Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Denis Švara

Upravljanje avtomatiziranega sistema z govornimi ukazi

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Uroš Lotrič

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00331/2012
Datum: 04.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DENIS ŠVARA**

Naslov: **UPRAVLJANJE AVTOMATIZIRANEGA SISTEMA Z GOVORNIMI UKAZI**
CONTROL OF AUTOMATED SYSTEM WITH VOICE COMMANDS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V pametnih hišah sodobni dosežki s področij avtomatizacije, komunikacije, varnosti in umetne inteligence povečujejo udobje in dvigajo kakovost življenja uporabnikov. Raziščite možnosti za upravljanja pametne hiše z govornimi ukazi preko pametnega telefona in izdelajte pilotski sistem.

Mentor:
prof. dr. Uroš Lotrič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Denis Švara, z vpisno številko **63070153**, sem avtor diplomskega dela z naslovom:

Upravljanje avtomatiziranega sistema z govornimi ukazi

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Uroša Lotriča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 26. novembra 2012 Podpis avtorja:

Zahvaljujem se izr. prof. dr. Urošu Lotriču za napotke, nasvete in pomoč pri izdelavi diplomske naloge.

Zahvaljujem se svojim staršem in dekletu, ki so me podpirali in vzpodbujali tekom študija.

Zahvaljujem se tudi prijatelju Janiju Gračnarju in kolegu Gregorju Rijavcu za strokovne nasvete.

Kazalo

Povzetek

V pametnih hišah sodobni dosežki s področij avtomatizacije, komunikacije, varnosti in umetne inteligence povečujejo udobje in izboljšujejo kakovosti življenja uporabnikov. V okviru diplomske naloge smo izdelali sistem za upravljanje pametne hiše z govornimi ukazi prek pametnega telefona. Največ poudarka smo namenili govornim ukazom. Uporabnikom želimo dodatno olajšati delo in preiti s komunikacije s prsti - dotikanjem, na bolj naraven, človeški odnos - govor. Sestavili in razvili smo celotno komunikacijsko verigo, po kateri gredo ukazi od pametnega telefona, prek strežnika na krmilnik in obratno.

Ključne besede:

Pametna hiša, avtomatizacija, govorni ukazi, pametni telefon, upravljanje

Abstract

In smart houses contemporary achievements in the fields of automation, communications, security and artificial intelligence, increase comfort and improve the quality of user's lifes. For the purpose of this thesis we developed a system for managing a smart house with voice commands via smart phone. We focused at voice commands most. We want move from communication with fingers - touches, to a more natural, human relationship - speech. We developed the entire chain of communication, by which the commands go from a smartphone, trough a web server to a controller and vice versa.

Keywords:

Smart house, automatization, speech commands, smartphone, management

Poglavje 1

Uvod

Živimo v dobi, kjer je računalništvo del našega vsakdana. Večina ljudi se tega najbrž ne zaveda, ampak tehnologija nas obdaja z vseh strani. Prenosni računalniki, mobilni telefoni, sodobni avtomobili so nekatere izmed mnogih stvari, brez katerih si danes življenje težko predstavljamo. Zadnja leta se tehnologija vztrajno širi tudi v naše domove. Hišna avtomatizacija je vedno bolj aktualna tema. Uporabnikom ponuja nadzor in upravljanje doma, pri tem povečuje udobje in kakovost življenja uporabnikov. Med drugim se je razširila tudi uporaba pametnih telefonov. Večina si nas dneva brez telefona ne predstavlja, spremlja nas na vsakem koraku. Da bi uporabnikom dodatno olajšali delo in povečali udobje, smo upravljanje hišne avtomatizacije prenesli na pametni telefon.

V diplomski nalogi bomo predstavili sistem za govorno upravljanje doma s pametnim telefonom. Sistem vključuje mobilno aplikacijo za pametne telefone podjetja Apple, spletni strežnik podjetja Microsoft in krmilnik podjetja Siemens. V prvem delu diplomske naloge si bomo ogledali uporabljeno strojno opremo in predstavili njene lastnosti. Nadaljevali bomo s predstavitvijo programske opreme in tehnologij, ki smo jih uporabili za razvoj.

V drugem delu diplomske naloge bomo opisali razvoj strežniškega dela sistema. Predstavili bomo knjižnico, ki smo jo uporabili za komunikacijo s krmilnikom in implementacijo upravitelja zahtev. Sledil bo opis razvoja mo-

bilne aplikacije. Predstavili bomo knjižnico za prepoznavanje govora, opisali definicijo govora, njegovo sestavo in postopel prepoznavanja. V nadaljevanju bomo opisali še strukturo aplikacije in njeno uporabo. Za konec bomo povzeli, kaj sistem dela in kako izboljšuje življenje uporabnikom, ter podali predloge za njegovo razširitev in izboljšavo.

Poglavje 2

Strojna oprema

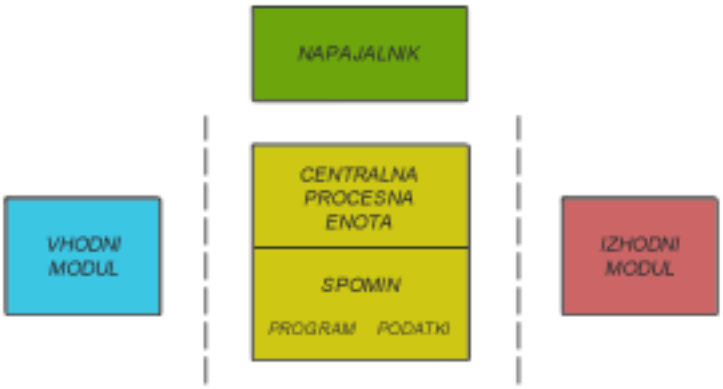
Za delovanje sistema smo potrebovali programirljiv logični krmilnik, mobilni telefon, usmerjevalnik in strežnik. Odločili smo se, da bomo uporabili programirljiv logični krmilnik podjetja Siemens, mobilni telefon podjetja Apple, usmerjevalnik podjetja Linksys in strežnik. Strežnik smo postavili na virtualno okolje Parallels Desktop for Mac, ki virtualizira strojno opremo za računalnike Macintosh. V nadaljevanju bomo predstavili strojno opremo, ki smo jo uporabili pri razvoju.

2.1 Siemens S7-300 CPU

Programirljiv logični krmilnik (PLK) je digitalni računalnik, namenjen avtomatizaciji procesov, kot so krmiljenje strojev, proizvodne linije itd. Večinoma se uporabljajo v industrijske namene, saj so odpornejši na zunanje dejavnike (to so temperatura, vlaga, vibracije, prah, električne motnje, voda, umazanija). Vgrajenih imajo več izhodov in vhodov, digitalnih in analognih, po potrebi pa lahko vhodne in izhodne module razširimo. Programirljivi logični krmilniki so sistemi, ki tečejo v realnem času.

Programirljiv logični krmilnik je sestavljen iz centralno procesne enote, napajalnika in vhodno-izhodnih modulov (slika 2.1) [1].

Programirljiv logični krmilnik, ki smo ga uporabili za razvoj, spada v



Slika 2.1: Arhitektura krmilnika

skupino standardnih centralno procesnih enot in ima oznako SIMATIC S7-300 CPU 315-2 PN/DP.

2.1.1 Programski cikel

Programski cikel je čas, ki ga krmilnik porabi za celotno zanko procesa od začetka do konca. Navadno traja nekaj milisekund. Čas se od cikla do cikla spreminja, glede na preračunavanje vhodnih in izhodnih parametrov. Na sliki 2.2 je prikazan potek programskega cikla programirljivega logičnega krmilnika. Opisan je v naslednjih točkah [2].

1. Med vsakim programskim ciklom programirljivi logični krmilnik najprej naredi samodiagnozo. Ta vključuje preverjanje napak, spomina in integriranih vezij. S tem programirljivi logični krmilnik preveri, ali centralno procesna enota in povezana vezja delujejo pravilno.

2. Naslednji korak vključuje preverjanje napak na vhodno-izhodnih napravah.

3. Po uspešnem preverjanju, programirljivi logični krmilnik pridobi vhodne podatke vseh priključenih naprav in jih shrani v pomnilniško sliko

vhodov. Med izvajanjem programa se vhodne vrednosti ne spreminjajo.

4. V naslednjem koraku se izvede uporabniški program.

5. Programirljiv logični krmilnik posodobi izhodne vrednosti glede na vhodne vrednosti in ukaze v programu. Izhodne vrednosti shrani v pomnilniško sliko izhodov in preslika v izhodne module. Po tem koraku se celoten postopek ponavlja.

Slika 2.2: Programski cikel PLK.

2.1.2 Komunikacijski protokoli

Komunikacija s krmilnikom poteka prek komunikacijskih protokolov. Krmilniki Siemens podpirajo komunikacijske protokole MPI, Profibus, Profinet in S7.

Vmesnik MPI

Večtočkovni vmesnik (ang. Multi point interface) je komunikacijski protokol krmilnikov SIMATIC S7 v lasti podjetja Siemens. Uporablja se za povezovanje naprav za programiranje krmilnikov, upravljalnih konzol in drugih naprav. Vmesnik MPI temelji na standardu EIA-485 (uradno RS-485) in deluje s hitrostjo 187,5 kb/s [4].

Profibus

Profibus (ang. Process Field Bus) je standard za komunikacijska vodila v avtomatizacijskih procesih. Razvit je bil leta 1987 v Nemčiji. Obstajajo tri različice: Profibus FMS (ang. Fieldbus Message Specification), Profibus DP (ang. Decentralized Periphery) in Profibus PA (ang. Process Automation). Profibus FMS se uporablja za prenos velikih količin podatkov. Danes ga izpodriva industrijski Ethernet. Profinet DP je namenjen povezovanju merilnih in izvršnih členov na programirljivi logični krmilnik. Profibus PA se uporablja za procesne avtomatizacije v nevarnih okoljih [3].

Profinet

Profinet je odprt, mednarodni, industrijski standard, zasnovan na protokolu Ethernet. Obstajata dve različici: Profinet IO in Profinet CBA (ang. Component Based Automation). Profinet IO je bil razvit za vodenje sistemov v realnem času. Profinet CBA je namenjen porazdeljenim sistemom vodenja [3]. Definirane so tri vrste komunikacije: NRT (ang. Non-Real Time), RT (ang. Real Time) in IRT (ang. Isochronous Real Time). Komunikacija NRT za prenos podatkov uporablja standardni protokol TCP/IP, medtem ko ko-

munikacija RT za hitrejši prenos podatkov potrebuje programske razširitve, komunikacija IRT pa razširitve na ravni strojne opreme [3].

Protokol S7

Protokol S7 je interni protokol podjetja Siemens. Uporabljen je v zbirki knjižnic Prodave in omogoča komunikacijo s krmilniki SIMATIC S7. Knjižnice omogočajo s protokolom Ethernet hitrejše in lažje pridobivanje ter pošiljanje podatkov na krmilnik.

2.2 Apple iPhone 3GS

Telefon iPhone 3GS (slika 2.3) je pametni telefon z zaslonom na dotik podjetja Apple. Je telefon tretje generacije pametnih telefonov iPhone. Lastnosti telefona so prikazane v tabeli 2.1.

Slika 2.3: Apple iPhone 3GS. (Vir: [5])

2.3 Usmerjevalnik Linksys WRT-54GL

Usmerjevalnik Linksys WRT-54GL je brezžični usmerjevalnik za domačo rabo ali mikro podjetja. Uporabljen je za internetno povezavo, žično prek

protokola 802.3 Ethernet in brezžično prek 802.11b/g. Linksys je usmerjevalnik predstavil leta 2005, da bi podprl namestitve programske opreme drugih razvijalcev. Usmerjevalnik ima naslednje lastnosti [7]:

- CPU Broadcom BCM5352 @ 200 MHz,
- 16 MB RAM,
- 4 MB spomina Flash,
- 4-portno 10/100Mb stikalo za klasično mrežno povezavo,
- hitrost prenosa podatkov do 54Mb/s,
- strežnik DHCP.

Slika 2.4: Usmerjevalnik Linksys WRT-54GL. (Vir: [8])

Splošno	600 MHz Cortex-A8 CPU
	PowerVR SGX535 GPU
	GSM/EDGE (850, 900, 1800, 1900 MHz)
	UMTS/HSDPA (850, 1900, 2100 MHz)
Velikost in teža	115,5 x 62,1 x 12,3 mm
	135 g
Zaslon	3,5 palčni kapacitivni večdotični zaslon
	16 milijonov barv
	320 x 480 pikslov ločljivosti
	163 pikslov gostote na palec (ang. PPI)
Spomin	8/16/32 GB notranjega spomina
	256 MB RAM
Fotoaparat	Snemanje videa, VGA do 30 slik na sekundo (ang. FPS)
	fotoaparat, 3MP
	Fokus
	dodajanje lokacije videu in fotografijam (ang. geotagging)
Povezovanje	Wi-Fi 802.11 b/g
	Bluetooth 2.1 + EDR
	GPS s podporo A-GPS
Senzorji	Pospeškomer
	Senzor bližine
	Svetlobni senzor

Tabela 2.1: Lastnosti telefona Apple iPhone 3GS [6].

Poglavje 3

Programska oprema

Programski del našega sistema je razdeljen na več področij. Obsega implementacijo mobilne aplikacije za operacijski sistem Apple iOS, upravitelja zahtev ASHX za spletni strežnik Microsoft IIS (ang. Internet Information Services), in kode na krmilniku. V sledečih poglavjih bomo predstavili uporabljene tehnologije, s katerimi smo razvijali programsko opremo.

3.1 Mobilni operacijski sistem iOS

Mobilni operacijski sistem iOS je razvilo podjetje Apple. Prvič je bil predstavljen leta 2007 za mobilni telefon iPhone in predvajalnik glasbe iPod Touch. Sistem izvira iz operacijskega sistema računalnikov Apple (OS X) in temelji na Unixovem operacijskem sistemu. V nasprotju z mobilnimi operacijskimi sistemi Windows Phone podjetja Microsoft in Android podjetja Google, Apple dovoli namestitve samo na svojo strojno opremo.

Uporabniški vmesnik sistema iOS temelji na neposredni manipulaciji z uporabo večdotične tehnologije. Elementi uporabniškega vmesnika so sestavljeni iz drsnikov, stikal in gumbov. Interakcija z OS vključuje kretnje z enim ali več prsti kot so: drsenje, dotikanje in povečevanje.

Knjižnica iOS SDK vsebuje orodja in vmesnike, ki so potrebni za razvoj, namestitev, poganjanje in testiranje aplikacij. Aplikacije so razvite z uporabo

sistemskih zbirk iOS in v programskem jeziku Objective-C. Objective-C je visokonivojski, objektno usmerjen programski jezik, ki programskemu jeziku C doda pošiljanje sporočil v slogu jezika Smalltalk.

Primerjava programske kode jezika C in Objective-C

Klic metode *metoda* na objektu, na katerega kaže kazalec *objekt*, je v programskem jeziku C sledeč:

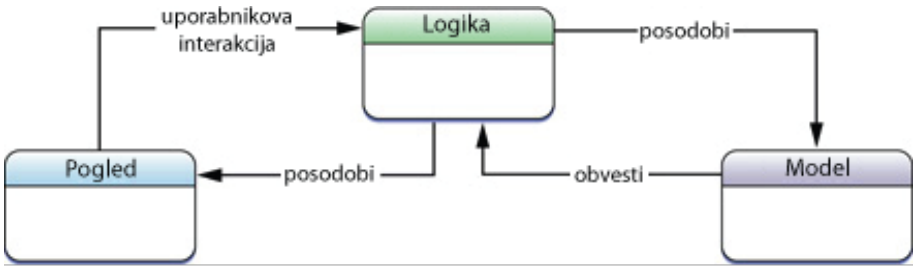
```
objekt->metoda(argument).
```

V programskem jeziku Objective-C to napišemo:

```
[objekt metoda:argument].
```

3.1.1 Arhitektura

Aplikacijska zbirka mobilnega operacijskega sistema iOS se imenuje Cocoa Touch. Zgrajena je na osnovi vzorca model-pogled-logika (ang. Model-View-Controller), ki ga prikazuje slika 3.1.



Slika 3.1: Vzorec MVC.

iOS je sestavljen iz štirih abstraktnih plasti, ki jih lahko vidimo na sliki 3.2. Naslednje točke opisujejo vsebino posamezne plasti [9].

1. Jedro operacijskega sistema (ang. Core OS)

   Tukaj sso jedro sistema, datotečni sistem, omrežna infrastruktura, varnost, upravljanje z energijo, gonilniki.

2. Jedro storitev (ang. Core Services)

   Omogoča manipulacijo nizov, menedžment zbirk, mrežne storitve, nastavitve, menedžment stikov. Zagotavlja storitve GPS, kompas, pospeškomer in žiroskop. Storitve so odvisne od strojne opreme telefona.

3. Mediji (ang. Media)

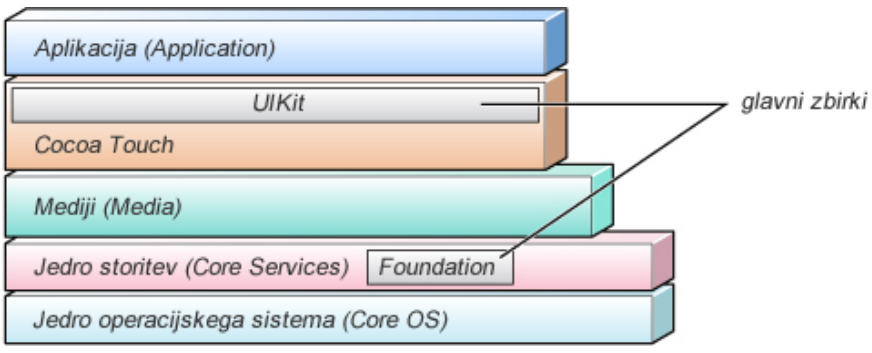
   Zbirke in storitve v tej plasti so odvisne od plasti Core Services. Zagotavljajo grafične in multimedijske storitve plasti Cocoa Touch.

4. Cocoa Touch

   Zbirke v tej plasti neposredno podpirajo aplikacije, ki temeljijo na mobilnem operacijskem sistemu iOS.

Plasti Cocoa Touch in Core Services imata zbirke Objective-C, ki so pomembne za razvijanje aplikacij iOS. Ključne zbirke za iOS so:

- UiKit, ki zagotovi objekte za prikaz v uporabniškem vmesniku in določa strukturo za obnašanje aplikacije, ki vključuje dogodke in risanje,
- Foundation, ki določa obnašanje objektov in vzpostavi mehanizme za njihovo manipulacijo.

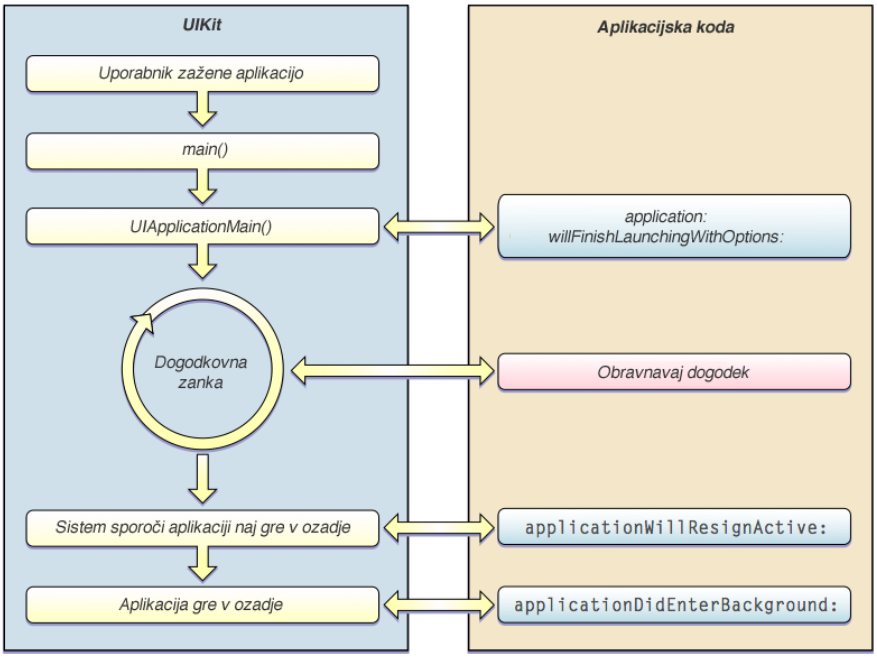


Slika 3.2: Abstraktne plasti mobilnega operacijskega sistema iOS.

3.1.2 Struktura aplikacije iOS

Vsaka aplikacija iOS vsebuje naslednje komponente, ki definirajo njeno strukturo: metodo *main()* v datoteki *main.m*, konfiguracijsko datoteko *Info.plist*, datoteki *AppDelegate.h in AppDelegate.m*, slike in druge datoteke. Slika 3.3 prikazuje življenjski cikel aplikacije. Ob njenem zagonu se zgodijo naslednji koraki [10]:

1. Uporabnik zažene aplikacijo.
2. Zažene se metoda *main()*.
3. Pokliče se metoda *UIApplicationMain()*.
4. Nastane primerek aplikacijskega objekta.
5. Začne se izvajanje aplikacijske kode.



Slika 3.3: Življenjski cikel aplikacije iOS.

3.1.3 Razvojno okolje

Xcode je razvojno okolje za razvijanje aplikacij za mobilni operacijski sistem iOS in operacijski sistem Mac OS. Vključuje vse potrebno za implementacijo aplikacij, urejevalnik programske kode in uporabniškega vmesnika. Med pisanjem kode Xcode prepozna in pokaže sintaktične in logične napake ter ponudi nasvet za njihovo odpravo [11].

3.2 Upravitelj zahtev ASHX

Upravitelj zahtev ASHX spada v skupino upraviteljev zahtev ASP.NET HTTP. ASP.NET je spletno programsko ogrodje, ki ga je razvilo podjetje Microsoft. Omogoča gradnjo dinamičnih spletnih strani, spletnih aplikacij in spletnih servisov [12]. Upravitelji zahtev HTTP so nova tehnologija, ki v klasičnem programskem ogrodju ASP še ni bila uporabljena. V nasprotju s stranmi ASP.NET jih ne pišemo v jeziku HTML, nimajo dogodkov in preostalih podpor. Omogočajo le zapis podatkov v strežniški odgovor HTTP. Upravitelji zahtev ASP.NET imajo končnico .ashx. Ker se izognejo obdelavi, so lažji objekti kot spletne strani [13]. Uporabljamo jih za:

- dinamično generiranje slik,
- dinamično generiranje datotek PDF,
- binarne datoteke,
- performančno kritične spletne strani,
- datoteke XML in
- minimalne spletne strani.

Zgradba upravitelja zahtev ASHX je prikazana na sliki 3.4.

```
<%@ WebHandler Language="C#" Class="Handler" %>

using System;
using System.Web;

public class Handler : IHttpHandler
{
    public void ProcessRequest (HttpContext context)
   {
        context.Response.ContentType = "text/plain";
        context.Response.Write("Hello World");
    }

    public bool IsReusable
   {
        get
        {
            return false;
        }
    }
}
```

Slika 3.4: Zgradba upravitelja zahtev ASHX.

3.3 SIMATIC STEP 7 Professional

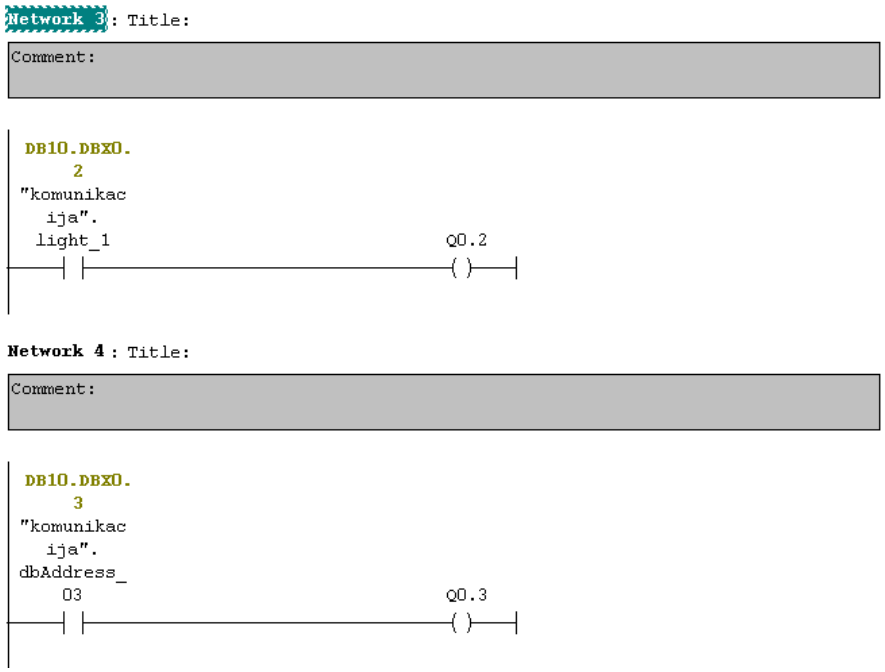
SIMATIC STEP 7 Professional je razvojno okolje podjetja Siemens za krmilnike SIMATIC. Vsebuje orodja za vse stopnje avtomatizacijskega projekta [14]. V sklopu orodij omogoča:

- konfiguracijo in parametrizacijo strojne opreme,
- določevanje vrste komunikacije,
- programiranje,
- testiranje,
- zagon,
- dokumentacijo in arhiviranje,
- spremljanje dogajanja na krmilniku.

Za programiranje so podprti naslednji programski jeziki:

- lestvični diagram (ang. Ladder- LAD),
- funkcijski načrt (ang. Function Block Diagram - FBD),
- lista ukazov (ang. Instruction List - IL).

Za implementacijo smo uporabili lestvične diagrame [15]. Program sestavljamo grafično z dodajanjem gradnikov na prečne lestve. V sklopu implementacije smo definirali 16 lestvičnih diagramov, ki ponazarjajo izhodne kontrolne lučke na krmilniku. Nalogi vsakega lestvičnega diagrama sta preverjanje, ali je spremenljivka na določenem naslovu nastavljena na nič ali na ena, in glee na njeno vrednost vklop ali izklop kontrolne lučke. Primer lestvičnega diagrama je predstavljen na sliki 3.5.



Slika 3.5: Primer lestvičnega diagrama.

Poglavje 4

Strežnik

Strežnik je vmesni člen med mobilno aplikacijo in programirljivim logičnim krmilnikom. Uporabili smo spletni strežnik Microsoft IIS. Njegova naloga je sprejemati zahteve mobilne aplikacije, jih obdelati z upraviteljem zahtev ASHX in posredovati odgovore na prejete zahteve. Za lažjo komunikacijo s krmilnikom smo uporabili knjižnico Libnodave.

4.1 Knjižnica LibNoDave

Odprtokodna knjižnica LibNoDave omogoča izmenjavo podatkov med računalnikom in krmilniki Siemens. Podpira krmilnike serije S7-200/300/400. Knjižnica je napisana v programskem jeziku C++ in lahko komunicira prek protokolov MPI, PPI ali ISO prek TCP. Deluje na sistemih Windows in Linux, na voljo pa je v programskih jezikih: C++, VB, C#, Delphi in Java [16].

Knjižnico smo uporabili za lažjo komunikacijo med strežnikom in krmilnikom. V našem primeru smo uporabili knjižnico, napisano v programskem jeziku C# .

4.2 Struktura podatkov

Želeli smo, da bi bila konfiguracija prostorov in objektov čim enostavnejša, zato v kodi krmilnika nismo definirali nobenih prostorov in objektov v njih. V ta namen smo naredili dva razreda, *DataBlockObject* in *DataBlockObjectsController*.

Razred *DataBlockObject*, prikazan na sliki 4.1, predstavlja objekt v prostoru, na primer, luč v dnevni sobi. Definiran je z devetimi parametri: imenom sobe, identifikacijsko številko sobe, tipom objekta, identifikacijsko številko tipa objekta, imenom, identifikacijsko številko objekta, vrednostjo, ki predstavlja stanje (izklopljen/zaprt, vklopljen/odprt) in pozicijo v prostoru.

```
public class DataBlockObject
{
    private string room;
    private int roomId;
    private string objectType;
    private int objectTypeId;
    private string objectName;
    private int objectId;
    private int value { get; set; }
    private int xPos;
    private int yPos;

    ...
}
```

Slika 4.1: Razred DataBlockObject.

Razred *DataBlockObjectsController*, prikazan na sliki 4.2, je kontrolni razred, v katerem je shranjeno stanje primerkov razreda *DataBlockObject*. Lahko je samo enkrat inicializiran (ang. singleton). Ob prvem klicu parame-

tra *Instance* se inicializira primerek razreda, vsak nadaljnji klic pa vrne isti primerek razreda. Ob inicializaciji se pokliče privzeti konstruktor, v katerem inicializiramo primerke razreda *DataBlockObject*. Mesto dodanega primerka je enako mestu izhoda na krmilniku, ki predstavlja njegovo stanje. Na primer, če pošljemo zahtevo za vklop objekta z imenom *blinds 1*, se bo na krmilniku prižgal tretji izhod.

```
public class DataBlockObjectsController
{
    ...
    private DataBlockObjectsController()
    {
        cache = new ArrayList();

        // livingroom
        cache.Add(new DataBlockObject("livingroom", 0, "light", 0,
                                      "light 1", 0, 160, 100));
        cache.Add(new DataBlockObject("livingroom", 0, "light", 0,
                                      "light 2", 1, 160, 300));
        cache.Add(new DataBlockObject("livingroom", 0, "blinds", 1,
                                      "blinds 1", 2, 80, 10));
       ...
    }
    ...
}
```

Slika 4.2: Razred DataBlockObjectsController

4.3 Implementacija upravitelja zahtev ASHX

Upravitelj zahtev ima naslednje naloge: sprejemanje zahtev strežnika, vzpostavitev povezave s krmilnikom, izvajanje ukazov glede na vrsto zahteve in posredovanje odgovorov o uspešnosti izvedbe zahteve.

4.3.1 Vzpostavitev povezave s krmilnikom

Za vzpostavitev povezave s krmilnikom smo uporabili knjižnico Libnodave. Vsakič, ko upravitelj zahtev prejme zahtevo, se moramo povezati na krmilnik. V nadaljevanju bomo po točkah opisali, kako smo uporabili knjižnico za hitro in enostavno povezavo in komunikacijo s krmilnikom.

1. Definiramo spremenljivke (slika 4.3, vrstice 1-5). Struktura *daveOSserialType* je ovojni tip (ang. wrapper), ki vsebuje spremenljivke, ki predstavljajo prihajajoči in odhajajoči komunikacijski kanal na nivoju operacijskega sistema. Razred *daveInterface* predstavlja žično povezavo do krmilnika ali omrežja krmilnikov, razred *daveConnection* pa predstavlja žično povezavo do enega krmilnika in hrani parametre, vezane na točno določen krmilnik. Spremenljivka *rack* predstavlja letev, na kateri je centralno procesna enota, *slot* pa režo vtičnika.

2. Iinicializiramo in odpremo povezavo TCP/IP do krmilnika (slika 4.3, vrstica 6). Metoda sprejme dva parametra, številko vrat protokola in naslov IP v obliki niza.

3. Inicializiramo razred *daveInterface* (slika 4.3, vrstica 7). Konstruktor sprejme pet parametrov, strukturo daveOSserialType, poljubno ime, naslov vmesnika MPI (v našem primeru ga ne potrebujemo), protokol vmesnika (uporabljamo ISO prek TCP) in hitrost vmesnika (ne potrebujemo, uporabno samo pri protokolih MPI in Profibus).

4. Inicializiramo razred *daveConnection* (slika 4.3, vrstica 8). Konstruktor sprejme štiri parametre, razred *daveInterface*, naslov vmesnika MPI (ga

ne potrebujemo), letev (ang. rack) in režo (ang. slot), na kateri je krmilnik.

5. Preverimo, ali je povezava do krmilnika vzpostavljena (slika 4.3, vrstica 9).

6. Metoda *readBytes* (slika 4.3, vrstica 11) prebere bajte iz podatkovnega bloka krmilnika. Metodo pokličemo s petimi parametri: prvi parameter predstavlja območje branja (beremo iz podatkovnega bloka - DB), drugi številko podatkovnega bloka, tretji začetek prvega bajta v bloku, četrti dolžino bajtov za branje, peti pa tabelo bajtov, v katero naj bo rezultat kopiran. Metoda *writeBytes* (slika 4.3, vrstica 12) deluje podobno, le da bajte zapiše na podatkovni blok na krmilniku. Kot zadnji parameter sprejme tabelo, v kateri so podatki, ki bodo zapisani v podatkovni blok na krmilniku.

7. Z metodo *disconnectPLC* prekinemo povezavo do krmilnika (slika 4.3, vrstica 14).

8. Na koncu prekinemo še povezavo TCP/IP (slika 4.3, vrstica 15).

4.3.2 Vrste zahtev

Strežniki podpirajo več vrst zahtev. V našem primeru potrebujemo samo zahtevi *GET*, za pridobivanje podatkov s strežnika in *PUT* za pošiljanje sprememb in spreminjanje podatkov na strežniku. V nadaljevanju bomo opisali, kaj se zgodi, ko upravitelj zahtev prejme določeno zahtevo.

Pridobivanje podatkov s strežnika

Za pridobivanje podatkov s strežnika mobilna aplikacija pošlje zahtevo *GET* na naslov strežnika. Upravitelj zahtev prejme zahtevo, jo prepozna in pridobi podatke s krmilnika. S podatki posodobi objekte *DataBlockObject*, ki so shranjeni v razredu *DataBlockObjectsController*. Na koncu v odgovor zahteve zapiše objekte *DataBlockObject*.

```
daveOSserialType fds;
daveInterface di;
daveConnection dc;
int rack = 0;
int slot = 2;
fds.rfd = openSocket(102, "192.168.1.2");
di = new daveInterface(fds, "IF1", 0, daveProtoISOTCP, 0);
dc = new daveConnection(di, 0, rack, slot);
if (dc.connectPLC() == 0)
{
        dc.readBytes(daveDB, 10, 0, 2, buffer);
        dc.writeBytes(daveDB, 10, 0, 2, buffer);
}
dc.disconnectPLC();
closeSocket(fds.rfd);
```

Slika 4.3: Koraki za vzpostavitev povezave s krmilnikom.

Spreminjanje podatkov na strežniku

Za spreminjanje podatkov na strežniku mobilna aplikacija pošlje zahtevo *PUT* s slovarjem podatkov na naslov strežnika. Upravitelj zahtev prejme zahtevo, jo prepozna in iz njenega telesa pridobi slovar podatkov. V slovarju so zapisana nova stanja objektov, ki jih je uporabnik prek mobilne aplikacije spremenil. Zatem posodobi vsak objekt *DataBlockObject*, ki ga ima slovar. V naslednjem koraku se vse vrednosti stanj objektov *DataBlockObject* pošljejo na krmilnik, kjer se shranijo. Na koncu, prav tako kot pri zahtevi *GET*, v odgovor zahteve zapiše objekte *DataBlockObject*.

Poglavje 5

Mobilna aplikacija

Mobilno aplikacijo smo razvili na mobilnemu operacijskemu sistemu iOS. Temelji na govornih ukazih, s katerimi posredno upravljamo programirljivi logični krmilnik. Pri razvoju smo uporabili zbirko za prepoznavanje govora OpenEars.

5.1 Zbirka OpenEars

OpenEars je programska zbirka za prepoznavanje in sintezo govora angleškega jezika za operacijski sistem iOS. Uporablja odprtokodne knjižnice *CMU Pocketsphinx, CMU Flite, in CMUCLMTK* in je brezplačna. Zelo natančno in široko besedno prepoznavanje govora (to pomeni, prepoznati katero koli besedo, ki jo uporabnik izgovori, izmed vec tisoč besedami) v aplikaciji in njegova obdelava še nista mogoča zaradi omejitve strojne opreme. Na primer, aplikacija na podprtih napravah iOS za prepoznavanje in sintezo govora Siri, obdela prepoznavanje govora na strežniški strani. Knjižnica Pocketsphinx, ki jo zbirka OpenEars uporablja, je sposobna lokalno prepoznavati besede v slovarjih, ki vsebujejo več sto besed. Z jezikovnimi modeli se odreže zelo dobro, največja prednost pa je, da ne uporablja internetne povezave, ker vsa obdelava poteka na mobilni napravi. Jezikovni model je seznam besed - slovar, za katerega hočemo, da ga zbirka OpenEars prepozna. Manjši in bolj

prilagojen aplikaciji, kot je jezikovni model, večja je natančnost prepoznavanja. Idealen jezikovni model ima manj kot 200 besed [17]. Funkcije, ki jih OpenEars omogoča so:

- neprekinjeno preverjanje prisotnosti govora na ločeni niti v ozadju,
- uporaba devetih glasov, moških in ženskih, z nastavljanjem hitrosti in kakovosti,
- spreminjanje višine, hitrosti sinteze govora,
- sporočanje kateremu koli delu aplikacije rezultat prepoznavanja govora ali sprememb stanja avdio seje (klic),
- dinamično generiranje jezikovnih modelov v aplikaciji iz tabele besed,
- enostavna uporaba z uporabo metod Objective-C.

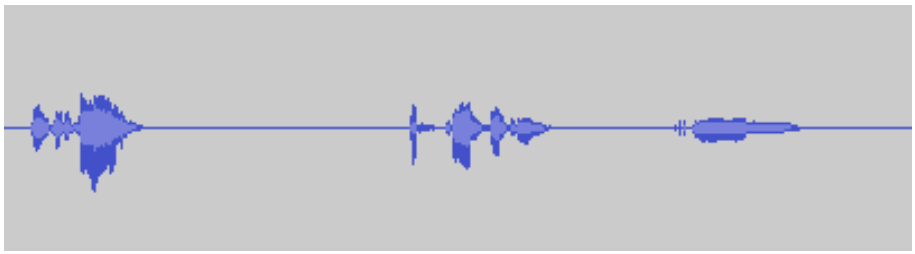
Zbirka OpenEars ima enostavno arhitekturo. Razred *PocketsphinxController* prepoznava govor z jezikovnim modelom, ki ga dinamično tvori razred *LanguageModelGenerator*. Razred *FliteController* ustvari sintetiziran govor. Razred *OpenEarsEventsObserver* obvešča o vsaki funkciji, ki jo je izvedel OpenEars.

5.1.1 Osnovni koncepti govora

Govor je zapleten pojav. Ljudje težko razumemo, kako je ustvarjen in zaznan. Znano je, da je sestavljen iz besed in da je vsaka beseda sestavljena iz fonemov. V teoriji to drži, realnost je pa veliko bolj zapletena. Gre za dinamičen postopek, brez jasno razlikovanih delov. Na sliki 5.1 lahko vidimo, kako je videti posnet govor v avdio urejevalniku.

Struktura govora

Govor je zvezni avdio tok, kjer se stabilna stanja mešajo z dinamično spremenjenimi. V teh sekvencah stanj lahko definiramo bolj ali manj podobne



Slika 5.1: Videz govora v avdio urejevalniku.

razrede zvokov ali fonemov. Besede so sestavljene iz fonemov. Akustične lastnosti zvočnega zapisa, ki predstavlja fonem, se lahko zelo razlikujejo. Odvisne so od sobesedila, govoreče osebe, sloga govora in drugih dejavnikov. Ker so prehodi med besedami bolj informativni kot stabilne regije, razvijalci večkrat govorijo o *difonemih* - delih fonemov med dvema zaporednima fonemoma [18].

Fonemi sestavljajo zloge. To je uporabno pri spremembi hitrosti govora. Takrat se fonemi pogosto spreminjajo, zlogi pa ostanejo enaki.

Zlogi sestavljajo besede. Te so v prepoznavanju govora pomembne, ker zelo omejijo število kombinacij fonemov. Angleščina ima 40 fonemov, povprečna beseda je sestavljena iz sedmih fonemov, torej v grobem obstaja 40^7 besed [19]. Angleško govoreči ljudje redko uporabljajo več kot 20 tisoč besed v svojem vsakdanu, kar naredi prepoznavanje govora veliko bolj izvedljivo.

Govor vključuje tudi *polnila*. To so zvoki, ki jih uporabljamo v pogovoru in dajo sogovorniku vedeti, da smo se ustavili za premišljevanje, ampak nismo še končali govoriti. Primer takih zvokov so: uh, er in um.

Postopek prepoznavanja govora

Postopek prepoznavanja govora v zbirki *OpenEars* je sledeč: zvočni zapis razdeli na dele glede na *polnila* in tišino in nato poskuša prepoznati, kaj je bilo rečeno v določenem delu. To naredi tako, da vzame vse mogoče kombinacije besed in jih primerja z zvočnim zapisom. Na koncu izbere kombinacijo, ki se najbolj ujema. Ujemanje je odvisno od značilk, izbranega modela in postopka

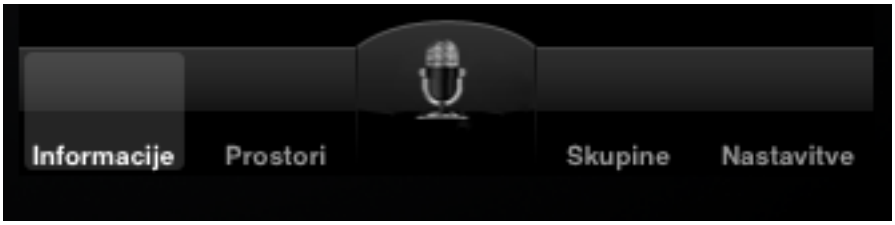
optimizacije. Značilke enostavno opisujejo zvočni zapis. Določijo se tako, da se zvočni zapis razdeli na dele, dolge deset milisekund. Za vsak del, imenu se vektor značilk, se izračuna 39 parametrov, ki ga predstavljajo.

Modeli za prepoznavanje govora

Glede na strukturo govora se uporabljajo trije modeli za njegovo prepoznavanje. *Akustični model* vsebuje akustične lastnosti za vsak fonem. Obstajajo sobesedilno odvisni in sobesedilno neodvisni. *Slovar fonemov* vsebuje povezavo med besedami in fonemi. *Jezikovni model* se uporablja za omejitev iskanih besed in definira, katere besede lahko sledijo prejšnjim prepoznanim besedam [19].

5.2 Struktura aplikacije

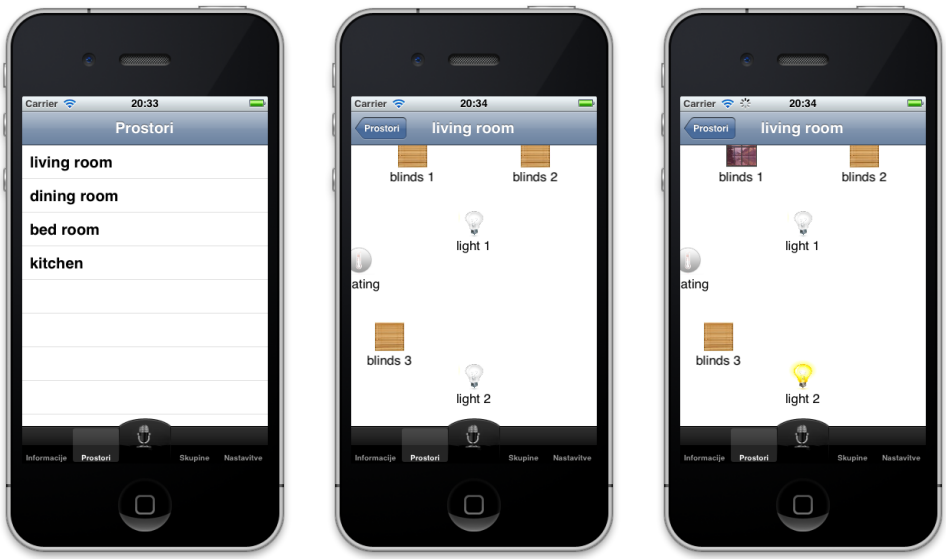
Aplikacijo smo zasnovali tako, da bo hitra, enostavna in intuitivna za uporabo. Zato, so na glavnem oknu aplikacije vedno štirje zavihki in en gumb (slika 5.2). Zavihek *Informacije* prikazuje splošne informacije (število odprtih rolet, luči), zavihek *Prostori* pa omogoča pregled in upravljanje določenega prostora. V tretjem zavihku *Skupine* so objekti razdeljeni na skupine, v četrtem pa so nastavitve. Na sredini je gumb z ikono mikrofona, ki omogoča sprejemanje govornih ukazov. V nadaljevanju si bomo podrobno pogledali funkcije zavihkov *Prostori* in *Skupine* ter gumba za govorne ukaze.



Slika 5.2: Zavihki v aplikaciji.

5.2.1 Prostori

V zavihku *Prostori* imamo pregled nad vsemi prostori v objektu. Pogled je sestavljen iz tabele, v kateri vsaka vrstica ustreza določenemu prostoru (slika 5.3). Uporabnik lahko z izbiro prostora preide v naslednji pogled, ki predstavlja prostor v dvodimenzionalnem pogledu, kjer vidimo objekte v prostoru, postavljene tako, kot so tudi v resničnem prostoru. Objekti so predstavljeni s sliko. Glede na to, ali je objekt odprt ali zaprt se, spreminja tudi njegova slika. Ta pogled uporabniku omogoča tudi interakcijo z objekti v prostoru. Ob dotiku na določen objekt se stanje spremeni. Če je objekt zaprt, ga z dotikom nanj odpremo in obratno, če je odprt, ga zapremo.



Slika 5.3: Zavihek *Prostori*.

5.2.2 Skupine

V zavihku *Skupine* imamo pregled nad skupinami objektov, na primer luči. Pogled sestavlja tabela, v kateri vsaka vrstica ustreza skupini objektov, kar lahko vidimo na sliki 5.4. Uporabnik z izbiro skupine objektov preide v naslednji pogled, ki združuje vse objekte izbrane skupine. Tudi tu vidimo tabelo,

v kateri so objekti razdeljeni po oddelkih. Vsak od njih predstavlja prostor z objekti v njem. Vsak objekt je v svoji vrstici predstavljen z imenom, stanje pa ponazarja drsnik, s katerim je mogoča interakcija (slika 5.4). Drsnik ima poleg standardnih stanj, odprto in zaprto, možnost nastavljati tudi vmesna stanja. To je priročno za nastavljanje zatemnitve luči, položaja rolet in temperature. Pogled ima poleg spreminjanja stanj posameznega objekta tudi možnost vklopa in izklopa vseh objektov v enem ali vseh oddelkih.

Slika 5.4: Zavihek *Skupine*.

5.2.3 Govorni ukazi

Govorne ukaze sprožimo s pritiskom na sredinski gumb z ikono mikrofona. Gumb za govorne ukaze je vedno prikazan na sredini aplikacije, kar omogoča hitro dostopnost. Ob pritisku nanj se začne snemanje zvoka prek vgrajenega mikrofona na telefonu. Med snemanjem aplikacija nenehno preverja, kdaj je uporabnik nehal govoriti. Ko se to zgodi, se snemanje konča in posnetek gre v obdelavo razredu *PocketsphinxController*. Postopek prepoznavanja se časovno razlikuje glede na dolžino posnetega govora. Ob koncu postopka

prepoznavanja razred *PocketsphinxController* sporoči rezultat, ki je predstavljen z nizom. Nato se preveri, ali niz ustreza kakšnemu ukazu. V našem primeru vrstni red besed v nizu ni pomemben, saj smo preverjanje ukazov implementirali tako, da preverjamo, ali niz vsebuje besede nekega ukaza. Ukaz je sestavljen iz:

- glavnega ukaza (na primer open, close, set),
- skupine objektov ali posameznega objekta (na primer lights, light 1),
- prostora (na primer living room) in
- številsko pozitivne vrednosti.

Glavni ukaz in objekti so obvezni, prostor in številsko pozitivna vrednost pa ne. Primer ukaza, ki nastavi temperaturo na 21 v dnevni sobi, je *set temperature in living room to 21*.

Prepoznavanje ukaza

Potek prepoznavanja ukaza smo implementirali na sledeči način (slika 5.5). Najprej preverimo, ali niz vsebuje katerega od glavnih ukazov. Nato preverimo še, ali je v nizu prostor. To storimo zato, ker prostor zmanjša nadaljnje možnosti ukaza. Če ukaz nima prostora, se celoten ukaz navezuje na celo stavbo. V tem primeru pogledamo, ali niz vsebuje kakšen tip objekta. Če obstaja, pridobimo vse objekte tega tipa in jih dodamo v tabelo. Če ukaz vsebuje prostor, najprej preverimo, ali niz vsebuje kakšen tip objekta. Če ga ima, pridobimo vse objekte tega tipa in jih dodamo v tabelo, v nasprotnem primeru pa pogledamo, ali ima niz v prostoru objekt. Nato preverimo, ali imamo v tabeli objektov kakšen element in v tem primeru moramo preveriti tip ukaza. Glede na tip ukaza nastavimo vrednost elementu ali elementom v tabeli. Po tem koraku sledi priprava elementov v tabeli za pošiljanje na strežnik.

```
tabela_objektov;
vrednost = -1;
tip_ukaza = poišči UkazVNizu();
if (tip_ukaza){
    tip_prostora = poišči ProstorVNizu();
    if (tip_prostora == brez_prostora){
        tip_objekta = poišči TipObjektaVNizu();
        if (tip_objekta){
            objekti = poišči(tip_objekta, brez_prostora);
            dodaj objekti v tabela_objektov;
        }
    }else{
        tip_objekta = poišči TipObjektaVNizu();
        if (tip_objekta){
            objekti = poišči(tip_objekta, tip_prostora);
            dodaj objekti v tabela_objektov;
        }else{
            objekt = poišči ObjektVNizu(tip_prostora.objekti);
            dodaj objekt v tabela_objektov;
        }
    }
    if (tabela_objektov.count() > 0){
        if (tip_ukaza == "OPEN"){
            vrednost = 1;
        }else if (tip_ukaza == "CLOSE"){
            vrednost = 0;
        }else{
            vrednost = poišči VrednostVNizu();
        }
        if (vrednost >= 0){
            nastaviVrednost(tabela_objektov, vrednost);
            pripraviObjekte(tabela_objektov);
        }
    }
}
```

Slika 5.5: Postopek prepoznavanja ukaza, prikaz v psevdokodi.

Pošiljanje zahteve na strežnik

Zahteva, ki pošlje podatke na strežnik, vsebuje eno ali več identifikacijskih številk objektov in vrednosti, na katero naj se objekti nastavijo.

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi smo razvili pilotski sistem za upravljanje elementov hišne avtomatizacije s pametnim telefonom. Upravljanje smo uporabnikom dodatno olajšali z uporabo govornih ukazov. Spoznali smo sestavo in delovanje programirljivega logičnega krmilnika ter predstavili mobilni operacijski sistem iOS, za katerega smo razvili mobilno aplikacijo. Na splošno smo opisali, kaj je govor, njegovo sestavo in sam postopek prepoznavanja govora s knjižnico OpenEars.

Celoten sistem deluje dobro, šibka točka je prepoznavanje govornih ukazov, ki je povprečno. Definitivno aplikacija še ni primerna za produkcijo, saj je verjetnost napak prevelika. V primerjavi z drugimi aplikacijami za prepoznavanje govora, Siri podjetja Apple ali Evi podjetja Google, knjižnica OpenEars niše na njihovi ravni uporabnosti in zanesljivosti. Uporabniki bi se ob velikem številu napačnih prepoznavanj ukazov razjezili in se tega naveličali, s tem pa bi uporabnika odvrnili od uporabe aplikacije. Pri izboljšavi je treba najprej zagotoviti boljše in zanesljivejše prepoznavanje ukazov uporabnika.

Na področju strojne opreme bi lahko na krmilnik priključili različne izhodne enote (na primer luči, rolete, termostate), ki bi delovale glede na ukaze, prejete prek mobilne aplikacije. Potencial celotnega sistema je velik, uporaben je tudi v praksi, izboljšati je treba le prepoznavanje govornih ukazov.

Literatura

[1] H. Berger. "SIMATIC Controllers - the Hardware Platform", v *Automating with SIMATIC: Integrated Automation with SIMATIC S7-300/400*. Publicis MCD Verlag, Erlangen and Munich, 2000.

[2] (2012) How PLCs Work. Dostopno na `http://www.machine-information-systems.com/How_PLCs_Work.html`.

[3] U. Lotrič. *Predavanja pri predmetu Procesna avtomatika*, 2011. Dostopno na: `https://ucilnica1112.fri.uni-lj.si/course/view.php?id=53`.

[4] (2012) Multi-Point Interface - Wikipedia, the free encyclopedia. Dostopno na: `http://en.wikipedia.org/wiki/Multi-Point_Interface`.

[5] (2012) apple-iphone-3g. Dostopno na: `http://www.thegeeksclub.com/wp-content/uploads/2011/07/apple-iphone-3g.png`.

[6] (2012) Apple iPhone 3GS - Full phone specifications. Dostopno na: `http://www.gsmarena.com/apple_iphone_3gs-2826.php`.

[7] (2012) Linksys WRT54G series - Wikipedia, the free encyclopedia. Dostopno na `http://en.wikipedia.org/wiki/Linksys_WRT54G_series#WRT54GL`.

[8] (2012) brezzicni_router_linksys_wrt54gl. Dostopno na: `http://www.ttc.si/iimg/2874/brezzicni_router_linksys_wrt54gl.jpg`.

[9] (2012) Cocoa Fundamentals Guide: What Is Cocoa?. Dostopno na: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html.

[10] (2012) iOS App Programming Guide: App States and Multitasking. Dostopno na: http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html#//apple_ref/doc/uid/TP40007072-CH4-SW3.

[11] (2012) What's New in Xcode - Developer Tools Technology Overview - Apple Developer. Dostopno na: https://developer.apple.com/technologies/tools/whats-new.html.

[12] (2012) ASP.NET - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/ASP.NET.

[13] (2012) HTTP handler - Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/HTTP_handler.

[14] (2012) STEP 7 Professional - Software for SIMATIC Controllers - Siemens. Dostopno na: http://www.automation.siemens.com/mcms/simatic-controller-software/en/step7/step7-professional/Pages/Default.aspx.

[15] J. R. Hackworth, F. D. Hackworth Jr. "Ladder Diagram Fundamentals", v *Programmable Logic Controllers: Programming Methods and Applications*. Prentice Hall, 2003.

[16] (2012) LIBNODAVE, a free communication library for Simatic S7 PLCs. Dostopno na: http://libnodave.sourceforge.net/.

[17] (2012) OpenEars - iPhone Voice Recognition and Text-To-Speech. Dostopno na: http://www.politepix.com/openears/.

[18] J. Holmes, W. Holmes. "Human Speech Communication", v *Speech Synthesis and recognition 2nd Edition*. CRC Press, 2001.

[19] (2012) Basic concepts of speech - CMUSphinx Wiki. Dostopno na: `http://cmusphinx.sourceforge.net/wiki/tutorialconcepts`.