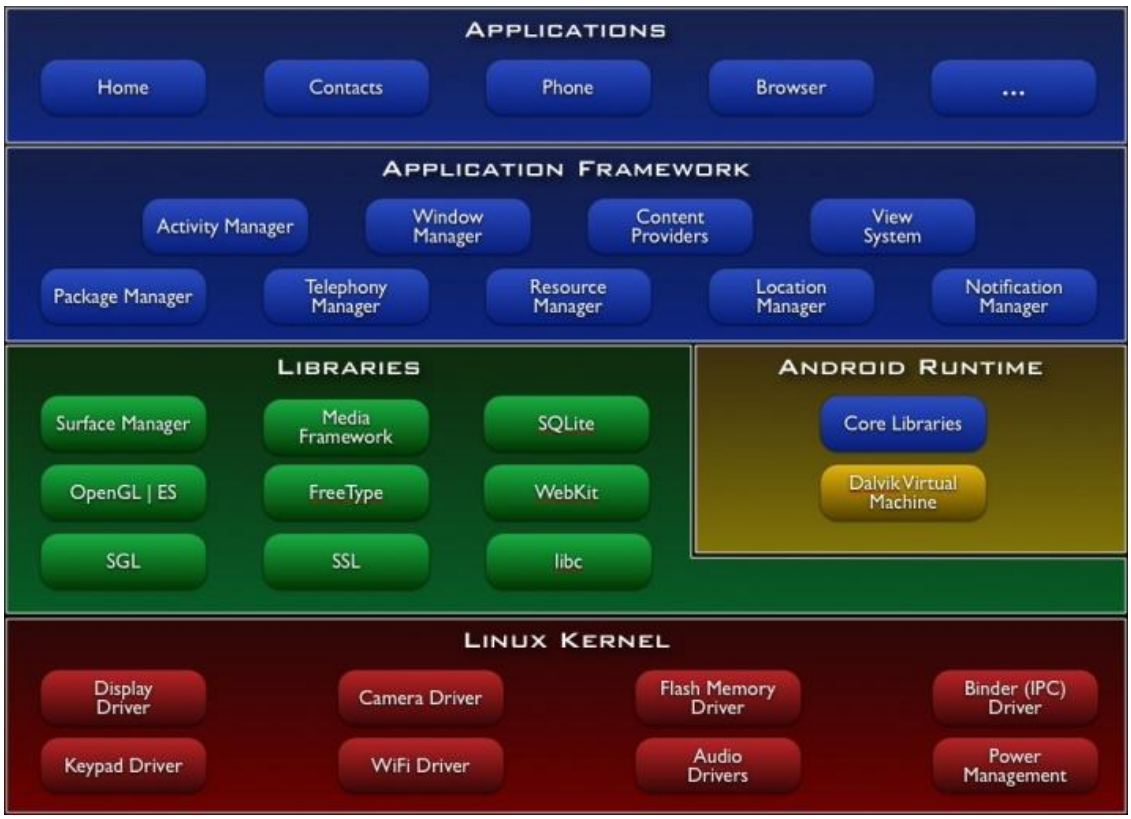
U sedmom poglavlju je dat spisak korišćene literature tokom izrade ovog rada.

2. Teorijske osnove

U ovom poglavlju dat je kratak opis Android platforme, opis uloge CHAL sloja u okviru programske podrške za DTV funkcionalnost. Takođe, dat je i kratak istorijat o razvoju metoda i tehnika testiranja programske podrške.

2.1 Android platforma

Android operativni sistem je prvobitno razvijan za mobilne uređaje sa ekranima osetljivim na dodir. Primer tih uređaja su pametni telefoni i tablični računari. Pored primarne upotrebe Android se često može naći i na drugim uređajima kao sto su digitalne kamere, kućni aparati i televizijski uređaji. Ovakvoj rasprostranjenosti oprativnog sistema je najviše doprinelo to sto je otvorenog koda. To omogućava proizvođačima uređaja da ga menjaju i prilagođavaju svojim potrebama [2]. Na slici 2.1 je dat prikaz arhitekture Android operativnog sistema.



Slika 2.1 Arhitektura Android operativnog sistema.

Najniži sloj programske podrške je baziran na Linux jezgru. Iz Linux jezgra su isključeni sistemi za rukovanje sa prozorima (X Window System) i standardni skup GNU biblioteka (glibc biblioteka), ali su dodata odgovarajuća proširenja koja su potreban Android platformi. Neka od proširenja su alarm, binder, ashmem, logger, itd.

Sledeći sloj sadrži biblioteke pisane u C i C++ programskom jeziku. Tu se nalaze biblioteke koje služe za iscrtavanje grafike (OpenGL/ES, SGL), dekodiranje video i audio zapisa (Media Framework), rukovanje bazama podataka (SQLite), itd. U sklopu biblioteka se nalazi i Android Runtime.

Android Runtime je dizajniran za uređaje sa ograničenom procesorskom moći, veličinom memorije i trajanjem baterije. Jezgro Android runtime-a je Dalvik virtuelna mašina koja je zadužena za pokretanje aplikacija pisanih u Java programskom jeziku. Svaka Android aplikacija se izvršava u okviru jednog procesa i ima svoju instancu Dalvik virtuelne mašine. Dalvik virtuelna mašina izvršava bajt kod u .dex formatu. Dex (engl. Dalvik Executable) predstavlja rezultat prevođenja .class datoteka. Dobijeni .dex bajt kod je znatno efikasniji od .class bajt koda sa stanovišta brzine izvršavanja.

Za razliku od Java virtuelne mašine, koja je zasnovana na steku, Dalvik virtuelna mašina je bazirana na registarskoj arhitekturi. Registarska arhitektura u proseku zahteva 47% manje instrukcija u odnosno na stek baziranu arhitekturu. Smanjenje broja instrukcija za posledicu ima

povećanje veličine koda za 25%. Uprkos povećanju veličine koda, cena dobavljanja instrukcija je zanemarljivo veća. Merenja efikasnosti su pokazala da se registarskom arhitekturom dobija ubrzanje izvršavanja instrukcija do 30%.

Na sledećem nivou se nalazi Application Framework, skup biblioteka i sistemskih aplikacija neophodnih za pokretanje i korišćenje korisničkih aplikacija. Ovaj deo programske podrške, kao i bazne biblioteke (engl. *core libraries*), je pisan u Java programskom jeziku.

Na vrhu Android arhitekture se nalaze aplikacije koje su dostupne krajnjem korisniku. Aplikacije su pisane u Java programskom jeziku. Postoje brojne sistemske aplikacije koje dolaze u sklopu Android uređaja (npr. Home, Contacts, Browser, Phone, itd.). Pored sistemskih, postoje i korisničke aplikacije razvijane od strane korisnika Android operativnog sistema. Veliki broj korisničkih aplikacija je dostupan preko Android servisa pod imenom PlayStore.

2.2 CHAL (Comedia Hardware Abstraction Layer)

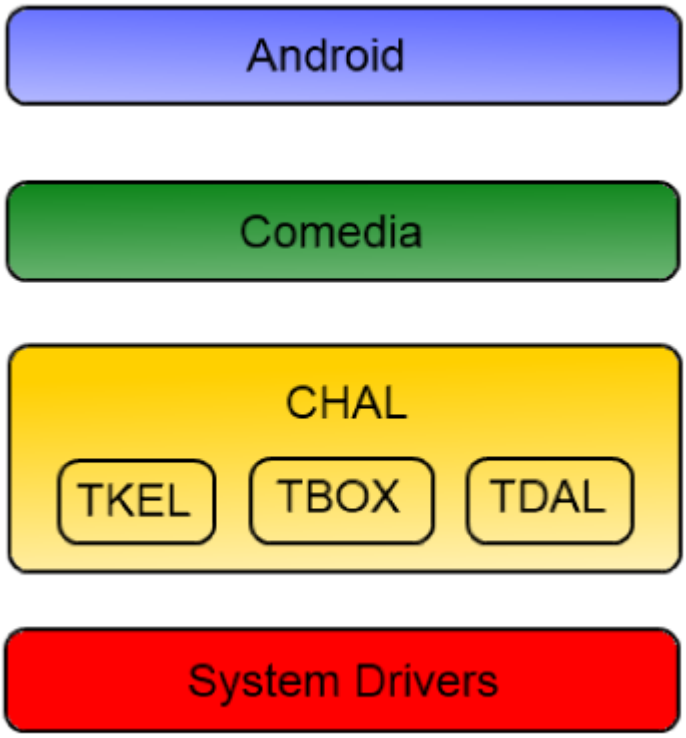
Najbitniji deo programske podrške prijemnika digitalnog TV signala, korišćene u ovom radu, predstavlja Comedia. U okviru Comedia programske podrške, realizovane su funkcije za kontrolu reprodukcije slike i zvuka, pristup informacijama o emisijama, kontrolu pristupa sadržaju, podršku za snimanje itd.

CHAL (eng. *Comedia Hardware Abdstraction Layer*) je sloj koji igra važnu ulogu kada je u pitanju prenosivost Comedia sprežnog sloja programske podrške na različite platforme. Čitavo jezgro Comedia-e, kao i njene sistemske funkcije, se ne oslanjaju neposredno na pozive funkcija rukovalaca, već se oslanjaju na pozive funkcija CHAL sloja. Implementacija rukovalaca se nalazi u jezgru operativnog sistema i razlikuje se od platforme do platforme. Na ovaj način je prilagođavanje Comedia-e na drugu arhitekturu svedeno na implementaciju CHAL sloja, pri čemu jezgro same Comedia-e ostaje netaknuto.

CHAL je sloj programske podrške koji se nalazi između programske podrške za televizijske prijemnike i operativnog sistema. Ovaj sloj raspolaže skupom funkcija koje apstrakuju funkcionalnost vezanu za korišćeni operativni sistem i fizičku arhitekturu i samim tim je zavistan od fizičke arhitekture. Položaj CHAL sloja u sistemu je dat na slici 2.2.

CHAL se sastoji od sledećih modula:

- TBOX (Tool Box),
- TKEL (Thin Kernel Encapsulation Layer),
- TDAL (Thin Driver Adaptation Layer)



Slika 2.2 Prikaz položaja CHAL sloja.

2.2.1 TBOX

TBOX modul predstavlja spregu pomoću koje se vrši praćenje i merenje vremena izvršavanja funkcija. Takođe, dostupne su i funkcije za ispis poruka u konzolu ili u određene datoteke u svrhu pronalaženja grešaka.

2.2.2 TKEL

Ovaj modul enkapsulira funkcije koje su specifične za korišćeni operativni sistem (sinhronizacija, kritične sekcije, semafori, komunikacija, rukovanje programskim nitima, itd.).

2.2.3 TDAL

TDAL modulom su obuhvaćeni svi blokovi koji su neophodni za izvršavanje programske podrške za televizijske prijemnike (zvuk, slika, demultiplekser, demodulator, itd.). Ovaj modu takođe abstrakuje rukovaoce (eng. drivers). Svaki blok fizičke arhitekture DTV prijemnika poseduje odgovarajući TDAL rukovalac. Ovaj modul omogućava korišćenje fizičke arhitekture prijemnika televizijskog signala od strane viših slojeva programske podrške, bez potrebe za dodatnim poznavanjem detalja realizacije te arhitekture. Navedeni su neki od blokova koji se nalaze u ovom modulu:

- TDAL_AV - Kontroliše jačinu zvuka, rukuje videom, itd.

- TDAL_AVS - Upravlja formatom slike i zvuka. Rukovaoci kontrolišu i usmeravaju signale koji dolaze od unutrašnjeg dekodera, ili spoljašnjih periferija.
- TDAL_DMX - Sadrži funkcije koje omogućavaju izdvajanje podataka za zvuk, sliku i dodatne informacije iz digitalnog prenosnog toka. Do željenih informacija se dolazi podešavanjem filtera sa određenim PID-om koji je definisan DVB standardom [4].
- TDAL_DMD - Uloga ovog modula je da upravlja fizičkim mrežnim sprežnim modulom - NIM (tjuner i demodulator). Podešava parametre za demodulaciju, zavisno od korisničkog standarda / tipa NIM-a (DVB-T/T2, DVB-S, DVB-C). Obavlja pretraživanje zadatog frekvencijskog opsega (eng. scanning). Obaveštava ostale module o važnim događajima vezanim za prijem signala (izgubljen signal, progres pretraživanja i sl.). Omogućava pristup i kontrolu parametara kvaliteta signala (AGC, BER, SNR i sl.) [5].
- TDAL_GFX - Predstavlja grafičku spregu. Upravlja grafičkim regionima, paletama boja i iscrtavanjem slika [6].
- TDAL_KBD - Upravlja ulazima sa tastature.
- TDAL_TS - Predstavlja spregu za upravljanje prenosnim tokovima podataka.
- TDAL_OUTPUT - Ima ulogu da obezbedi spregu između programske podrške televizijskog prijemnika i izlaza slike.
- TDAL_DISP - Njegova uloga je da obezbedi spregu između programske podrške televizijskog prijemnika i prikaza slike [7].

2.3 Testiranje i verifikacija

Ranih dana razvoja softvera, osnovni način testiranja je bilo debagovanje. Traženje i uklanjanje grešaka (eng. bug) bilo je isključivo zadatak programera koji je pisao kod. Cilj ovog procesa je bio da se aplikacija dovede u stanje u kojem radi bez kritičnih grešaka i rušenja sistema.

Krajnji korisnici su, u to vreme, igrali veoma malu ulogu u testiranju aplikacije. Na njima je bilo da nauče da koriste program kako bi dobili željene rezultate. Ukoliko program ne bi proizvodio željene rezultate, korisnici bi morali da pronađu način kako da prilagode obrađivane podatke, ili tehniku korišćenja programa kako bi došli do željenog cilja.

Ideja o razdvajanju debagovanja i testiranja softvera je uvedena od strane Glenford J. Mayersa 1979 godine. U ovom periodu testiranje je bilo destruktivno orijentisano. Što znaci da su pod uspešnim testovima bili podrazumevani oni testovi koji su pronalazili grešku.

U današnje vreme testiranje predstavlja metodu kojom se demonstrira da određeni softver zadovoljava datu specifikaciju. Testovi su orijentisani prema otkrivanju grešaka i mana, kao i prema njihovoj prevenciji.

Testiranje danas predstavlja značajan deo u procesu razvoja softvera [8]. Postoje razne metode testiranja, koje se mogu razdvojiti u dve grupe: testiranje bele i crne kutije.

Prilikom testiranja metodom bele kutije (eng. white-box), testira se unutrašnja struktura programa, za razliku od funkcionalnosti dostupne krajnjim korisnicima [9]. Prilikom dizajniranja ovakvih testnih slučajeva potrebno je detaljno poznavanje implementacije testiranog sistema ili programa. Postoje razne tehnike u okviru metode bele kutije. Neke od njih se zasnivaju na pisanju testnih slučajeva koji pokrivaju izvršavanje svake linije koda bar jedanput, prolaze kroz sve moguća grane programa i pokrivaju sve kombinacije uslova. Takođe postoje tehnike kojima se namerno uvode greške prilikom izvršavanja programa, kako bi se proverilo rukovanje greškama i unapredila robusnost programa.

U okviru metode crne kutije (eng. *black-box*) testirani softver se posmatra kao crna kutija, i ispituje se njegova funkcionalnost bez znanja on unutrašnjoj implementaciji [10]. Testovi su pisani u skladu sa specifikacijom i njihov cilj je da provere funkcionalnost. Prilikom testiranja dostupni su samo ulazi, izlazi i specifikacija. Na osnovu čega se proverava funkcionalnost, upoređivanjem dobijenih rezultata sa odgovarajućim očekivanim rezultatima.

Svrha testiranja softvera nije samo da bi se pronašle i ispravile greške, već služi i kao alat za validaciju i verifikaciju. Razvoj nekog softvera nije završen sve dok ne prođe kroz temeljno testiranje i verifikaciju, čiji je cilj da pokažu da je implementacija u skladu sa specifikacijom.

Prilikom testiranja, jedan od važnih aspekata je i automatizacija. Ručno testiranje zahteva pojedinačno pokretanje testova. Prilikom pokretanja testova, dobijeni rezultati se porede sa očekivanim i beleže se odstupanja. Za razliku od ručnog, automatsko testiranje predstavlja uzastopno izvršavanje velikog broja testova. Tom prilikom se postiže isti efekat sa znatno većom preciznošću i efikasnošću.

U okviru razvoja softvera postoji česta potreba za ponovnim pokretanjem testova. Prilikom svake značajnije izmene izvornog koda potrebno je testiranje. Ručno testiranje oduzima značajno vreme. Automatskim testiranjem se vreme izvršavanja ponavljanih testova može smanjiti sa nekoliko dana na svega nekoliko sati. Automatskim testiranjem se može postići znatno veća pokrivenost raznih slučajeva. Dugački testovi, koji se obično izbegavaju prilikom ručnog testiranja, se mogu izvršavati bez nadzora. Takođe, testovi se mogu istovremeno pokretati na više uređaja sa različitim podešavanjima. Još jedna od prednosti automatskog testiranja je ta što se ljudski faktor može u potpunosti izbaciti iz procesa testiranja. Svakim ponovnim pokretanjem automatskih testova, izvršavaju se svi koraci bez mogućnosti da se neki od njih zaboravi.

3. Koncept rešenja

U ovom poglavlju je dat kratak opis rešenja implementacije aplikacije za testiranje. Dat je opis osnovnih komponenti aplikacije, kao i opis organizacije testnih slučajeva.

Realizovana aplikacija omogućava automatsko pokretanje velikog broja testnih slučajeva. Jedan od zahteva je taj da se omogući interakcija sa korisnikom prilikom izvršavanja testova. U nekim specifičnim testovima, od korisnika se može zahtevati određena akcija, koja se ne može programski izvršiti. Kao na primer uključivanje kabela.

Alat za testiranje se sastoji iz dva osnovna dela. To su rukovalac testova i pokretač testova.

U okviru rukovaoca testova se nalazi programska podrška za organizaciju i pisanje testnih slučajeva. Sadržane su funkcije koje se koriste u okviru testnih slučajeva za provere određenih logičkih izraza i očekivanih rezultata. Kao i funkcije pomoću kojih se omogućava interakcija sa korisnikom u toku testiranja.

Pokretač testova predstavlja najveći deo aplikacije i služi za pokretanje, odnosno izvršavanje testova. U okviru pokretača testova je takođe realizovana sprega za upravljanje sa aplikacijom.

3.1.1 Organizacija Testova

Testovi su organizovani u vidu stabla. Svaki čvor stabla (eng. node) može i ne mora imati potomke (eng. children). U slučaju da čvor stabla nema potomke on predstavlja jedan testni slučaj (eng. test case), dok u slučaju da ima potomke, predstavlja skup testova (eng. test suite). Dat je primer izgleda stabla u pisanoj formi:

```
root (0)
  TestTKEL (0 0)
   TestMutex (0 0 0)
    TestBasicMutex (0 0 0 0)
    TestMutexMultipleLockPerThread (0 0 0 1)
   TestSemaphore (0 0 1)
    TestBasicSemaphore1 (0 0 1 0)
    TestBasicSemaphore3 (0 0 1 1)
  TestTDAL (0 1)
   Test_TDAL_AV (0 1 0)
    TestAudioConfig (0 1 0 0)
    TestVideoConfig (0 1 0 1)
    TestStillPicture (0 1 0 2)
```

U navedenom primeru stabla se može primetiti da je svaki čvor stabla identifikovan sa imenom i odgovarajućim brojem. Broj čvora se određuje na osnovu njegovog položaja u stablu. Početak stabla, odnosno koren (eng. root), predstavlja početni čvor stabla i označen je samo jednim brojem: 0. Svakom novom potomku određenog čvora se dodeljuje novi broj. Dodeljeni broj ima početnu vrednost: 0. Na primer, potomci korena stabla, `TestTKEL(0 0)` i `TestTDAL(0 1)`, imaju po dva broja. Prilikom dodavanja novih potomaka istom čvoru, svaki novi potomak ima poslednji broj uvećan za jedan, u odnosu na prethodnog potomka istog čvora. Na primer pošto su `TestTKEL(0 0)` i `TestTDAL(0 1)` potomci istog čvora, `TestTDAL(0 1)` ima poslednji broj uvećan. Broj testa prvenstveno sluzi za lakše odabiranje prilikom izvršavanja.

3.1.2 Upravljanje aplikacijom

Pri pokretanju aplikacija dobija se komanda linija preko koje se unose komande. Komande su predstavljene u vidu jedne reči i imaju skraćenu verziju u vidu jednog slova i/ili broja. Većina komandi takođe ima i dodatne parametre. Postoje tri vrste parametra:

- *options* - Predstavlja celobrojni parametar pomoću kojeg se moze odabrati određeni skup testova na koje se određena komanda odnosi. Vrednosti parametara su:
  - 1 – Komanda se odnosi samo na grupe testova, a ne i na sve pojedinačne testova.
  - 2 – Komanda se odnosi na sve uspešne testove.
  - 3 – Komanda se odnosi na sve neuspešne testove.
  - 4 – Komanda se odnosi na testove koji još nisu izvršeni.

Parametar je prisutan kod komandi koje izlistavaju testove.

- *node* - Predstavlja ime određenog čvora stabla na čije potomke se data komanda za izlistavanje odnosi.
- on/off ili 0/1 - Služi za uključivanje ili isključivanje određene funkcionalnosti na koju se odabrana komanda odnosi.

Dostupne komande se mogu izlistati unosom "help" komande ili ukratko "?". Dostupne komande su:

- list - komanda izlistava sve dostupne testove u vidu stabla. U zavisnosti od parametra, ima opciju da izlista određene skupove testova. Takođe ima opciju da izlista određenu grupu testova koja počinje sa specificiranim čvorom stabla.
- interactive - uključuje ili isključuje interaktivni mod.
- stopatfirsterror - uključuje ili isključuje način rada programa u kome se izvršavanje testova prekida nakon prvog neuspešnog testa.
- result - služi za izlistavanje rezultata testova pri čemu se dobija pregled o izvršenju testova, odnosno da li je dati test izvršen, i dobija se broj uspešnih i neuspešnih testova.
- reset - omogućava resetovanje rezultata određene grupe testova.
- cd - komanda služi za promenu početnog čvora stabla, odnosno korena, i omogućava korisniku da izvršava samo određene grupe testova.
- help - izlistava sve dostupne komande.
- quit - završetak rada programa

3.1.3 Izvršavanje testova

Testovi se pokreću unošenjem njihovih imena u konzolu. Pored unošenja imena, testovi se mogu pokretati i unošenjem njihovog broja. Prilikom pokretanja određenog testa, svi potomci datog testa će takođe biti izvršeni. Na primer unošenjem komande:

```
CMD>TestTKEL
```

će biti izvršeni svi testovi koji se nalaze u datoj grupi testova. Testovi se izvršavaju tako što se prvo izvršavaju svi potomci čvora na koji se prvo naiđe, odnosno po principu „depth-first“ pretrage.

Neki testovi, prilikom izvršavanja, mogu zahtevati aktivno učestvovanje korisnika tokom njihovog izvršavanja. Takve testove možemo nazvati interaktivnim testovima. U ovoj aplikaciji

se razlikuju dve vrste interaktivnih testova: testovi koji zahtevaju interaktivnost radi same provere ispravnosti nekog testnog slučaja i testovi koji zahtevaju funkcionalnu interaktivnost.

Pod prvom vrstom interaktivnih testova se podrazumeva postavljanje upita korisniku, na čiji se odgovor čeka pre daljeg izvršavanja testa. Pitanje može biti bilo kakvog tipa. Ukoliko se testira ispravnost prikazivanja slike, od korisnika se može zahtevati da odgovori na pitanje. Na primer, da li se stvarno na ekranu može videti očekivana slika.

Kod funkcionalno interaktivnih testova se od korisnika zahteva interakcija sa samim uređajem, na kojem se dati testovi izvršavaju (pritisak dugmeta na daljinskom upravljaču, uključivanje mrežnog kabela, itd...).

Pre pokretanja testova, korisnik moze da izabere da li će se testovi pokretati u interaktivnom modu ili ne. U slučaju da je interaktivni mod isključen, testovi koji zahtevaju interakciju u vidu pitanja, će biti izvršeni tako što će svako pitanje biti zamenjeno sa vremenskim intervalom od jedne sekunde. Na ovaj način se postiže izvršavanje testova bez prekida, i bez čekanja na odgovor korisnika. Testovi, koji zahtevaju interakciju sa samim uređajem, neće biti izvršeni u slučaju isključenog interaktivnog moda.

4. Programsko rešenje

U ovom poglavlju je dat detaljan opis realizacije programskih modula, odnosno realizacija funkcija u okviru programske podrške zadužene za upravljanje, izvršavanje i prikupljanje rezultata testova. Takođe je dat opis realizacije nekih od ključnih testova neophodnih za verifikaciju ispravnosti implementacije CHAL programskog sloja.

4.1 Organizacija testova

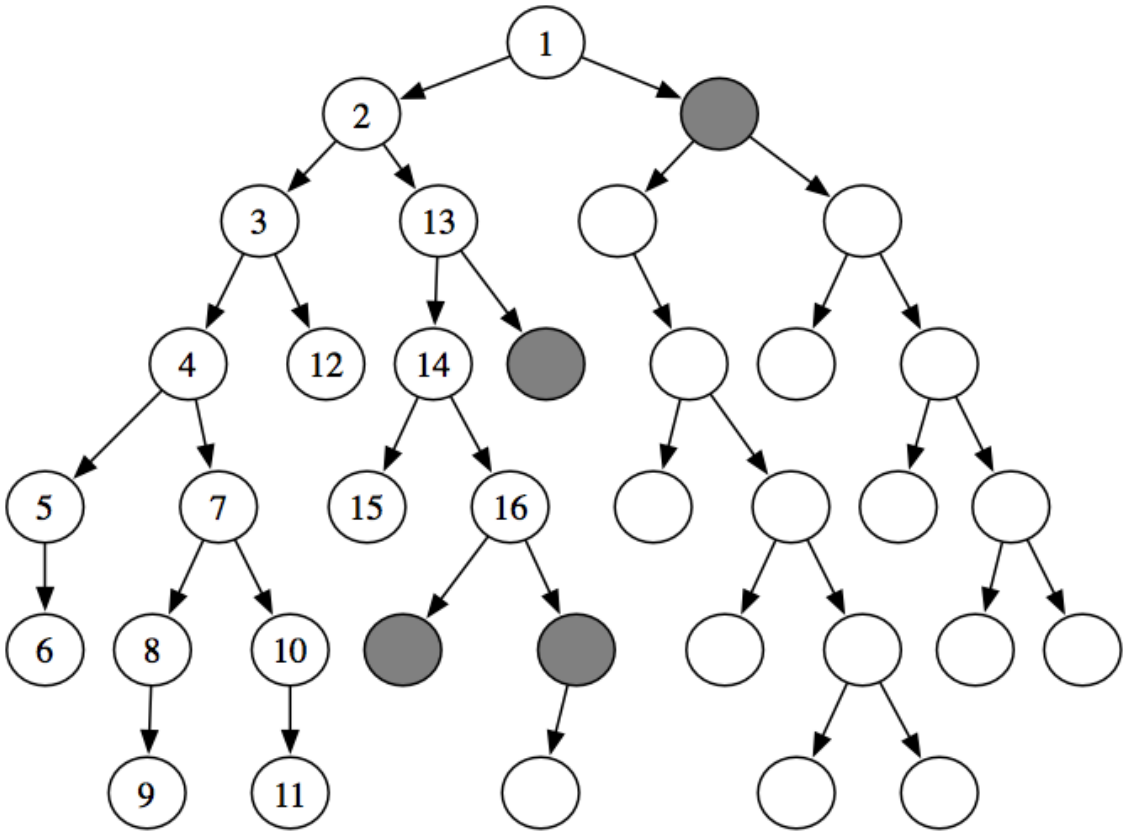
Alat za testiranje je realizovan u C programskom jeziku. Testovi su organizovani u obliku stabla. Stablo se sastoji od čvorova, koji su predstavljeni strukturom *tTestNode*.

```
typedef struct {
    char *msName;
    char *msPurpose;
    int   mbIsFunctionalyInteractive;
    tTestCase_Execute mExecuteFunction;
    void **mTestNodeVector;
} tTestNode;
```

Svaki čvor predstavlja jedan testni slučaj (eng. test case) ili skup testova (eng. test suite), i identifikovan je jedinstvenim imenom **msName*. Potrebno je da svaki čvor ima jedinstveno ime zbog mogućnosti pokretanja testova na osnovu njihovog imena. Ukoliko više čvorova ima isto ime, uvek će se izvršavati onaj čvor koji se prvi pronađe prilikom pretrage stabla. Pored imena, čvor sadrži i kratak opis koji ukratko opisuje njegovu svrhu ili sadržaj u slučaju da predstavlja grupu testova. Najznačajniji članovi strukture *tTestNode* su *mExecuteFunction* i *mTestNodeVectr*. U slučaju da čvor predstavlja testni slučaj, potrebno je da se definiše član

mExecuteFunction, koji predstavlja funkciju koja će biti izvršena u okviru datog testnog slučaja. Dok, u slučaju da čvor predstavlja grupu testova, potrebno je da definiše član *mTestNodeVectr*, koji predstavlja niz čvorova odnosno potomaka. Čvor ne može u isto vreme da bude i testni slučaj i da sadrži potomke, odnosno da predstavlja grupu testova. U slučaju da su oba člana definisana, čvor će uvek biti interpretiran kao grupa testova. Kao indikator, da test zahteva interakciju sa korisnikom aplikacije, služi član *mbIsFunctionalyInteractive*.

Prolazak kroz sve čvorove stabla, čija je putanja određena vezama između predaka i potomaka čvorova, se naziva "šetnja stablom" (eng. walking the tree). Funkcija kojom se inicira prolazak kroz stablo je *WalkInTree*. Prilikom prolaska kroz stablo, moguće je specificirati određeni početni čvor. Algoritam pomoću kog se prolazi kroz čvorove stabla je poznat pod nazivom "depth-first" pretraga. Pretraga se započinje od korena stabla. Prvo se posećuju svi čvorovi na jednoj grani sve dok se ne dođe do poslednjeg čvora koji nema potomke. Nakon prolaska cele grane po istom principu se prelazi na svaku sledeću granu. Na slici 4.1 je ilustrovan ovakav način pretrage stabla.



Slika 4.1 Ilustracija “depth-first” pretrage.

Funkcija *WalkInTree* započinje prolazak kroz stablo od određenog početnog čvora. Takođe, vrši inicijalizaciju promenljivih pomoću kojih se prati prolazak kroz stablo. Sam prolazak kroz stablo se izvršava rekurzivno, pozivom funkcije *RecursWalkInTree*. Rekurzija se nastavlja sve dok se ne prođe kroz svaki čvor stabla ili dok se ne prekine pod određenim uslovom.

```
typedef void (*WalkTreeCallBack)(tTestNode *aNode, int *pbKeepWalking);

void WalkInTree(tTestNode *aTestRoot, WalkTreeCallBack aPrefixedCallback,
                WalkTreeCallBack aPostfixedCallback, int *pbKeepWalking)
```

Prilikom poziva *RecursWalkInTree*, prosleđuju se dve funkcije povratnog poziva (eng. callback). Prva prosledjena funkcija se poziva pre svake sledeće iteracije prolaska kroz stablo, odnosno pre svake sledeće rekurzije. Dok se druga funkcija poziva prilikom povratka rekurzije.

```
void RecursWalkInTree(tTestNode *aTestRoot,
                      WalkTreeCallBack aPrefixedCallback,
                      WalkTreeCallBack aPostfixedCallback,
                      int *abKeepWalking)
```

Parametri:

- *aTestRoot* – Početni član stabla od kog se započinje iteracija stablom
- *aPrefixedCallback* – Pokazivač na funkciju koja se izvršava pre svake sledeće iteracije
- *aPostfixedCallback* – Pokazivač na funkciju koja se izvršava pre samog povratka svake rekurzije
- *pbKeepWalking* – Predstavlja indikator da li će se prolazak kroz stablo nastaviti nakon određene iteracije

4.2 Izvršavanje testova

Funkcija *ExecuteTestTree* se poziva prilikom izvršavanja testova. Kao parametar funkciji se prosleđuje određeni čvor stabla od kog počinje izvršavanje. U slučaju da korisnik prilikom pokretanja izvršavanja testova ne odabere određeni čvor, izvršavanje započinje od samog korena stabla i izvršavaju se svi testni slučajevi.

```
typedef enum
{
    TESTMANAGER_NO_ERR,
    TESTMANAGER_NOT_DONE,
    TESTMANAGER_BAD_ARG
}TESTMANAGER_err;

TESTMANAGER_err ExecuteTestTree(tTestNode *aTestRoot)
```

Parametri:

- *aTestRoot* – Predstavlja početni član stabla, od kog počinje izvršavanje testova

Povratna vrednost:

- *TESTMANAGER_NO_ERR* – Izvršavanja testova je uspešno

Izvršavanje testova se zasniva na prolaženju kroz stablo pomoću funkcije *WalkInTree*. Prilikom čega se za parametar *aPrefixedCallback* prosleđuje funkcija *ExecuteTreeCallback*. Funkcija *ExecuteTreeCallback* je zadužena za samo izvršavanje testova i prikupljanje njihovih rezultata. Potrebno je da interaktivni mod bude uključen u slučaju da neki od testova zahtevaju funkcionalnu interaktivnost. Test u potpunosti neće biti izvršen u slučaju da interaktivni mod nije uključen.

U slučaju da je došlo do greške prilikom izvršavanja određenog testnog slučaja i ukoliko je uključen mod za zaustavljanje na prvoj grešci, prekida se dalji prolazak kroz ostatak stabla i izvršavanje se prekida.

```
void ExecuteTreeCallback(tTestNode *aTestNode, int *bKeepWalking)
```

Parametri:

- *aTestNode* – Pokazivač na testni slučaj koji treba da se izvrši
- *bKeepWalking* – Indikator nastavka testiranja.

4.3 Pisanje testova

Pošto su testovi organizovani u obliku stabla, prilikom njihovog pisanja, prvo je potrebno definisati koren stabla. Da bi određen čvor stabla bio interpretiran kao koren, od strane aplikacije, potrebno je da bude nazvan *gTestRoot*.

```
tTestNode gTestRoot  = {
    "root",
    "the test root",
    0,
    NULL,
    gRootTestNodeList
};
```

Prilikom pisanja testova, dostupne su tri funkcije za proveru očekivanih rezultata, kao i njihovo prikupljanje. Takođe su dostupne i dve funkcije koje služe za merenje proteklog vremena.

```
TESTMANAGER_err TestManager_AssertTrue(int aAssert, char *aMessage)
```

Parametri:

- *aAssert* – Predstavlja vrednost nekog logičkog izraza čija se tačnost proverava
- *aMessage* – Poruka koja služi kao kratki opis proveravanog izraza

Povratna vrednost:

- *TESTMANAGER_NO_ERR* – Provera datog izraza je uspešna

```
TESTMANAGER_err TestManager_AssertEqual(long aLeftExpr, long aRightExpr, char *aMessage)
```

Parametri:

- *aLeftExpr* – Predstavlja levu stranu izraza čija se jednakost proverava
- *aRightExpr* – Predstavlja desnu stranu izraza čija se jednakost proverava
- *aMessage* – Poruka koja služi kao kratak opis proveravanog izraza

Povratna vrednost:

- *TESTMANAGER_NO_ERR* – Provera datog izraza je uspešna

```
TESTMANAGER_err TestManager_AssertGreater(double aLeftExpr, double aRightExpr, char *aMessage)
```

Parametri:

- *aLeftExpr* – Predstavlja levu stranu izraza, čija očekivana vrednost treba da bude veća od desne strane izraza
- *aRightExpr* – Predstavlja desnu stranu izraza, čija očekivana vrednost treba da bude manja od leve strane izraza
- *aMessage* – Poruka koja opisuje proveravani izraz

Povratna vrednost:

- *TESTMANAGER_NO_ERR* – Provera datog izraza je uspešna

```
typedef struct {
    TKEL_tck mStartTime;
    TKEL_tck mStopTime;
} tTestManagerChrono;

TESTMANAGER_err TestManager_StartChrono(tTestManagerChrono * aChrono)
```

Parametri:

- *aChrono* – Struktura u koju se upisuje trenutno vreme

Povratna vrednost:

- *TESTMANAGER_NO_ERR* – Pokretanje merenja vremena uspešno

```
TESTMANAGER_err TestManager_StopChrono(tTestManagerChrono * aChrono, long
*ms)
```

Parametri:

- *aChrono* – Struktura u koju se upisuje vreme zaustavljanja merenja.
- *ms* – Predstavlja ulazno-izlazni parametar koji sadrži proteklo vreme od trenutka pokretanja i zaustavljanja merenja

Povratna vrednost:

- *TESTMANAGER_NO_ERR* – Zaustavljanje merenja vremena uspešno

Funkcija *TestManager_AskForChecking* služi za postavljanje pitanja korisniku. Pitanje se postavlja ukoliko je potrebna interakcija sa korisnikom. Pitanje, na primer, može da glasi: “Da li je započela reprodukcija video signala” U okviru funkcije *TestManager_AskForChecking* poziva se funkcija *TestManager_AskForCheckingTimeout*. Funkcija *TestManager_AskForCheckingTimeout* ima ulogu da omogući nastavak izvršavanja testova

ukoliko je isključen interaktivni mod. U tom slučaju se, prilikom upita, izvršavanje nastavlja nakon pauze od jedne sekunde.

```
TESTMANAGER_err TestManager_AskForChecking(char *aQuestionMessage)
```

Parametri:

- *aQuestionMessage* – Poruka koja predstavlja postavljeno pitanje korisniku.

Povratna vrednost:

- *TESTMANAGER_NOT_DONE* – Postavljanje upita korisniku neuspešno
- *TESTMANAGER_NO_ERR* – Postavljanje upita korisniku uspešno

```
TESTMANAGER_err TestManager_AskForCheckingTimeout(char *aQuestionMessage
, unsigned int timeout_ms)
```

Parametri:

- *aQuestionMessage* – Poruka koja predstavlja postavljeno pitanje korisniku.
- *timeout_ms* – Parametar predstavlja vreme nakon kog će se izvršavanje nastaviti ukoliko se aplikacija trenutno ne nalazi u interaktivnom modu.

Povratna vrednost:

- *TESTMANAGER_NOT_DONE* – Postavljanje upita korisniku neuspešno
- *TESTMANAGER_NO_ERR* – Postavljanje upita korisniku uspešno

Neki testovi mogu od korisnika zahtevati funkcionalnu interaktivnost. Funkcija *TestManager_AskForAction* se koristi u takvim slučajevima interaktivnosti. Sama funkcija ne radi ništa osim ispisa prosleđene poruke. Uloga ove funkcije je da omogući proveru da li je testni slučaj pravilno definisan, odnosno da li sadrži pravilno definisan član *mbIsFunctionalyInteractive*.

```
TESTMANAGER_err TestManager_AskForAction(char *aActionMessage)
```

Paramtri:

- aActionMessage – Poruka koja opisuje zahtevanu akciju od korisnika.

Povratna vrednost:

- *TESTMANAGER_NO_ERR* – Zahtev akcije od korisnika uspešno izvršen
- *TESTMANAGER_NOT_DONE* – Zahtev nije izvršen zbog isključenog interaktivnog moda

4.4 Prikupljanje rezultata

Struktura *tTestManager* služi za praćenje i čuvanje rezultata testova. Rezultati se čuvaju u vidu niza *mTestResultVector*. Svaki rezultat je predstavljen novom strukturom *tTestResult*, koja sadrži podatke o uspešnosti odgovarajućeg testa. Rezultat se sastoji od broja uspešnih i neuspešnih iskaza, odnosno provera očekivanih rezultata. Prikupljanje rezultata se vrši pomoću prethodno navedenih takozvanih "assert" funkcija kao i funkcije *TestManager_AskQuestion*, koja se poziva prilikom interakcije sa korisnikom. Za prikupljanje rezultata služe funkcije *TestManager_AddOk* i *TestManager_AddFailure*. Funkcija *TestManager_AddOk* uvećava brojač *mTheOkTotal*. Dok, funkcija *TestManager_AddFailure* uvećava brojač *mTheFailureTotal*. Brojači *mTheOkTotal* i *mTheFailureTotal* se nalaze u okviru strukture *tTestResult*.

```
typedef struct {
    tTestNode *mTestNode;
    long mTheOkTotal;
    long mTheFailureTotal;
    long mNbExecution;
    long mNbSonFullyExecuted;
} tTestResult;

typedef struct {
    int mInteractiveMode;
    int mStopAtFirstErrorMode;
    tTestResult *mTestResultVector;
    tTestResult *mCurrentTestResult;
} tTestManager;
```

Nakon završenog izvršavanja testova potrebno je prikupiti sve rezultate za testne grupe. Prikupljanje se vrši pozivom funkcije *AggregateTestSuiteResult*. Funkcija *AggregateTestSuiteResult* prolazi kroz celo stablo, takođe pomoću funkcije *WalkInTree*. Prilikom poziva se, kao parametar, prosleđuju pokazivači na funkcije *ResetTestSuiteResultCallback* i *PostfixedAggregateTestSuiteResultCallback*.

```
void ResetTestSuiteResultCallback(tTestNode *aTestNode,
        int *bKeepWalking)
```

Parametri:

- *aTestNode* - predstavlja određeni čvor stabla, čiji se odgovarajući rezultat izvršavanja poništava.
- *bKeepWlaking* - predstavlja indikator pomoću kojeg je moguće zaustaviti prolazak kroz stablo

```
TESTMANAGER_err AggregateTestSuiteResult(tTestNode *aTestRoot)
```

Parametri:

- *aTestRoot* - predstavlja čvor stabla od kog se započinje prikupljanje rezultata

Povratna vrednost:

- *TESTMANAGER_NO_ERR* - prikupljanje svih rezultata je uspešno

Funkcija *ResetTestSuiteResultCallback* služi za inicijalizaciju rezultata testnih grupa, odnosno čvorova stabla koji imaju potomke. Funkcija *PostfixedAggregateTestSuiteResultCallback* služi za prikupljanje rezultata svih izvršenih testnih slučajeva koji pripadaju istoj grupi.

4.5 Upravljanje aplikacijom

Aplikacija za testiranje je konzolna aplikacija. Konzolna u smislu da se upravljanje vrši unošenjem komandi putem konzole, odnosno komandne linije.

Prilikom pokretanja aplikacije, pokreće se petlja za prihvatanje unetih komandi. Svaka uneta komanda, odnosno niz karaktera, se prosleđuje funkciji *TestManager_InterpretCommand*. Ova funkcija služi za interpretiranje komandi, odnosno proveru njihove ispravnosti. Na osnovu interpretacije komande se izvršava određena akcija.

```
int TestManager_InterpretCommand(const char *aInput)
```

Parametri:

- *aInput* – Parametar predstavlja unetu komandu u obliku niza karaktera

Povratna vrednost:

- 0 – Pokazatelj da je prosleđena komanda za izlaz iz aplikacije
- 1 – Nastavlja se dalje izvršavanje aplikacije

Neke od dostupnih komandi imaju dodatne parametre. Pomoću ovih parametara se postiže dodatna kontrola izvršavanja tih komandi. Komande za prikaz dostupnih testnih slučajeva i njihovih rezultata izvršavanja sadrže dodatni parametar. Dodatni parametar služi za odabir određenog skupa testnih slučajeva na koje se data komanda odnosi. Funkcija *SetOption* služi za postavljanje dodatnog parametra.

```
int SetOptions(const char *aSecondArg)
```

Parametri:

- *aSecondArg*- Predstavlja opciju određene komande i može biti vrednosti od 1 do 4

Povratna vrednost:

- 1 – Prosledjena je validna vrednost
- 0 – Proslednjena je pogrešna vrednost

Komandama kao što su list, result, reset i cd, čija je uloga objašnjena u prethodnom poglavlju, se može, kao dodatni parametar, proslediti broj ili ime određenog čvora stabla. Uloga ovog parametra je da omogući da se određena komanda odnosi samo na potomke datog čvora. Na primer, prilikom izvršavanja komande izlistavanja čvorova stabla, biće izlistani samo potomci zadatog čvora.

Funkcije za pronalaženje određenog člana u stablu su FindNodeFromNumber i FindNode. Funkcija FindNodeFromNumber pretražuje stablo na osnovu broja traženog člana, dok funkcija FindNode pretražuje stablo na osnovu imena traženog člana.

```
int FindNodeFromNumber(const char *aInput, tTestNode **apTestNode)
```

Parametri:

- aInput – Predstavlja broj traženog člana stabla u obliku niza karaktera
- apTestNode – Ulazno izlazni parametar koji predstavlja pokazivač na pronađeni član.

Povratna vrednost:

- 0 – Pronalaženje člana stabla je neuspešno
- 1 – Pronalaženje člana stabla je uspešno

```
int FindNode(const char *aCommand, tTestNode **aNode)
```

Parametri:

- aCommand – Predstavlja ime traženog člana stabla
- aNode – Ulazno – izlazni parametar koji predstavlja pokazivač na pronađeni član.

Povratna vrednost:

- 0 – Pronalaženje člana stabla je neuspešno
- 1 – Pronalaženje člana stabla je uspešno

5. Ispitivanje i verifikacija

U okviru ovo rada realizovana je aplikacija za testiranje, sa ciljem ispitivanja i verifikacije CHAL programskog sloja. Napravljen je niz testnih slučajeva kako bi se potvrdilo ispravno funkcionisanje pojedinačnih modula programske podrške televizijskih prijemnika.

Testni slučajevi su organizovani u grupe od kojih su najznačajnije:

- TestTKEL
- TestTDAL
- TestIP

5.1 Ispitivanje TKEL modula

U okviru TestTKEL grupe nalaze se testni slučajevi čija je uloga provera robusnost i konzistentnosti funkcionalnosti CHAL programske podrške vezane za operativni sistem. U ovu grupu spadaju testovi za proveru osnovne funkcionalnosti specifične za dati operativni sistem (sinhronizacije, kritične sekcije, semafori, međuprocesna komunikacija, rukovanje programskim nitima, itd.).

5.2 Ispitivanje TDAL modula

U okviru TestTDAL grupe, nalaze se testni slučajevi za proveru funkcionalnosti TDAL modula CHAL programske podrške. U okviru ove grupe je ispitivana funkcionalnost vezana za programsku podršku za televizijske prijemnike. Ispitivani su svi blokovi obuhvaćeni TDAL modulom (slika, zvuk, demultiplekser, demodulator, itd.)

5.3 Ispitivanje IP modula

U okviru TestIP grupe izdvojeni su testni slučajevi specifični za verifikaciju programske podrške koja se odnosi na prikaz televizijskog sadržaja. U okviru ove grupe ispitivana je

funkcionalnost promene kanala, reprodukcije video i audio zapisa, kao i trik modova (premotavanje, pauziranje, itd.) uz upotrebu takozvanih "stress" testova. Njihov cilj je da, velikim brojem ponovljenih simuliranih akcija korisnika (npr. promena kanala, pauziraj-nastavi), dovedu sistem u nekonzistentno stanje kako bi se uočila neka greška ili curenje memorije, koji se inače teško uočavaju nakon svega nekoliko takvih akcija.

6. Zaključak

U ovom radu je opisana realizacija alata za izvršavanje testnih slučajeva sa ciljem testiranja i verifikacije ciljne programkse podrške. Svrha ovog alata je prvenstveno testiranje i verifikacija rada CHAL programske podrške televizijskih prijemnika. Alat za testiranje predstavlja konzolnu aplikaciju realizovanu u C pogramskom jeziku. Opisano rešenje je realizovano na ciljnoj platformi sa Android operativnim sistemom i Comedia programskom podrškom za televizijske prijemnike.

Realizovana aplikacija predstavlja konzolnu aplikaciju. Upravljanje aplikacijom se isključivo vrši putem konzole. Jedan od aspekata daljeg razvoja bi mogao biti realizacija grafikče korisničke sprege. Na ovaj način bi se korisnicima olakšalo korišćenje aplikacije. Rezultati testova bi, takođe, bili znatno pregledniji ako bi se prikazali grafičkim putem.

Realizovano rešenje sadrži tri “assert” funkcije. Funkcije za proveru određenih logičkih izraza, kao što su: veće, manje i jednako. Dalji razvoj bi mogao da podrazumeva implementaciju dodatnih funkcija. Primer toga bi bile funkcije za poređenje složenijih tipova podataka (npr. nizovi, liste, stabla, itd.).

7. Literatura

[1] Vladimir Kovačević, Miroslav Popović: Sistemska programska podrška u realnom vremenu, Univerzitet u Novom Sadu, Fakultet Tehničkih Nauka, 2002.

[2] Sajt android podrške za razvij, www.developer.android.com

[3] Reto Meier, Professional Android 2 Application Development

[4] Iwedia, TDAL_DMX Technical Specifications, 2009.

[5] Iwedia, TDAL_DMD Technical Specifications, 2009.

[6] Iwedia, TDAL_GFX Technical Specifications, 2009.

[7] Iwedia, TDAL_DISP Technical Specifications, 2009.

[8] A. Bertolino, "Chapter 5: Software Testing," in IEEE SWEBOK Trial Version 1.00, May 2001.

[9] Laurie Williams: White-Box Testing, 2006

[10] Laurie Williams: Testing Overview and Black-Box Testing Techniques, 2006