УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ДИПЛОМСКИ – МАСТЕР РАД

Кандидат: **Бојан Мајсторовић**
Број индекса: **E11044**

Тема рада: **Једно решење програмског окружења за ефикасан пренос програмске подршке наменских уређаја са графичком спрегом на iPhone**

Ментор рада: **Доц. Др Иштван Пап**

Нови Сад, април, 2011

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

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Дипломски – мастер рад	
Аутор, **АУ**:		Бојан Мајсторовић	
Ментор, **МН**:		др Иштван Пап, доцент	
Наслов рада, **НР**:		Једно решење програмског окружења за ефикасан пренос програмске подршке наменских уређаја са графичком спрегом на iPhone	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2011	
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)		7/46/0/4/18/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кључне речи, **ПО**:		iPhone, пренос програмске подршке, наменски уређај, контрола светла	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом раду описује се приступ развоју програмског окружења за наменске уређаје који обезбеђује њихов једноставан пренос на iPhone платформу. Изложени приступ обезбеђује верну емулацију графичког изгледа програма и функционалности. Окружење за емулацију се конфигурише одговарајућом XML датотеком. Као пример у раду је емулиран наменски уређај за контролу светла на iPhone платформи.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	**Проф. др Миодраг Темеринац**	
	Члан:	**др Небојша Пјевалица, доцент**	Потпис ментора
	Члан, ментор:	**др Иштван Пап, доцент**	

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

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Master Thesis	
Author, **AU**:		Bojan Majstorović	
Mentor, **MN**:		Istvan Pap, PhD	
Title, **TI**:		A programming framework for efficient embedded device software and graphical user interface porting to iPhone	
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**:		2011	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/46/0/4/18/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		iPhone, software transfer, embedded device, light control	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This paper describes the approach to the development of software for embedded devices which provides their easy porting to iPhone platform. The presented approach provides a complete emulation of graphic design and functionality. Emulation environment is configured in the appropriate XML file. As an example in this paper is presented emulated embedded device for controling the lights on iPhone platform.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**Miodrag Temerinac, PhD**	
	Member:	**Nebojsa Pjevalica, PhD**	Menthor's sign
	Member, Mentor:	**Istvan Papp, PhD**	

Zahvalnost

Zahvaljujem se porodici na svesrdnoj pomoći i razumevanju.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

SDL	- Simple DirectMedia Layer, multimedijalna biblioteka
SDK	- Software Development Kit, razvojno okruženje za izradu aplikacija
API	- Application Programming Interface, aplikativna programska sprega
IDE	- Integrated Development Environment, integrisano razvojno okruženje
XML	-eXtensible Markup Language, proširivi metajezik za označavanje tekstualnih dokumenata
OpenGL	- Open Graphics Library, višeplatformska grafička biblioteka
GUI	- Graphical User Interface, grafičko radno okruženje

1. Uvod

Tehnologija se iz dana u dan razvija i savremeno društvo postaje zavisno od tehnologije. Razvojem tehnologije pojavljuju se novi uređaji čiji je cilj da olakšaju život savremenom čoveku. Neka se uzme za primer kuća, u njoj se nalaze uređaji kao što su: televizor, DVD plejer, muzička linija, klima uređaj, računar. Svaki od tih uređaja poseduje daljinski upravljač, a to je ukupno pet različitih daljinskih upravljača. Razvojem tehnologije nastaju univerzalni daljinski upravljači tako da se svi ti daljinski upravljači sadrže u jednom uređaju. Sada se može upravljati sa svim uređajima u kući pomoću jednog upravljača. Međutim tu nije kraj, razvojem koncepta pametne kuće proširuje se skup uređaja koji se mogu daljinski kontrolisati. Samim tim se i univerzalni daljinski upravljači poboljšavaju, u smislu da sada sadrže i mali ekran na kome se korisniku prikazuju trenutne informacije o kontrolisanom uređaju. Noviji uređaji sadrže ekrane osetljive na dodir. Poenta pametne kuće je da se može u svakom momentu videti stanje svakog uređaja u kući i da se njihovo stanje može menjati u skladu sa zahtevima. Problem se javlja kada se korisnik nalazi van kuće, on tada neće moći da vidi i da kontroliše nijedan od uređaja u kući. Predlog rešenja ovog problema ja da se univerzalni upravljač implementira na iPhone uređaju. Tako bi bilo moguće upravljati uređajima u kući čak i kada je korisnik van kuće. Postavlja se pitanje kako najlakše izvršiti prenos postojeće programske podrške namenskog uređaja na iPhone uređaj. Ovaj problem je moguće efikasno rešiti korišćenjem koncepta opisanog u ovom radu.

1.1 Osnovni zadaci rada

Zadatak rada je da se predloži i realizuje koncept programskog okruženja na iPhone platformi koje obezbeđuje jednostavan prenos programske podrške namenskog uređaja na iPhone platformu, uz vernu emulaciju grafičkog izgleda programa i funkcionalnosti. Programska podrška na iPhone platformi treba da obezbedi odgovarajući nivo apstrakcije fizičke arhitekture namenskog uređaja. Parametri emulacije fizičke arhitekture namenskog uređaja zadaju se u ulaznoj XML datoteci. Omogućiti emulaciju fizičkih tastera namenskog uređaja na ekranu iPhone uređaja.

Kao potvrda predloženog koncepta potrebno je razviti programsku podršku namenskog uređaja za kontrolu svetla, i preneti je na iPhone platformu uz upotrebu realizovanog programskog okruženja. Pri tome obezbediti verodostojnu emulaciju grafičkog izgleda i funkcionalnosti.

2. Razvojno okruženje

Aplikacija je namenjena da se izvršava na iPhone OS operativnom sistemu. iPhone je izabran kao veoma popularna platforma, široko rasprostranjena, a takođe sa mogućnošću plasiranja date aplikacije širokom krugu korisnika preko AppStore internet prodavnice. Aplikacija je razvijana je na MAC OS X operativnom sistemu u aplikativnoj programskoj sprezi Cocoa sa ugrađenom podrškom za Objective-C programski jezik i programskim alatima: Xcode i Interface Builder. Pored pomenutih alata, za pokretanje aplikacije je korišten iPhone simulator verzije 4.0, a kasnije i iPhone 3GS uređaj verzije 4.0

Za obezbeđivanje prenosivosti programske podrške potrebno je odabrati odgovarajuću programsku biblioteku koja obezbeđuje neophodne funkcije. Analizirane su biblioteke kao što su SDL, SFML, PLIB, OpenGL, FLTK, ClanLib, ALLEGRO, GTK+, Qt. Međutim većina tih biblioteka nije podržana na operativnom sistemu iPhone OS.

Biblioteke	Podržani operativni sistemi							
	Windows	Linux	MAC OS X	UNIX	BeOS	Symbian	X11	iPhone OS
SFML	+	+	+	-	-	-	-	-
PLIB	+	-	+	+	-	-	-	-
OpenGL	+	-	-	-	-	-	-	-
FLTK	+	+	+	+	-	-	-	-
ClanLib	-	+	+	+	-	-	-	-
ALLEGRO	+	-	+	+	+	-	-	-
GTK+	+	-	+	-	-	-	+	-
Qt	+	+	+	-	-	+	+	+
SDL	+	+	+	-	+	-	+	+

Tabela 2.1 Pregled multimedijalnih biblioteka

SDL biblioteka od verzije 1.3 sadrži licencu za korišćenje u komercijalnim aplikacijama. SDL biblioteku je moguće uvezati u projekat statički i dinamički. Besplatna verzija dozvoljava samo dinamičko uvezivanje, dok je verziju sa statičkim uvezivanjem potrebno kupiti. Tip uvezivanja je bitan iz prostog razloga što AppStore prihvata samo one iPhone aplikacije koje koriste statički uvezanu SDL biblioteku. SDL biblioteka se usvojila kao osnova za realizaciju grafičke korisničke sprege.

2.1 Cocoa API

Cocoa je jedna od pet osnovnih aplikativnih programskih sprega (API), ostale četiri su: Carbon, POSIX, X11, Java. Cocoa je ugrađena aplikativna programska sprega za izradu aplikacija na MAC OS X operativnom sistemu. Uz korištenje Cocoa Touch dodatka za prepoznavanje gestova i realizaciju animacije moguća je izrada aplikacija i na iOS operativnom sistemu koji se koristi na mobilnim uređajima kao što su iPhone, iPod i iPad. Cocoa aplikacije se razvijaju pomoću Xcode i Interface Builder alata, i razvijaju se u programskom jeziku Objective-C. Pored nabrojanih alata Cocoa aplikaciju je moguće pisati u jednostavnom uređivaču teksta i napraviti izvršnu verziju programa pomoću GCC prevodioca iz komandne linije. Jedna od osobina Cocoa sprege jeste način na koji se upravlja dinamički dodeljenom memorijom. NSObject klasa u Cocoa sprezi, iz koje je većina klasa izvedena, upravlja dinamički dodeljenom memorijom računanjem broja alokacija objekta. Objekti koji su izvedeni iz NSObject klase čuvaju informaciju o broju alokacija. Alociranjem memorije za objekat pomoću komandi alloc ili copy postavlja broj alokacija na jedan. Broj alokacija se uvećava porukom retain, a smanjuje se porukom release. Poruka retainCount se koristi za određivanje broja trenutnih alokacija objekta. Kada broj alokacija objekta dođe do nule objekat se dealocira slično kao destruktor u C++ jeziku. Ovakav pristup računanju alokacija je veoma sličan Microsoft Component Object Model (COM) principu. Poruke retain i release u Cocoa sprezi su ekvivalentne metodama AddRef i Release u Microsoft COM okruženju.

2.2 Objective C

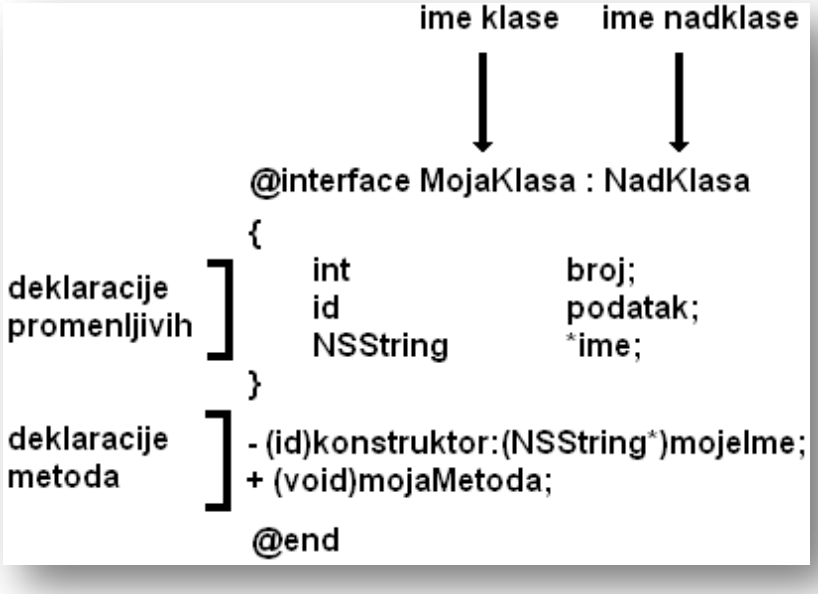
Objective-C je programski jezik dizajniran da omogući objektno orijentisano programiranje. Objective-C programski jezik je dizajniran u ranim osamdesetim. Jezik je zasnovan na strukturi tada već poznatog programskog jezika Smalltalk. 1988. godine kompanija NeXT Software licencira Objective-C sa osnovnim skupom biblioteka, i razvija razvojno okruženje NEXTSTEP na kom Apple temelji buduća izdanja operativnog sistema OS X.

Objective-C predstavlja proširenje ANSI C programskog jezika. Iz tog razloga zadržava osnovnu C sintaksu. Objective-C podržava većinu standardnih objektno-orijentisanih koncepata, sa određenim razlikama, kao što su enkapsulacija, nasleđivanje i polimorfizam. Što se tiče operacija koje nisu objektno orijentisane (promenljive, preprocesiranje, izrazi, deklaracije funkcija, pozivi funkcija) su identične onima koje postoje u programskom jeziku C, dok su objektno orijentisane operacije implementirane slanjem poruka u jeziku Smalltalk.

Klasa u Objective-C programskom jeziku može imati dve vrste metoda:

- Metode primerka
- Metode klase

Metoda primerka predstavlja metodu koju izvršava objekat, odnosno primerak klase. Drugim rečima, pre pozivanja metode primerka mora se prvo napraviti primerak klase. Za razliku od metoda primerka, za pozivanje metoda klase nije potrebno stvoriti primerak klase, već data klasa poziva metodu klase. Metode klase jednoznačno odgovaraju statičkim metodama u C++ jeziku. Na narednoj slici su prikazane dve metode. Minusom (-) se u Objective-C programskom jeziku označava metoda primerka, a plusom (+) metoda klase. Zatim sledi tip promenljive koju metoda vraća. Na slici su to metode koje vraćaju vrednost (id) u jednom slučaju, a u drugom slučaju metoda ne vraća vrednost (void). Zatim slede ključne reči metode, tipovi i imena parametara ako postoje.



Slika 2.1 Struktura klase

2.2.1 Poruke

Objective-C jezik se zasniva i na jeziku Smalltalk, koji u sebi sadrži sistem slanja poruka ka objektima. Zbog toga se Objective-C model za objektno programiranje zasniva na prenosu poruke u objekat, što znači da se ne poziva metoda koja treba da se izvrši, već se vrši slanje poruke koja u svom nazivu sadrži metodu koja treba da izvrši pozvani objekat. Razlika između pozivanja metode i slanja poruke je u tome kako će se dati kod izvršiti. Kod pozivanja metode u C++ se u prevodiocu koda odredi kojoj klasi pripada tj. vezana je za tu klasu pre izvršavanja. U Objective-C jeziku kada se šalju poruke primalac poruke se određuje u toku izvršavanja.

Pozivanje metode u C++ jeziku:

```
Object.method(parameter)
```

Slanje poruke u Objective-C jeziku:

```
[Object method:parameter]
```

2.2.2 Definicija i implementacija klase

Objective-C jezik definiše klasu u datoteci sa proširenjem h (`classname.h`). Definicija klase počinje direktivom `@interface`, a završava se direktivom `@end`.

```
@interface classname : superclassname {
// instance variables
}
+ classMethod1;
+ (return_type)classMethod2;
+ (return_type)classMethod3:(param1_type)parameter_varName;
- (return_type)instanceMethod1:(param1_type) param1:(param2_type)param2;
@end
```

Za uključivanje dodatnih modula ili biblioteka koristi se direktiva `@import`, koja zamenjuje preprocesorsku direktivu `#include` koja se koristi u programskom jeziku C. Klasa je implementirana u datoteci sa proširenjem m (`classname.m`). Implementacija klase počinje direktivom `@implementation`, a završava direktivom `@end`.

```
@import "classname.h"
@implementation classname
+ classMethod {
// implementation
}
```

```
- instanceMethod {
// implementation
}
@end
```

2.2.3 Protokoli

Protokoli deklarišu metode koje mogu biti implementirane od bilo koje klase. Protokol sam po sebi nije klasa. Protokol samo definiše okruženje koje drugi objekti treba da implementiraju. Kada se implementira metoda nekog protokola u klasi tada se kaže da ta klasa podržava dati protokol. Protokol počinje deklaracijom `@protocol`, a završava direktivom `@end`. Metode protokola mogu biti opcione ili obavezne i deklarišu se direktivama `@optional` i `@required`. Protokol može biti formalni i neformalni. Objective-C jezik koristi protokole koji su definisani od strane programera i oni spadaju u neformalne protokole. Formalni protokoli su protokoli nametnuti od strane prevodioca.

Formalni protokol:

```
@protocol MyProtocol
- (void)requiredMethod;
@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;
@required
- (void)anotherRequiredMethod;
@end
```

Neformalni protokol:

```
@interface NSObject ( MyProtocol )
- (void)requiredMethod;
- (void)anotherRequiredMethod;
@end
```

2.2.4 Svojstva

Svojstva (engl. properties) predstavljaju drugačiji način deklarisanja članova klase u jeziku Objective-C, tj. mogu da zamene deklaraciju članova klase, a takođe mogu da zamene implementaciju pristupnih metoda. Pristupne metode su poznate pod nazivom (get i set). Svojstva se definišu sa direktivom `@property` pored toga sadrže atribute date promenljive, njen tip i njeno ime.

```
@property(attributes) type name;
```

Pristupne metode omogućavaju pristup privatnim članovima klase. Atributi mogu biti:

- `readwrite`
- `readonly`

Atribut `readwrite` daje do znanja prevodiocu da je omogućen pristup privatnim članovima klasa za upis i za čitanje, što znači da ce prevodilac da napravi get i set metode. U drugom slučaju kada je atribut `readonly` tada je omogućeno samo čitanje privatnih članova klase i prevodilac pravi samo get metodu.

Dodatni atributi koji se koriste označavaju način na koji se dodeljuju vrednosti privatnim članovima klase. Atributi su međusobno isključivi.

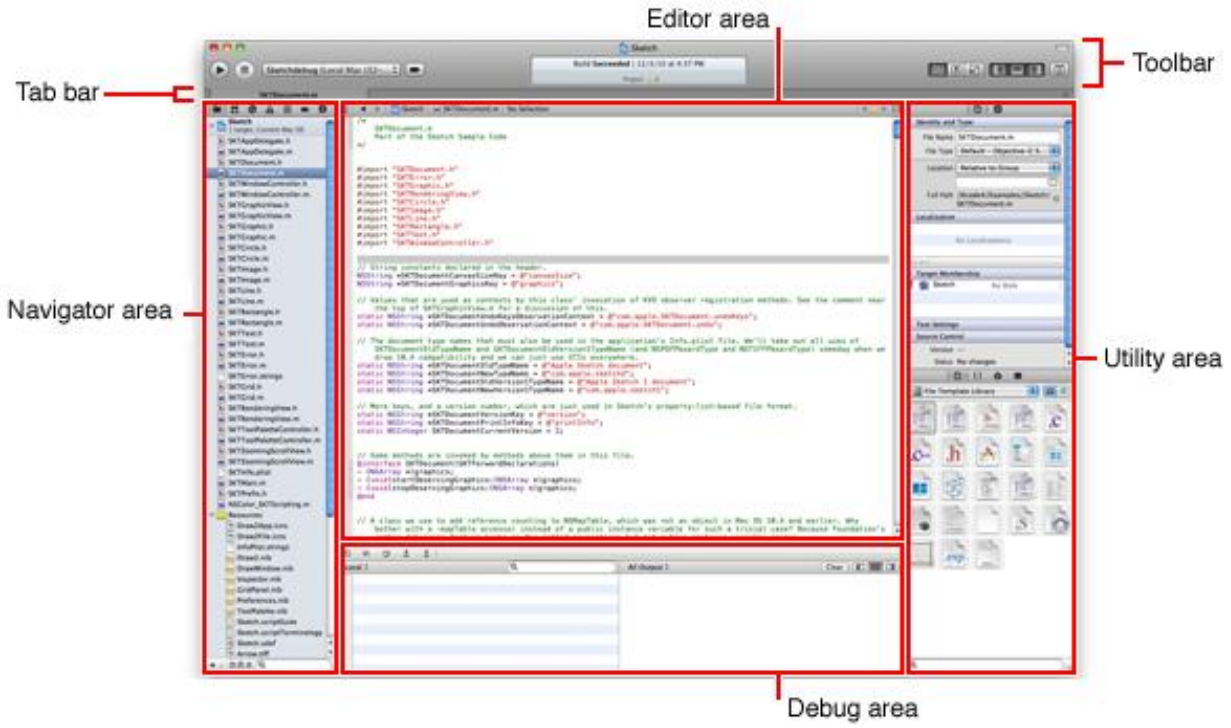
- `assign`
- `copy`
- `retain`

Atribut `assign` predstavlja jednostavnu dodelu vrednosti. Atribut `copy` dodeljuje kopiju originalne vrednosti. Atribut `retain` uvećava broj alokacija objekta, jer bi bez toga objekat postao nevalidan.

Svojstva redukuju deo nepotrebnog koda koji programer treba da napiše, eliminišući potrebu za implementacijom get i set metoda. To se postiže korišćenjem `@synthesize` direktive koja podrazumeva da će prevodilac da napravi pristupne metode, a takođe postoji direktiva `@dynamic` kojom prevodilac prepušta programeru da napravi pristupne metode.

2.3 Integrisano razvojno okruženje - Xcode

Xcode je paket alata za razvoj softvera na MAC OS X operativnom sistemu. Dolazi besplatno u paketu sa MAC OS X verzijom 10.6, u verziji 3.2 za iPad uređaje i verziji 4.0 za iPhone i iPod uređaje. Paket sadrži modifikovane verzije besplatnog GCC (GNU Compiler Collection) prevodioca (verzije gcc 4.2.1 i gcc 4.0.1). Podržava jezike: C, C++, Objective-C, Objective-C++, Java, AppleScript i Python. Treća lica su dodala podršku za GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell i D jezike. Glavna primena ovog paketa je integrisano razvojno okruženje (IDE), takođe pod nazivom Xcode.



Slika 2.2 Razvojno okruženje Xcode

Xcode okruženje se koristi za prikaz i razvoj aplikacija na MAC OS X operativnom sistemu, a takođe i na prenosnim uređajima (iPhone, iPod, iPad). Xcode projekat je prikazan u prozoru koji predstavlja radni prostor. Na slici 2.2 je prikazana standardna konfiguracija prozora radnog prostora, sa šest glavnih komponenti.

- Traka sa alatkama (engl. Toolbar)
- Traka sa karticama (engl. Tab bar)
- Uređivač teksta (engl. Editor area)
- Prostor za navigaciju (engl. Navigation area)
- Prostor za debagovanje (engl. Debug area)
- Pomoćni prostor (engl. Utility area)

Traka sa alatkama obuhvata upravljanje i puštanje u rad određenih zadataka, pregled napretka u izvršavanju pokrenutog zadatka i alate za konfigurisanje prikaza radnog okruženja. Traka sa karticama je opciona ako je samo jedna kartica otvorena. Karticama može da se preuređuje redosled, mogu da se pojedinačno zatvaraju, a takođe je moguće izvući karticu iz trake i time prikazati karticu u novom prozoru. Glavni sadržaj se prikazuje ispod traka sa alatkama i uvek uključuje uređivač teksta. Kada se otvori neka datoteka u radnom prostoru njen sadržaj se prikazuje u uređivaču teksta. Glavni sadržaj sadrži i tri opcione oblasti. Prostor za

navigaciju koji obezbeđuje alatke za prikazivanje i manevrisanje kroz različite aspekte pokrenutog projekta. Prostor za debagovanje pruža kontrole za izvršavanje programa i otklanjanje grešaka. Prikazuje takođe prozor za prikaz promenljivih, prikaz registara i statusne informacije. Pomoćni prostor se koristi za prikaz dodatnih informacija o odabranom elementu. Prikazuju se detalji o elementu kao što su kratak opis elementa, gde i kako je deklarisan, životni vek ukoliko je u pitanju promenljiva, prikaz parametara ukoliko ih koristi. Ima mogućnost prikaza i upravljanja meta podacima aktivnih datoteka, kao što su ime datoteke, tip datoteke, putanja datoteke, lokacija u okviru projekta.

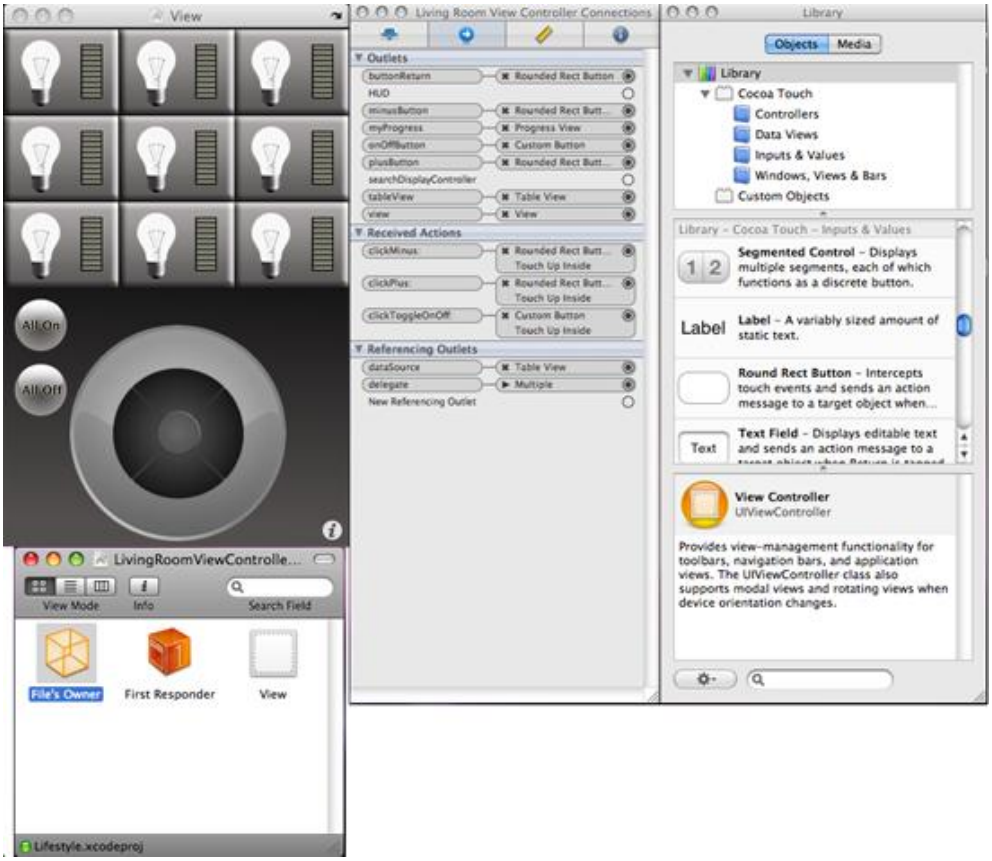
Xcode paket takođe sadrži veći deo dokumentacije koja pomaže u razvoju aplikacija, sadrži Interface Builder, aplikaciju za izradu grafičkog okruženja, a takođe sadrži i alat za testiranje pod nazivom Instruments.

2.3.1 Programska podrška za razvoj grafičkog okruženja (Interface Builder)

Interface Builder je sastavni deo Xcode paketa i koristi se za izradu grafičkog okruženja aplikacije. Korišćenjem Buildera, može se sastaviti prozor aplikacije prevlačenjem i otpuštanjem unapred definisane komponente na njega. Programsko okruženje obezbeđuje palete, odnosno kolekciju objekata korisničkog okruženja. Komponente su standardne kontrole sistema kao što su: polja za unos teksta, labele, tabele sa podacima, klizači, padajući meni, dugmići itd, kao i dodatni prikazi koji predstavljaju dodatne informacije o funkcijama koje aplikacija pruža. Palete su potpuno proširive što znači da svaki programer može da razvije nove objekte i palete. Nakon što su komponente postavljene na površinu prozora , mogu se pozicionirati tako što se prevlače na željenu lokaciju u prikazu, mogu se podesiti njihovi atributi koristeći inspektor prozor, u kome se prikazuju dodatne informacije o objektu, kao i uspostavljanje odnosa izmedu tih objekata i koda. Formirano grafičko okruženje se čuva u NIB datoteci sa proširenjem *xib*. NIB datoteke koje su kreirane u Interface Builderu sadrže sve informacije koje su potrebne da se rekonstruišu isti objekti u toku izvršavanja aplikacije. Učitavanje NIB datoteke u toku izvršavanja programa pravi izvršne verzije svih objekata koji se nalaze u datoteci, i konfiguriše ih tačno na način na koji su konfigurisani u Builderu. Takođe se koriste informacije koje su navedene u Builderu da se uspostavi veza između novonastalog objekata u aplikaciji i postojećeg objekta u datoteci.

Koristeći Builder štedi se ogromna količina vremena kada je u pitanju kreiranje grafičkog korisničkog okruženja aplikacije. Builder eliminiše pisanje koda neophodnog za stvaranje, konfigurisanje, i položaj objekata koji čine grafičko korisničko okruženje aplikacije. Zato što je

grafički pregledač, Builder omogućava prikaz kako će grafičko korisničko okruženje izgledati u toku izvršavanja.



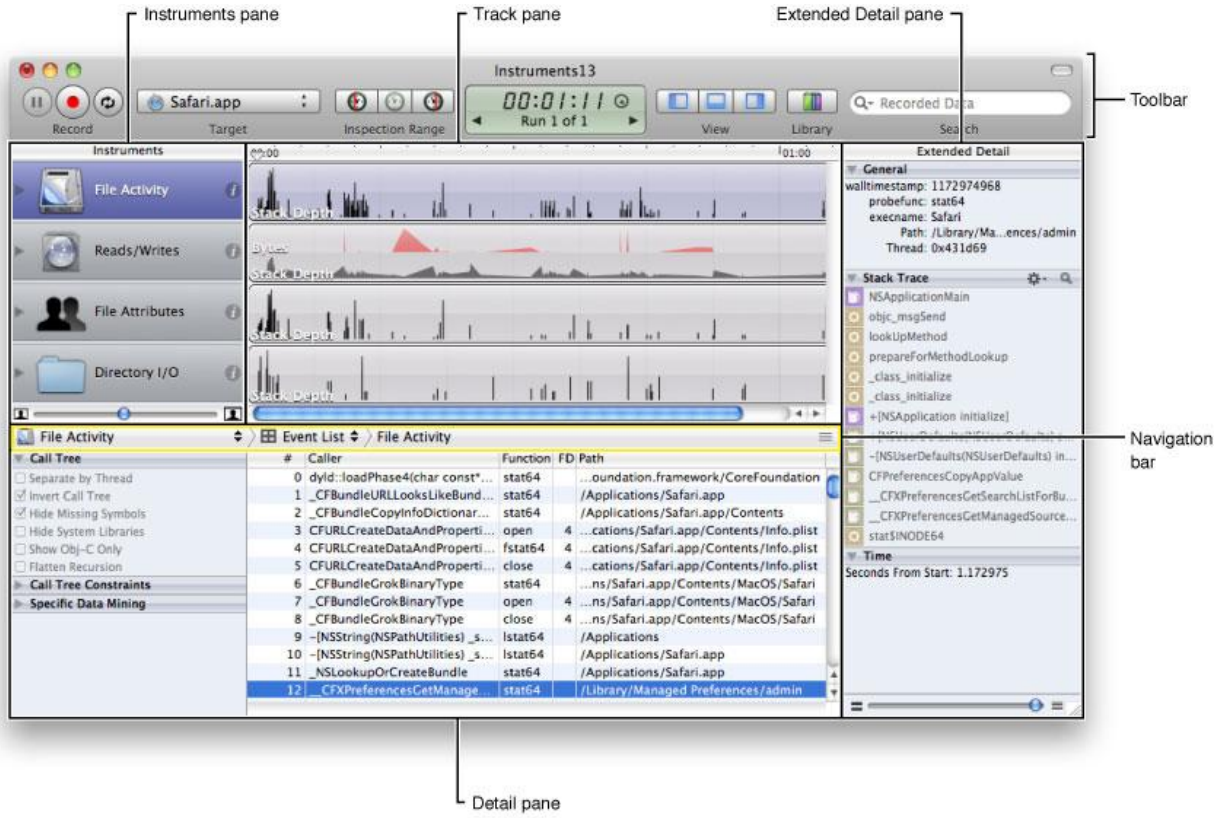
Slika 2.3 Interface Builder okruženje

Na slici 2.3 je prikazan izgled programske podrške za razvoj grafičkog okruženja. Prikazan je prozor koji predstavlja biblioteku svih grafičkih objekata koji mogu da se ubace u aplikaciju. Sledeći prozor koji je prikazan jeste prozor na kom se prikazuju svi ubačeni objekti, taj prozor ujedno predstavlja i izgled aplikacije koja se izvršavana iPhone uređaju. Pored ova dva prozora prikazan je i prozor u kome se svi ubačeni grafički objekti uvezuju za napisanim kodom.

2.3.2 Alat za testiranje (Instruments)

Instruments je moćan alat koji može da se iskoristi za prikupljanje podataka o performansama i ponašanju jednog ili više procesa u sistemu i praćenje podataka tokom vremena. Za razliku od većine drugih alata za prikupljanje podataka o performansama i alata za otklanjanje grešaka, Instruments alat omogućava okupljanje i uporedni prikaz više različitih tipova podataka. Na ovaj način je moguće lakše uočiti tendenciju rada programa koja bi bila teško uočljiva u drugim alatima. Omogućeno je analiziranje programa i analiziranje ponašanja memorije u jednom

pokretanju programa, a prikaz je realizovan grafički u obliku vremenske linije. Cilj rada jeste prikupljanje podataka o procesu tokom izvršavanja. Postoji više različitih vrsta informacija koje se prate i ispisuju kao što su: podaci o zauzeću memorije, curenju memorije, aktivnosti procesora, mrežne aktivnosti i grafičkih performansi. Instruments alat je moguće nadograditi ubacivanjem dodatnih instrumenata za ispitivanje stanja procesa. Ovaj alat omogućava analiziranje performansi aplikacija bez obzira da li se one izvršavaju na simulatoru ili direktno na uređaju.



Slika 2.4 Okruženje za testiranje Instruments

Na slici 2.5 je prikazan izgled okruženja za testiranje aplikacija.

- Prozor sa instrumentima (engl. Instruments pane)
- Prozor za prikaz vremenske linije (engl. Track pane)
- Prozor sa dodatnim informacijama (engl. Detail pane)
- Prozor sa proširenim dodatnim informacijama (engl. Extended Detail pane)
- Navigaciona traka (engl. Navigation bar)

Prozor sa instrumentima sadrži instrumente koji će se koristiti pri testiranju. Moguće je prevlačenje instrumenata u ovaj prozor, kao i brisanje instrumenata iz prozora. Svaki instrument sadrži dugme preko kog je moguće pristupiti opcijama instrumenta i izmeniti način prikaza

podataka i prikupljanje parametara. Prozor za prikaz vremenske linije prikazuje ponašanje sistema u obliku vremenske linije za svaki instrument posebno. Informacije u ovom prozoru se koriste samo za čitanje. Prozor sa dodatnim informacijama prikazuje detaljnije podatke dobijene od instrumenta. Prikazuju se događaji koji se koriste za stvaranje vremenske linije u prozoru za prikaz vremenske linije. Moguće je menjanje stepena detaljnosti prikaza podataka. Prozor sa proširenim dodatnim informacijama prikazuje još detaljnije informacije o stavci koja je izabrana u prozoru sa dodatnim opcijama. U ovom prozoru se najčešće prikazuje izgled stek memorije, vremenska oznaka i druge dodatne informacije specifične za dati instrument. Navigaciona traka prikazuje trenutno prikazani nivo detalja i kako se do tog nivoa stiglo. U traci se prikazuju dva menija, meni koji prikazuje aktivni instrument i meni za prikaz detalja. Omogućena je izmena aktivnog instrumenta u meniju, kao i nivo i prikaz tipa informacije u prikazu detalja u prozoru sa dodatnim informacijama.

2.4 Model-Prikaz-Upravljač arhitektura

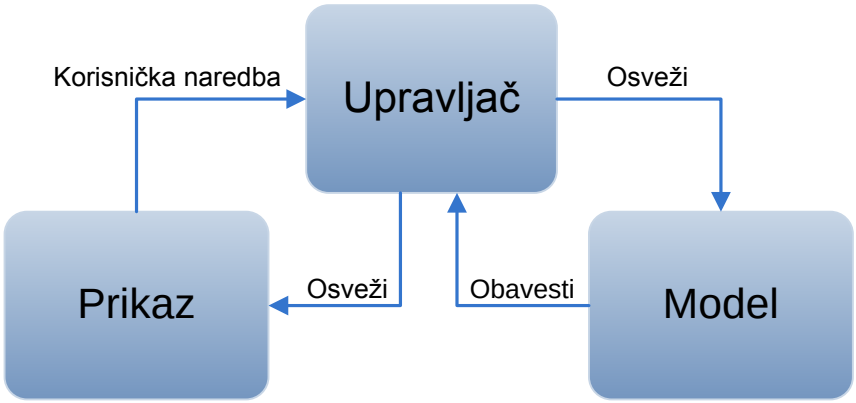
Model-Prikaz-Upravljač(MPU) je arhitektura koja je preuzeta iz Smalltalk jezika, i u ovom radu se koristi za formiranje grafičkog korisničkog okruženja. MPU se sastoji iz tri vrste objekata koji komuniciraju međusobno. Pre MPU-a dizajn korisničke sprege je formiran zajedno u jednom objektu. Korišćenjem MPU-a tj. razdvajanjem objekata povećava se fleksibilnost i lakše je ponovno korišćenje.

Model – čuva podatke i definiše način pristupa podacima. Korisnik ne može da pristupi direktno podacima u modelu. Model ima mogućnost višestruke upotrebe, distribuiran je, moguće ga je prenositi na različite platforme.

Prikaz – predstavlja prikaz podataka u korisničkom grafičkom okruženju. Prikaz nema uvid o tome gde se nalaze podaci koje prikazuje.

Upravljač – definiše način na koji korisnička sprega reaguje na korisničke zahteve. Upravljač posreduje u interakciji između Modela i Prikaza.

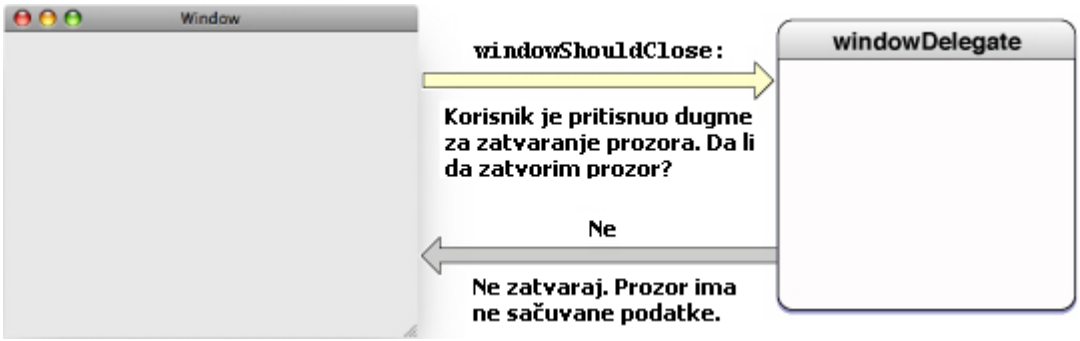
Ako su objekti Prikaz i Upravljač kombinovani kao rezultat se dobija Model/Prikaz arhitektura. Ovaj princip razdvaja način na koji se podaci čuvaju i način na koji se predstavljaju korisniku, ali omogućuje jednostavnije okruženje bazirano na istim principima kao MPU. Ovo razdvajanje omogućuje prikaz istih podataka u različitim Prikazima i implementaciju novih tipova prikaza bez potrebe za izmenom podataka koji se prikazuju.



Slika 2.5 Model-Prikaz-Upravljač arhitektura

Da bi se obezbedilo fleksibilnije rukovanje korisničkim zahtevima uvodi se koncept delegata. Delegacija predstavlja jednostavan princip prosleđivanja zadatka na obradu u nekom drugom delu programa. Delegacija je jednostavan obrazac u kojem jedan objekat u programu deluje u ime nekog drugog objekta ili u koordinaciji sa još jednim objektom. Delegirajući objekat čuva referencu na drugi objekat (delegat) i u odgovarajuće vreme šalje mu poruku. Poruka obaveštava delegat o događaju koji je delegirajući objekat izvršio ili tek treba da ga izvrši. Delegat može da odgovori na poruku ažuriranjem svog izgleda ili stanja, a takođe može da menja stanja drugih objekata u programu. U nekim slučajevima može da vrati vrednost koja utiče na rukovanje predstojećim događajima u programu. Osnovna vrednost delegacije je u tome što omogućava da prilagodi ponašanje više objekata u jednom centralnom objektu. Prednost korišćenja delegata u ovom okruženju je to što omogućuje promenu načina prikaza podataka i njihove izmene.

Sledeći primer pokazuje povezanost delegata sa delegirajućim objektom. Recimo da korisnik hoće da pritisne dugme zatvori prozor na MAC OS X operativnom sistemu. Delegirajući objekat šalje poruku `windowShouldClose` svom delegatu. Ta poruka daje delegatu priliku da stavi veto na zatvaranje prozora ili odloži zatvaranje ako je, na primer, prozor povezan sa podacima koji moraju biti sačuvani.



Slika 2.6 Primer delegata

2.5 SDL biblioteka

SDL biblioteka je višeplatformska multimedijalna biblioteka. Koristi se za dizajniranje: igara, razvojnog okruženja za izradu igara, emulatora, demoa i multimedijalnih aplikacija na različitim platformama. Dizajnirana je da pruži pristup niskog nivoa sledećim celinama:

- Zvučnom podsistemu
- Tastaturi
- Mišu
- Džojstiku
- Trodimenzionalnoj grafičkoj fizičkoj arhitekturi pomoću OpenGL biblioteke
- Dvodimenzionalnoj prihvatnoj memoriji slike (engl. frame buffer).

Pristup zvučnom podsistemu predstavlja mogućnost SDL biblioteke da reprodukuje zvuk zapisan u 8-bitnom ili 16-bitnom formatu, i koji može biti mono ili stereo. Moguća je konverzija zvučnog zapisa ukoliko format nije podržan na datoj fizičkoj arhitekturi. Podsistem za reprodukciju zvuka se izvršava u posebnoj niti. Za pristup tastaturi, mišu i džojstiku koristi se sistem događaja. Svaki od ovih uređaja stvara događaj koji se dodaje u red događaja i izvršava se onim redosledom kojim su dospeli u red. SDL biblioteka ima mogućnost stvaranja konteksta i može da koristi OpenGL kontekste na različitim platformama. Time je omogućeno korišćenje opcija iz SDL biblioteke (obrada zvuka, rukovanje događajima, rukovanje nitima, rukovanje tajmerima) u OpenGL aplikacijama. Takođe je preko određenih funkcija moguć direktan upis u dvodimenzionalnu prihvatnu memoriju slike.

Iako je SDL biblioteka sa pristupom na niskom nivou, moguće je razvijati višeplatformske prenosive aplikacije sa visokim nivoom fleksibilnosti. SDL biblioteka ima u svom nazivu reč sloj (engl. Layer), jer je to zapravo omotač oko specifične funkcionalnosti operativnog sistema. Cilj je da se obezbedi programsko okruženje za pristup specifičnoj funkcionalnosti. Jezik u kojem je pisana SDL biblioteka je programski jezik C, ali biblioteka može da se koristi sa sledećim jezicima:

C++	Haskell	PHP	Euphoria
Ada	Java	Pike	Guile
C#	Lisp	Pliant	Pascal
D	Lua	Python	Perl
Eiffel	ML	Ruby	Tcl
Erlang	Objective C	Smalltalk	

Tabela 2.2 Podržani programski jezici

SDL biblioteka podržava operativne sisteme koji su dati u narednoj tabeli.

Zvanično podržani operativni sistemi	Nezvanično podržani operativni sistemi
Linux	AmigaOS
Windows	Dreamcast
Windows CE	Atari
BeOS	AIX
MacOS	OSF/Tru64
Mac OS X	RISC OS
FreeBSD	Symbian OS
NetBSD	OS/2
OpenBSD	
BSD/OS	
Solaris	
IRIX	
QNX	

Tabela 2.3 Podržani operativni sistemi

SDL biblioteka pored podrške pristupu na niskom nivou sadrži i dodatne biblioteke koje proširuju funkcionalnost SDL biblioteke. Dodatne biblioteke su:

- SDL_image - podrška za različite formate slika
- SDL_mixer - podrška za kompleksne audio funkcije
- SDL_net - podrška za mrežnu komunikaciju
- SDL_ttf - podrška za prikaz fontova
- SDL_rtf - podrška za iscrtavanje formatiranog teksta

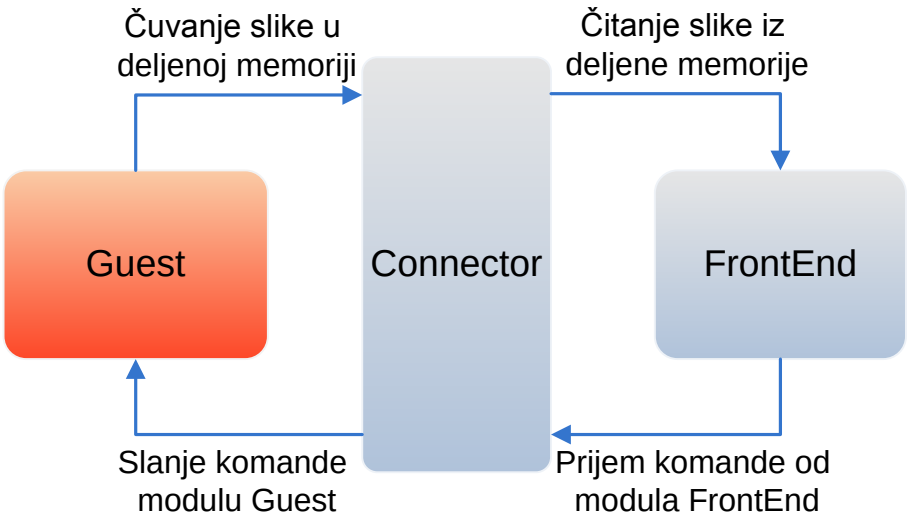
SDL_image podržava formate slika sa sledećim proširenjima: bmp, gif, jpeg, lbm, pcx, png, pnm, tga, tiff, xcf, xpm.

3. Analiza problema

Koncept rešenja predstavlja realizaciju programskog okruženja koje obezbeđuje jednostavan prenos programske podrške namenskog uređaja na iPhone platformu, uz vernu emulaciju grafičkog izgleda programa i funkcionalnosti. Uz upotrebu predloženog rešenja tokom razvoja programske podrške namenskog uređaja ono se kasnije veoma lako prenosi na iPhone, uz tačno preslikavanje svih funkcija i grafičke korisničke sprege. Programsko okruženje obezbeđuje i odgovarajući nivo apstrakcije fizičke arhitekture, prvenstveno tastera. Predlog rešenja se sastoji iz tri modula:

- *FrontEnd*
- *Connector*
- *Guest*

FrontEnd modul služi za slanje događaja (npr. pritisak na neki od tastera, dodir ekrana) *Guest* modulu preko *Connector* modula, za čitanje izgleda grafičke sprege iz deljene memorije i za prikaz istog na ekranu prenosivog uređaja. *Connector* modul služi za razmenu informacija između *FrontEnd* modula i *Guest* modula. *FrontEnd* i *Connector* moduli predstavljaju programsko okruženje. *Guest* modul se može posmatrati kao komponenta koja se pobuđuje komandama i događajima, a na koje reaguje izmenom internog stanja i odgovarajućom promenom prikazanog grafičkog sadržaja. *Guest* modul predstavlja programsku podršku namenskog uređaja realizovanu u skladu sa izloženim konceptom. Prenos *Guest* modula na iPhone se svodi samo na prevođenje izvornog koda na iPhone platformi. *Guest* modul prima komande preko utičnice (engl. socket). Utičnice kao vid komunikacije između procesa su podržane na velikom broju platformi, time je omogućeno korišćenje *Guest* modula koji je napisan u nekom drugom jeziku ili nekoj drugoj platformi.



Slika 3.1 Koncept rešenja

Stvaranjem odvojenih modula *FrontEnd* i *Guest* omogućen je jednostavan prenos programske podrške na iPhone uređaj, time što se sve komponente koje zavise od platforme izmeštaju van *Guest* modula. To se ogleda u činjenici da će se, u slučaju implementacije nekog drugog namenskog uređaja, menjati samo *Guest* modul, dok se ostali moduli mogu iskoristiti bez izmena. Programska podrška se realizuje na isti način kako na namenskom uređaju tako i na iPhone uređaju u sklopu *Guest* modula. Programska podrška na namenskom uređaju u ovom radu je realizovana preko SDL biblioteke. Takođe će i emulirana programska podrška u okviru *Guest* modula biti realizovana pomoću SDL biblioteke. Time će se znatno skratiti vreme potrebno za prenos programske podrške.

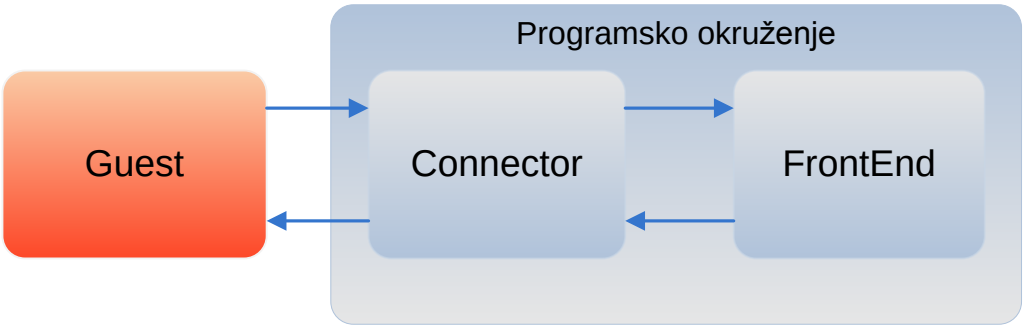
Parametri emulacije fizičke arhitekture uređaja koji će biti prikazan na ekranu će se učitavati iz ulazne XML datoteke. Ulazna XML datoteka treba da sadrži izgled fizičke arhitekture namenskog uređaja, izgled dugmadi koje namenski uređaj sadrži, izgled upravljačkog točkića i veličinu ekrana namenskog uređaja. Svaki od ovih elemenata treba da sadrži svoje dimenzije i koordinate na kojima se iscrtava na ekranu iPhone uređaja. Učitavanjem izgleda fizičke arhitekture namenskog uređaja preko ulazne XML datoteke omogućena je lakša izmena izgleda uređaja, ubacivanjem izmena samo u XML datoteku. Takođe se prilikom realizacije nekog drugog namenskog uređaja menja samo XML datoteka.

Kao potvrda predloženog koncepta odabran je razvoj programske podrške namenskog uređaja za kontrolu svetla, i preneta je na iPhone platformu uz upotrebu realizovanog programskog okruženja. Pri tome je obezbeđena verodostojna emulacija grafičkog izgleda i funkcionalnosti.

Kompanije Control4, Crestron ili Savant systems nude svoja rešenja na iPhone uređajima, ali sa bitnom razlikom. Njihova rešenja su realizovana na iPhone platformi konverzijom celokupne programske podrške namenskog uređaja. Kod ovakvog pristupa se javlja problem pri pokušaju emuliranja nekog drugog namenskog uređaja, jer se vreme potrebno za izradu aplikacije ne smanjuje. Odnosno ponovo je potrebno emulirati celokupnu programsku podršku namenskog uređaja. U ovom rešenju je izvršen jednostavan prenos programske podrške zahvaljujući realizovanom programskom okruženju, uz izmenu samo *Guest* modula. Svaka sledeća promena namenskog uređaja dovodi do kraćeg vremena potrebnog za prenos programske podrške zahvaljujući realizovanom programskom okruženju.

4. Opis rešenja

Cilj ovog rada je implementacija programskog okruženja za efikasan prenos programske podrške namenskog uređaja. Prvi korak u izradi ovog rada je kreiranje programskog okruženja. Programsko okruženje služi za slanje poruka *Guest* modulu, za čitanje izgleda grafičke sprege iz deljene memorije i za prikaz istog na ekranu prenosivog uređaja. Programsko okruženje predstavlja nepromenljivi deo aplikacije koja je opisana u ovom radu, tj. kada se jednom implementira nastavlja da se koristi bez ikakve promene. U okviru programskog okruženja su implementirani *Connector* i *FrontEnd* moduli.

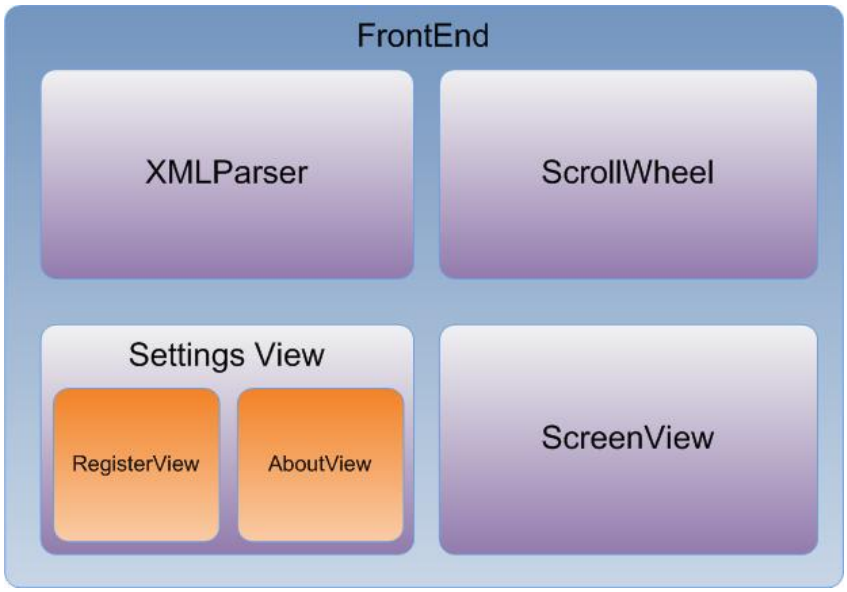


Slika 4.1 Programsko okruženje i Guest modul

Drugi korak u izradi rada jeste kreiranje *Guest* modula. *Guest* modul predstavlja internu logiku namenskog uređaja i to je promenljivi deo aplikacije. U ovom radu *Guest* modul predstavlja internu logiku namenskog uređaja za kontrolu svetla. U slučaju da se želi implementirati neki drugi namenski uređaj, sa nekom drugom funkcionalnošću, tada se menja samo *Guest* modul.

4.1 FrontEnd modul

FrontEnd modul obezbeđuje integraciju grafičke sprege namenskog uređaja i prikaz sadržaja deljene grafičke memorije u koju upisuje *Guest* modul. Ovaj modul služi za slanje događaja *Guest* modulu preko *Connector* modula, za čitanje izgleda grafičke sprege iz deljene memorije i za prikaz istog na ekranu prenosivog uređaja. Za slanje događaja poziva se metoda iz *Connector* modula pod nazivom *receiveMessage()*, dok se za čitanje iz deljene memorije koristi metoda *memoryRead()*.



Slika 4.2 FrontEnd modul

Kao što je prikazano na slici *FrontEnd* modul obuhvata četiri podmodula: XMLParser, ScrollWheel, ScreenView i SettingsView. Podmodul SettingsView sadrži u sebi dva podmodula: RegisterView i AboutView.

4.1.1 XMLParser modul

XML je standardni skup pravila za definisanje formata podataka u elektronskoj formi. Definiše opštu sintaksu za označavanje podataka pomoću odgovarajućih etiketa (engl. Tags) koje imaju poznato ili lako razumljivo značenje. Omogućava stvaranje dugotrajnih formata podataka koji su nezavisni od platforme. Osnovna svrha XML jezika jeste olakšano deljenje podataka kroz različite informacione sisteme, posebno kroz one sisteme koji su povezani sa internetom.

U ovom radu XML datoteka je korišćena za učitavanje izgleda emulirane fizičke arhitekture namenskog uređaja. Za uspešno otvaranje datoteke i njeno uspešno čitanje potrebno je implementirati parser XML datoteke. XMLParser modul radi upravo to, parsira ulaznu XML

datoteku. Ulazna XML datoteka sadrži elemente koji opisuju fizičku arhitekturu uređaja. Ukoliko se želi izmeniti izgled namenskog uređaja potrebno je menjati ulaznu XML datoteku. Korišćenjem XML datoteke je omogućena laka promena izgleda namenskog uređaja, jer se ne menja nista u kodu aplikacije. Takođe je moguće da korisnik napravi XML datoteku sa proizvoljnim izgledom uređaja. Elementi u XML datoteci su:

- `background` - izgled fizičke arhitekture namenskog uređaja
- `button` - izgled dugmića namenskog uređaja
- `scrollWheel` - izgled upravljačkog točkića
- `display` - pozicija i dimenzija ekrana namenskog uređaja

Element `background` zadaje putanju do datoteke sa slikom koja predstavlja izgled fizičke arhitekture namenskog uređaja. Izgled elementa `background` je sledeći:

```
<background>EmbeddedDevice.png</background>
```

Element `button` zadaje putanju do datoteke sa slikom dugmeta koje je u normalnom stanju (nije pritisnuto), slikom dugmeta kada je pritisnuto, jedinstvenim kodom dugmeta, koordinatama dugmeta na ekranu i veličini dugmeta.

```
<button>
  <normal_image>Button.png</normal_image>
  <pressed_image>PressedButton.png</pressed_image>
  <button_code>number</button_code>
  <button_x>X coord</button_x>
  <button_y>Y coord</button_y>
  <button_width>Width</button_width>
  <button_height>Height</button_height>
</button>
```

Element `button` je element roditelj koji sadrži sedam elemenata potomaka. Prvi element potomak jeste `normal_image` koji sadrži putanju do datoteke sa slikom dugmeta u normalnom stanju. Element `pressed_image` sadrži putanju do datoteke sa slikom dugmeta u pritisnutom stanju. Element `button_code` sadrži jedinstveni kod dugmeta. Elementi `button_x` i `button_y` sadrže X odnosno Y koordinate na kojima će dugme biti prikazano na ekranu. Elementi `button_width` i `button_height` sadrže širinu i visinu dugmeta.

Element `scrollWheel` zadaje putanju do datoteke sa slikom upravljačkog točkića, koordinatama točkića na ekranu i prečnikom točkića.

```
<scrollWheel>
  <wheel_image>scrollWheel.png</wheel_image>
  <wheel_x>X coord</wheel_x>
  <wheel_y>Y coord</wheel_y>
  <wheel_radius>radius value</wheel_radius>
</scrollWheel>
```

Element `scrollWheel` sadrži četiri elementa potomka. Element `wheel_image` zadaje putanju do datoteke sa slikom upravljačkog točkića. Elementi `wheel_x` i `wheel_y` sadrže X odnosno Y koordinate na kojima će upravljački točkić biti prikazan na ekranu. Element `wheel_radius` sadrži informaciju o prečniku upravljačkog točkića.

Element `display` zadaje putanju do datoteke sa koordinatama na kojima se nalazi ekran namenskog uređaja koji se prikazuje na ekranu iPhone uređaja i dimenzije ekrana.

```
<display>
  <display_x>X coord</display_x>
  <display_y>Y coord</display_y>
  <display_width>Width</display_width>
  <display_height>Height</display_height>
</display>
```

Element `display` sadrži četiri elementa potomka. Elementi `display_x` i `display_y` sadrže X odnosno Y koordinate na kojima će ekran biti prikazan na ekranu. Elementi `display_width` i `display_height` sadrže širinu i visinu ekrana.

Pri parsiranju ulazne XML datoteke koristi se sistemska klasa NSXMLParser. Klasa NSXMLParser sadrži metode:

- `didStartElement` - metoda pronalazi početak elementa
- `didEndElement` - metoda pronalazi kraj elementa
- `foundCharacters` - metoda čita karaktere iz elementa
- `parseErrorOccured` - metoda javlja grešku u parsiranju

4.1.2 ScrollWheel modul

ScrollWheel modul predstavlja emulaciju upravljačkog točkića kod koga je prilikom prevlačenja točkića u levo ili desno moguće upravljanje nekim sadržajem. Npr. moguće je listanje teksta, listanje slika, kontrolisanje muzičkih numera. Upravljački točkić je svoju slavu stekao na iPod uređaju, gde je korišćen za upravljanje muzičkim numerama. Pored kontrolisanja muzičkih numera korišćen je i za upravljanje u meniju samog uređaja. Na narednoj slici je prikazan upravljački točkić na iPod uređaju koji kontroliše muzičke numere.

Slika 4.3 Uređaj sa upravljačkim točkićem

U ovom radu se koristi za kontrolu nad osvetljenjem, odnosno kao potenciometar. Pomeranjem točkića u smeru kazaljke na satu se povećava izlazna snaga sijalice, dok se pomeranjem u suprotnom smeru od smera kazaljke na satu smanjuje izlazna snaga sijalice. Metode implementirane u ScrollWheel modulu su:

- `centeredPoint` - centralna tačka
- `getAngle` - ugao za koji je pomeren točkić
- `pointInsideRadius` - ispitivanje da li je dodirnut točkić
- `setImageView` - postavljanje slike točkića na ekran
- `beginTrackingWithTouch` - početak praćenja pomeraja točkića
- `continueTrackingWithTouch` - nastavak praćenja pomeraja točkića

4.1.3 ScreenView modul

ScreenView modul inicijalizuje okvir u kom će biti prikazan izgled emuliranog ekrana namenskog uređaja. Sadržaj koji se prikazuje na ekranu se čita iz deljene memorije. Metode u ScreenView modulu su:

- `initWithFrame` - inicijalizacija okvira koji predstavlja ekran uređaja
- `setImageData` - prihvat slike iz deljene memorije
- `drawRect` - prikaz slike u prethodno inicijalizovanom okviru

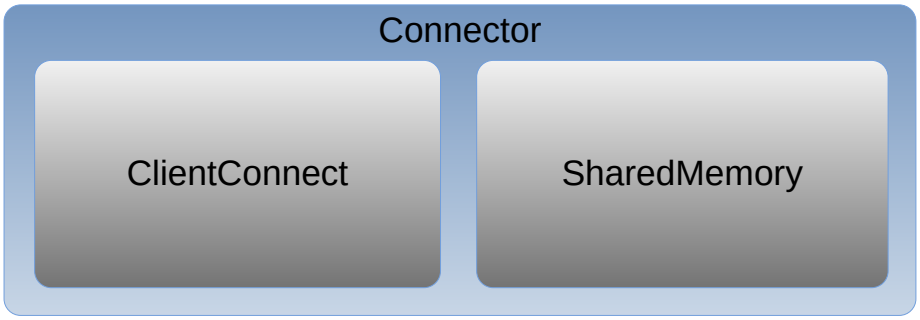
4.1.4 SettingsView modul

SettingsView modul služi za prikaz novog prozora na iPhone koji omogućava prikaz dodatnih opcija u programu. Prozor, koji je formiran u SettingsView modulu, sadrži Register dugme i About dugme. Prozor se prikazuje na ekranu iPhone uređaja pritiskom na info dugme koje se nalazi u donjem desnom uglu početnog ekrana iPhone uređaja. Dugmići koji su formirani u novom prozoru sadrže sledeće akcije:

- Register - prozor u kome je moguće registrovati aplikaciju
- About - prozor za prikaz dodatnih informacija o aplikaciji

4.2 Connector modul

Connector modul služi za komunikaciju *FrontEnd* i *Guest* modula. *Connector* modul treba da obezbedi mrežnu komunikaciju preko mrežnih utičnica (engl. socket). Mrežna komunikacija se obezbeđuje preko podmodula ClientConnect. *Connector* modul prihvata komande od *FrontEnd* modula preko metode *receiveMessage()* i šalje ih *Guest* modulu preko utičnica koristeći metodu *writeToServer()*.



Slika 4.4 Connector modul

Na slici je prikazan *Connector* modul i on obuhvata dva podmodula: SharedMemory i ClientConnect. SharedMemory modul predstavlja deljenu memoriju.

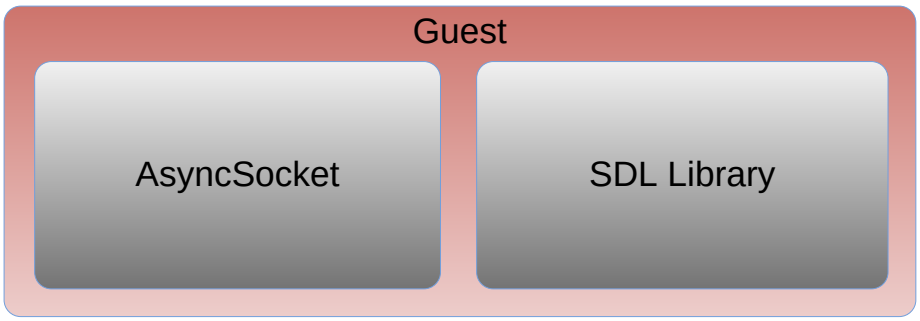
4.2.1 ClientConnect modul

ClientConnect modul omogućava uspostavljanje veze sa poslužiocem. Veza se stvara preko mrežnih utičnica (engl. socket). Poslužilac u ovom radu je *Guest* modul. Metode u ClientConnect modulu su:

- `connectToServerUsingStream` - uspostavljanje veze sa uslužiocem
- `writeToServer` - slanje podataka ka uslužiocu
- `streamHandleEvent` - prijem podataka od uslužioca
- `disconnect` - prekid veze sa uslužiocem

4.3 Guest modul

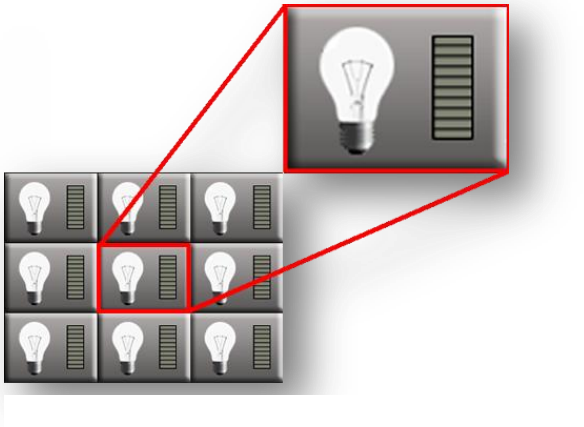
Guest modul oponaša internu logiku namenskog uređaja. *Guest* modul se može posmatrati kao komponenta koja se pobuđuje komandama i događajima, a na koje reaguje izmenom internog stanja i odgovarajućom promenom prikazanog grafičkog sadržaja. Poruke koje *Guest* modul prima od programskog okruženja se šalju preko mrežnih utičnica. Nakon što primi komandu, *Guest* modul pokreće novu nit koja poziva metodu `createImage`. Metoda `createImage` na osnovu date komande iscrtava sliku koja će biti sačuvana u deljenoj memoriji. Kada se slika sačuva u deljenoj memoriji nit se završava.



Slika 4.5 Guest modul

Guest modul obuhvata dva podmodula: AsyncSocket i SDL Library. SDL biblioteka je uvezana statički, kao što je to zahtevano od strane proizvođača iPhone - Apple. Metode koje se pozivaju unutar nove niti i pomoću kojih je moguće napraviti sliku su:

- `createRects` - metoda inicijalizuje segmente za svaku sijalicu koji će biti prikazani na ekranu i popunjava ih predefinisanim sadržajem
- `allLightsOff` - metoda gasi sve sijalice u sistemu
- `allLightsOn` - metoda pali sve sijalice u sistemu
- `powerStatusUp` - metoda povećava izlaznu snagu sijalice
- `powerStatusDown` - metoda smanjuje izlaznu snagu sijalice
- `toggleLight` - metoda pali/gasi sijalicu u zavisnosti od njenog prethodnog stanja
- `selectLamp` - metoda vraća broj odabrane lampe
- `selectRect` - metoda odabira segment koji sadrži odabranu lampu
- `saveScreenToMemory` - metoda čuva formiranu sliku u deljenu memoriju



Slika 4.6 Segment sa sijalicom

Na prethodnoj slici je prikazan izgled segmenta koji odgovara jednoj sijalici. Segment sadrži sliku sijalice i skalu koja grafički prikazuje izlaznu snagu sijalice. Slika sijalice se takođe menja prema promeni izlazne snage.

4.3.1 AsyncSocket modul

AsyncSocket modul omogućava asinhronu komunikaciju sa mrežnom utičnicom. U ovom radu AsyncSocket modul je korišćen za prijem komandi preko utičnice od *Connector* modula. Funkcije iz AsyncSocket modula koje su korištene u radu su:

- `acceptOnPort` - uspostava veze na datom prolazu
- `didAcceptNewSocket` - poziva se kada je prihvaćena veza preko utičnice
- `didConnectToHost` - poziva se kada je veza uspostavljena i kada je utičnica spremna za čitanje i upis
- `didWriteDataWithTag` - poziva se kada je utičnica završila upis traženih podataka
- `didReadData` - poziva se kada je utičnica pročitala sve podatke

- `willDiconnectWithError` - poziva se u slučaju greške i zatvara utičnicu
- `onSocketDidDisconnect` - poziva se prilikom prekida veze, bez obzira da li je javljena greška ili ne

4.4 Format poruka

Komande koje zadaje korisnik se interpretiraju u modulu *FrontEnd* koji ih pretvara u poruke koje se dalje preko *Connector* modula šalju *Guest* modulu. Tipovi komandi koje postoje u aplikaciji su:

- komande koje predstavljaju pritisak dugmeta
- komande koje predstavljaju pritisak na ekran
- komande koje predstavljaju pritisak i pomeranje točkića za upravljanje.

U aplikaciji postoji sedam različitih dugmadi, a samim tim postoji i sedam različitih komandi koje predstavljaju pritisak na dugme. Poruke koje se šalju *Guest* modulu na osnovu pritiska određenog dugmeta su sledeće:

- `button_Ok` – poruka kojom se pali/gasi odabrana sijalica
- `button_Up` – poruka kojom se uvećava izlazna snaga sijalice
- `button_Down` – poruka kojom se smanjuje izlazna snaga sijalice
- `button_Left` – poruka kojom se vrši odabir prethodne sijalice
- `button_Right` – poruka kojom se vrši odabir naredne sijalice
- `button_AllLightsOn` – poruka kojom se pale sve sijalice
- `button_AllLightsOff` – poruka kojom se gase sve sijalice

Namenski uređaj kao i iPhone uređaj sadrži ekran osetljiv na dodir samim tim su potrebne komande za pritisak na ekran. Komanda koja predstavlja pritisak na ekran sadrži koordinate mesta gde je dodirnut ekran. Pritiskom na ekran moguće je odabrati željenu sijalicu. Poruka koja se šalje *Guest* modulu ima sledeći oblik:

`pressed_Xcoord_Ycoord` – poruka kojom se vrši odabir sijalice

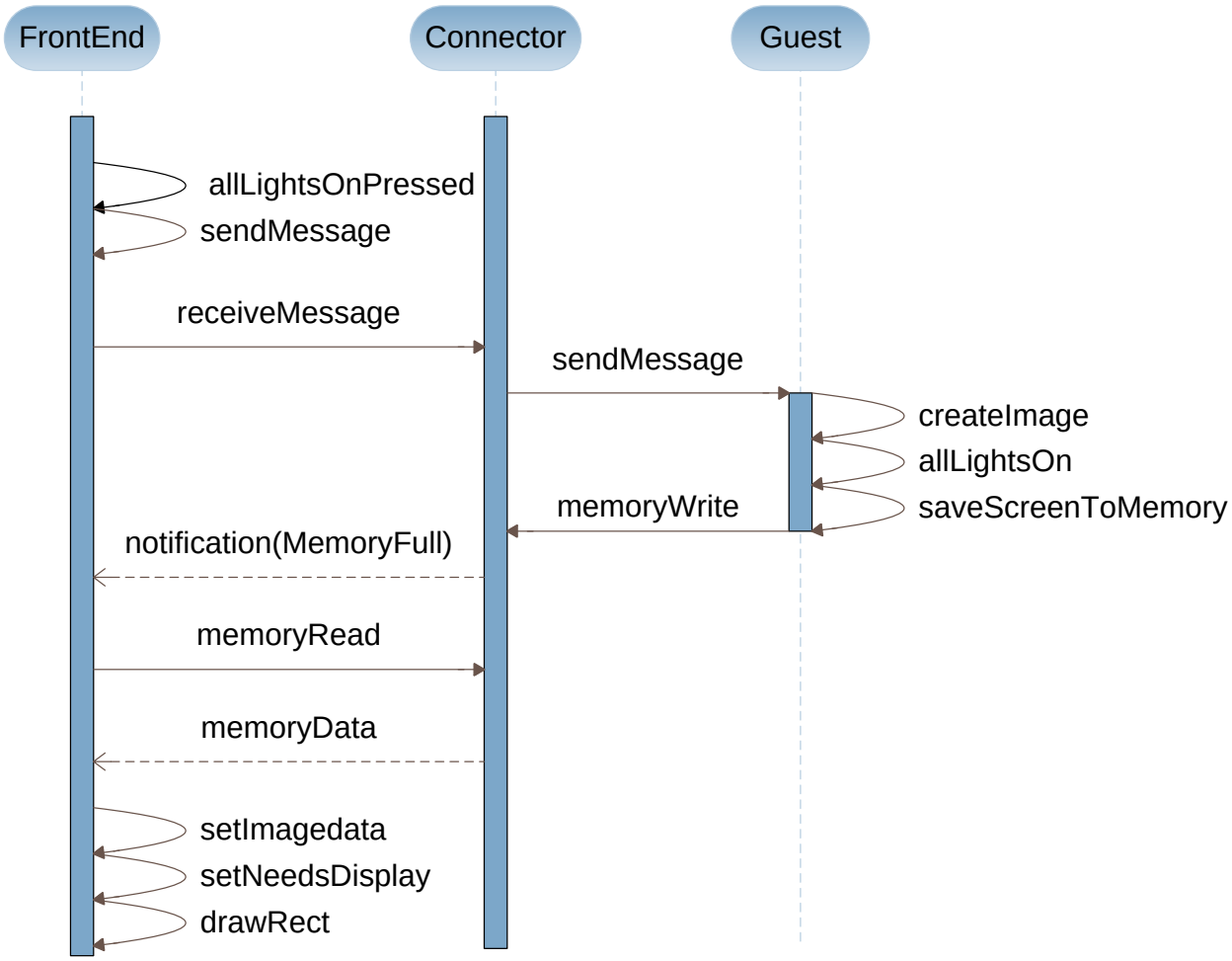
Svaka sijalica na ekranu je predstavljena segmentom na kome se nalazi slika sijalice i skala koja označava trenutnu vrednost izlazne snage sijalice. Sijalica će biti odabrana ukoliko se tačka sa poslatim koordinatama nalazi unutar segmenta koji predstavlja željenu sijalicu.

Poslednji tip komandi jesu komande koje predstavljaju pomeranje točkića za upravljanje. Postoje dve komande koja predstavljaju pomeranje točkića za upravljanje. Koja komanda će biti

poslata zavisi od toga da li se upravljački točkić pomera u smeru kazaljke na satu ili obrnuto. Poruke koje se šalju za ova dva tipa komandi su:

- `scrollWheel_down` – poruka kojom se smanjuje izlazna snaga sijalice
- `scrollWheel_up` – poruka kojom se povećava izlazna snaga sijalice

Scenario slanja poruke je prikazan na narednoj slici. Komanda koja se šalje predstavlja komandu za paljenje svih sijalica u sistemu (`button_AllLightsOn`).



Slika 4.7 Scenario slanja poruke za paljenje svih sijalica

Pritiskom na dugme AllLightsOn poziva se metoda `allLightsOnPressed()` koja formira poruku. Poruka je u stvari niz karaktera, konkretno u ovom slučaju ima vrednost `button_AllLightsOn`. Nakon toga poziva se metoda `sendMessage()` koja proverava da li je moguće slanje poruke. Promenljiva `isWorkingFlag` označava da li je moguće slanje poruke ili ne. Postavlja se na nulu ukoliko je komanda izvršena, a ukoliko se komanda trenutno izvršava promenljiva ima vrednost jedan. U slučaju da promenljiva `isWorkingFlag` ima vrednost jedan tada se komanda upisuje u red poruka spremnih za slanje. Ako promenljiva ima vrednost nula tada se ispituje da li ima poruka u redu za slanje. Ukoliko ima poruka u redu tada se trenutna

poruka stavlja na kraj reda, a šalje se prva komanda u redu. Ako je red prazan tada se trenutna poruka prosleđuje *Connector* modulu preko metode `receiveMessage()`. *Connector* modul primljenu poruku prosleđuje do *Guest* modula preko metode `sendMessage()`. U *Guest* modulu se poziva metoda `createImage()` u kojoj se stvara slika koja će biti prikazana na ekranu namenskog uređaja. Unutar pomenute metode se poziva metoda koja izvršava komandu poslatu u poruci i poziva se takođe metoda `saveScreenToMemory()` koja treba da sačuva izgled ekrana u deljenu memoriju. Metoda `saveScreenToMemory()` poziva metodu `memoryWrite()` iz *Connector* modula koja sačuvane podatke iz deljne memorije pretvara u tip NSMutableData iz koga će se kasnije čitati slika. Nakon toga se šalje obaveštenje *FrontEnd* modulu da je slika sačuvana u memoriji. *FrontEnd* modul zatim poziva metodu `setImageData()` koja poziva metodu `memoryRead()` iz *Connector* modula za čitanje podataka iz memorije. Kada su podaci pročitani iz memorije pozivaju se metode `setNeedsDisplay()` i `drawRect()` koje iscrtavaju sliku iz memorije na ekran. Princip slanja ostalih poruka u sistemu je sličan, s tim da se pozivaju metode koje odgovaraju datoj komandi.

4.5 Izgled emuliranog uređaja

Na narednoj slici je prikazan izgled emuliranog namenskog uređaja.

Slika 4.8 Izgled emuliranog uređaja

Gornji deo ekrana je rezervisan za emulirani ekran namenskog uređaja koji sadrži kontrolisane sijalice. Ekran je osetljiv na dodir tako da je moguće odabiranje željene sijalice bez potrebe da se koriste navigacioni dugmići. Donji deo ekrana prikazuje emuliranu fizičku arhitekturu namenskog uređaja tj. emulirane fizičke kontrole namenskog uređaja. Fizičke kontrole koje su emulirane obuhvataju dugme za paljenje svih sijalica, dugme za gašenje svih sijalica, navigacione dugmiće i upravljački točkić. Razlika između emulacije ekrana i emulacije fizičke arhitekture namenskog uređaja je u tome odakle se učitava. Emulirani ekran se učitava iz *Guest* modula, dok se emulirana fizička arhitektura učitava iz ulazne XML datoteke.

5. Testiranje

Koncept je testiran prenosom programske podrške namenskog uređaja za kontrolu svetla na iPhone. Prvo testiranje aplikacije je obavljeno na simulatoru gde su testirani moduli aplikacije, zatim je testirana i funkcionalnost aplikacije. Testiranje funkcionalnosti je izvršeno odabiranjem i izvršavanjem seta testnih slučajeva.

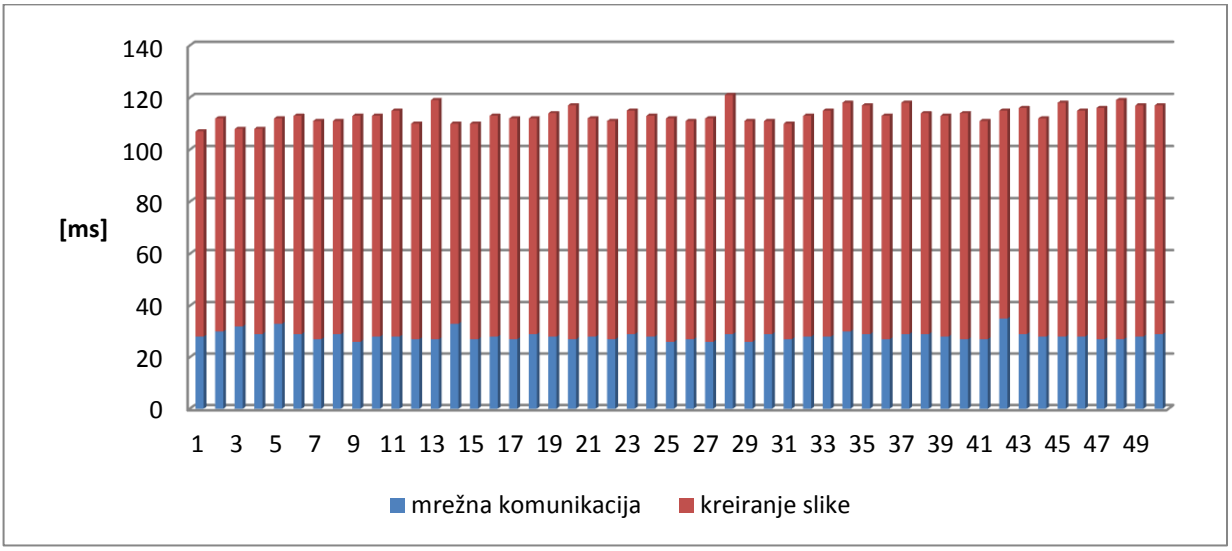
Broj testa	Opis testa	Očekivano ponašanje	Test prošao
1	Pritisak na dugme Ok	Ukoliko je sijalica upaljena na dugme se gasi, a ukoliko je ugašena na dugme se pali.	DA
2	Pritisak na dugme Right	Pritiskom na dugme se odabira prva sijalica u nizu. Smer kretanja je u desno.	DA
3	Pritisak na dugme Left	Pritiskom na dugme se odabira poslednja sijalica u nizu. Smer kretanja je u levo.	DA
4	Pritisak na dugme Up	Pritiskom na dugme se povećava izlazna snaga odabrane sijalice.	DA
5	Pritisak na dugme Down	Pritiskom na dugme se smanjuje izlazna snaga odabrane sijalice.	DA
6	Pritisak na dugme All on	Pritiskom na dugme se pale sve sijalice u sistemu.	DA
7	Pritisak na dugme All off	Pritiskom na dugme se gase sve sijalice u sistemu.	DA
8	Pritisak na dugme Info	Pritiskom na dugme se prikazuje novi prozor.	DA
9	Pritisak na dugme About	Pritiskom na dugme se prikazuje novi prozor u kom se nalazi više informacija o aplikaciji.	DA
10	Pritisak na dugme Register	Pritiskom na dugme se prikazuje novi prozor u kome je moguće registrovati aplikaciju.	DA
11	Unos registracionog ključa	Prilikom unosa ključa prikazana je odgovarajuća ikona, koja govori korisniku da li je unet tačan ključ ili ne.	DA

12	Dodir ekrana	Prilikom dodira ekrana odabira se sijalica koja se nalazi na datim kordinatama	DA

Tabela 5.1 Funkcionalno testiranje

Testovi predstavljeni u tabeli 5.1 opisuju ponašanje sistema na pritisak određenog dugmeta na emuliranom namenskom uređaju. Pored ovog ponašanja opisuju i ponašanje sistema prilikom otvaranja novog prozora za registrovanje aplikacije i prozora za prikaz dodatnih informacija o aplikaciji. Pošto namensko uređaj sadrži ekran osetljiv na dodir, testirano je ponašanje sistema prilikom izvršenja komande koja predstavlja dodir ekrana namenskog uređaja.

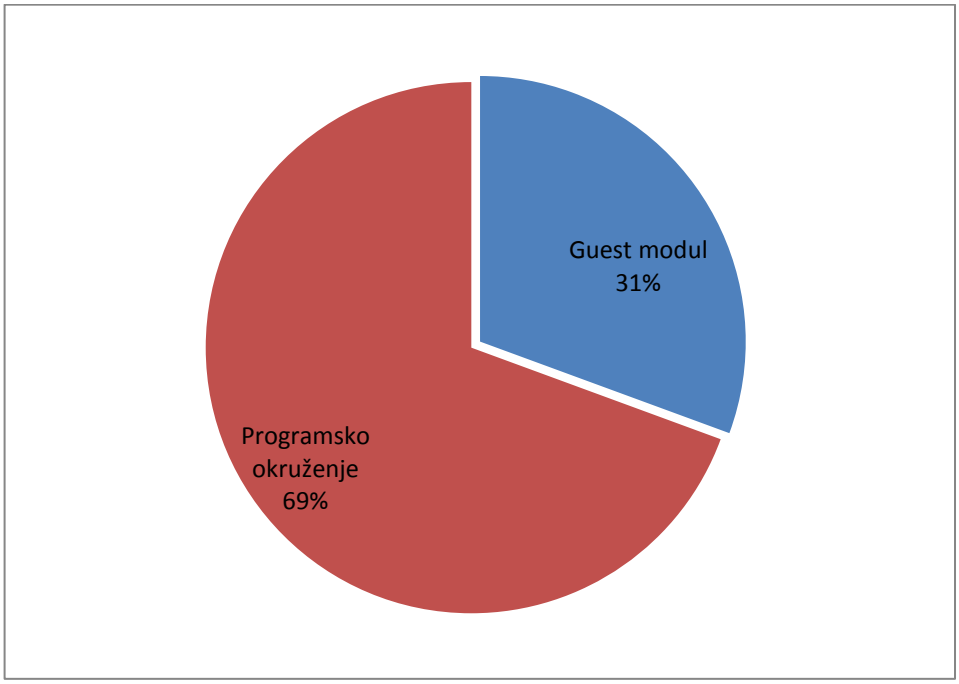
Sledeće testiranje je obavljeno na iPhone uređaju gde je testirana brzina odziva na zadate komande. Na slici je prikazan grafik na kome je prikazano vreme potrebno za *Guest* modul da proizvede sliku i vreme koje se utroši na mrežnu komunikaciju. Testni slučaj obuhvata pedeset odbiraka i svaki od njih predstavlja jednu istu komandu. Prosečno vreme potrebno da bi se napravila slika je 85 milisekundi. Na mrežnu komunikaciju se potroši prosečno 28 milisekundi, što bi u ukupnom zbiru značilo da je prosečno vreme za obradu komande 117 milisekundi.



Slika 5.1 Vreme odziva

Pored testiranja vremena odziva još jedna bitna karakteristika ovog rada jeste veličina koda tj. broj linija koda potreban za kreiranje sa jedne strane celog programskog okruženja i sa druge strane *Guest* modula. Cilj ovog rada je bio efikasan prenos programske podrške namenskog uređaja zahvaljujući kreiranom programskom okruženju. Dve trećine koda predstavlja programsko okruženje koje se ne menja, jedini deo u radu koji je promenljiv jeste *Guest* modul. Na osnovu ovog podatka je pokazano da je ovo efikasniji način za prenos programske podrške namenskih uređaja, jer se implemetiranjem nekog drugog namenskog uređaja menja samo

trećina koda. Na sledećem grafiku je prikazan odnos u veličini koda za *Guest* modul i za programsko okruženje.



Slika 5.2 Odnos u veličini koda

Kao što je ranije rečeno zahvaljujući realizovanom programskom okruženju ovo rešenje se odlikuje kraćim vremenom za ponovni razvoj aplikacija na iPhone. Prema ovom grafiku gledano to je tri puta kraće nego vreme razvoja bez programskog okruženja. U slučaju da postoji programsko okruženje na namenskom uređaju i na iPhone uređaju, prenos *Guest* modula na iPhone, ukoliko je kao u radu realizovan preko SDL biblioteke, se svodi na uvezivanje koda namenskog uređaja. Ovakav prenos predstavlja manje od 5% od vremena potrebnog za prenos cele aplikacije na iPhone. Ukoliko je *Guest* modul realizovan preko neke druge biblioteke koja nije podržana na iPhone uređaju samim tim i procenat vremena potrebnog za prenos programske podrške raste. U slučaju da ne postoji programsko okruženje tada je prenos aplikacije sa namenskog uređaja na iPhone uređaj znatno duži. Pretpostavka je da je za prenos aplikacije na iPhone potrebno 50% vremena od ukupnog vremena razvoja aplikacije na namenskom uređaju.

Slika 5.3 Izgled dva uređaja

Na slici je prikazan izgled namenskog uređaja i izgled emuliranog namenskog uređaja na iPhone uređaju. Ova slika pokazuje da je izgled namenskog uređaja verno emuliran na iPhone platformi. Verna emulacija proizilazi iz toga što namenski uređaj i iPhone uređaj koriste SDL biblioteku za prikaz grafičke sprege.

6. Zaključak

U ovom radu je uspešno realizovano programsko okruženje koje obezbeđuje jednostavan prenos programske podrške namenskog uređaja na iPhone platformu uz izmenu samo *Guest* modula. Kao što je pokazano u testiranju, postignute su željene performanse, funkcionalnost, brzina odziva i izgled namenskog uređaja. Za razliku od prenosa cele aplikacije na iPhone ovo rešenje se zahvaljujući realizovanoj programskoj podršci odlikuje značajno kraćim periodom vremena potrebnog za prenos na iPhone.

Pravac daljeg razvoja može da ide u smeru proširenja podrške za uvođenje novih klasa kontrole. Mogao bi da se implementira odabir između više namenskih uređaja, a odabrani uređaj bi se prikazao na ekranu i bila bi omogućena emulacija njegove funkcionalnosti. Dalji razvoj može da ide i u smeru proširenja broja uređaja na kojima se emulira namenski uređaj. Prvi korak može biti podrška za iPad uređaje. Zatim je moguće proširenje na uređaje na Android platformi, kao i druge prenosne uređaje.

7. Literatura

[1] Maher Ali, *Advanced Mobile Development for Apple iPhone and iPod Touch*, Wiley, Septembar 2009

[2] Bill Dudney, Chris Adamson: *iPhone SDK Development - Building iPhone Applications*, The Pragmatic Programmers, Decembar 2009

[3] Jonathan Zdziarski, *iPhone SDK Application Development,1st Edition*, O'Reilly Media, Januar 2009

[4] Erica Sadun: *The iPhone Developer`s Cookbook – Building Applications with the iPhone SDK*, Addison-Wesley Professional, Oktobar 2008

[5] Control4, proizvođač sistema za automatizaciju kuća, www.control4.com

[6] Crestron, proizvođač sistema za automatizaciju kuća, www.crestron.com

[7] Savant, proizvođač sistema za automatizaciju kuća, www.savantav.com

[8] SDL biblioteka, www.libsdl.org