SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1795

Glasovno upravljanje vanjskim uređajem pomoću osobnog računala

Iva Pavić

Zagreb, srpanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 6. ožujka 2019.

DIPLOMSKI ZADATAK br. 1795

Pristupnik: **Iva Pavić (0036468700)**
Studij: Elektrotehnika i informacijska tehnologija
Profil: Elektroničko i računalno inženjerstvo

Zadatak: **Glasovno upravljanje vanjskim uređajem pomoću osobnog računala**

Opis zadatka:

U okviru diplomskog rada potrebno je istražiti mogućnosti razvoja aplikacije za upravljanje vanjskim uređajem korištenjem glasa, tj. putem govornih komandi. Aplikacija treba biti razvijena za osobno računalno s operativnim sustavom Microsoft Windows10. Aplikacija treba prepoznati upravljačke riječi izgovorene na engleskom jeziku iz ograničenog skupa naredbi (oko dvadesetak riječi s fiksno odabranom sintaksom rečenice). Koristiti raspoloživu Microsoft tehnologiju za obradu i prepoznavanje glasa, te odabrati rješenje podržano u operacijskom sustavu Windows10. Razviti aplikaciju u C++, MSDN, .NET okruženju, koja govorne naredbe pretvara u tekstualni oblik i rezultat prepoznavanja prikazuje u kontrolnom ekranu. U svrhu demonstracije mogućnosti upravljanja vanjskog uređaja korištenjem razvijene aplikacije, potrebno je realizirati jednostavni ugradbeni računalni sustav temeljen na razvojnom sustavu za STM32F0 mikrokontroler koji će prepoznati tekst s osobnog računala prihvatiti preko serijskog UART sučelja ostvarenog preko USB veze i pripadajućeg kontrolera, te izvršenje odgovarajuće naredbe vizualizirati pomoću led prikaznika ovog ugradbenog sustava.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:
Prof. dr. sc. Davor Petrinović

Djelovođa:
Doc. dr. sc. Tomislav Petković

Predsjednik odbora za diplomski rad profila:
Prof. dr. sc. Mladen Vučić

SADRŽAJ

1. Uvod

Posljednjih godina javlja se sve veća raširenost upravljanja elektroničkim uređajima pomoću glasa. Takozvani virtualni pomoćnici izvršavaju zadatke koje korisnik daje govorom zamjenjujući time ručno upravljanje uređajima s glasovnim.

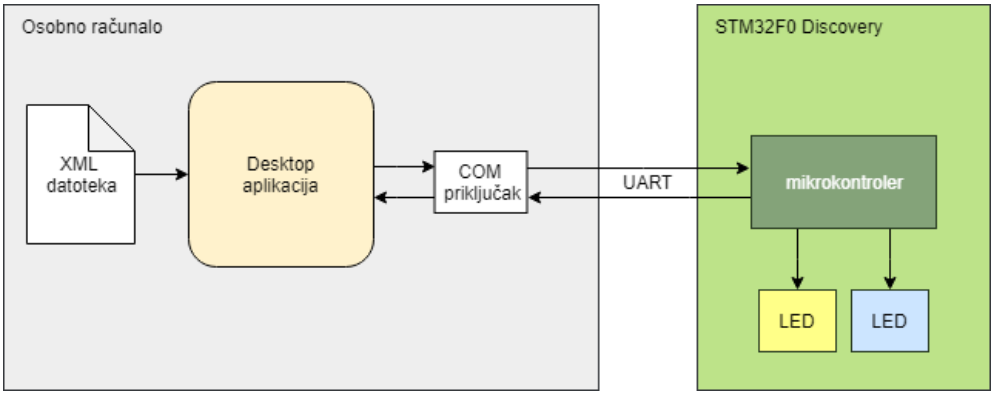
Cilj ovog diplomskog zadatka je bio razviti aplikaciju kojom bi se glasovno upravljalo vanjskim uređajem koristeći tehnologiju prepoznavanja govora koja je dostupna na svakom računalu s Windows® 10 operativnim sustavom. Kako bi se ovo realiziralo bilo je potrebno istražiti Microsoft tehnologiju prepoznavanja govora, implementirati ju te razviti programsku podršku za upravljanje vanjskim uređajem.

Dok je u ovom radu glasovno upravljanje demonstrirano paljenjem svjetlećih dioda, nadogradnjom sustava bilo bi moguće glasovnim upravljanjem upravljati pozicijom pametnog stola što je bila i motivacija izrade ovog rada. Budući da se pametni stolovi najčešće nalaze u uredima, time bi se svakom korisniku računala s Windows® 10 operativnim sustavom jednom aplikacijom omogućilo glasovno upravljanje stolom na kojem se računalo nalazi.

U prvom su poglavlju opisana razvojna okruženja i tehnologije korištene u izradi ovog diplomskog rada i cjelokupna arhitektura sustava. U drugom poglavlju se definira prepoznavanje i sinteza govora i detaljnije se opisuje tehnologija prepoznavanja govora korištena u izradi rada. U trećem poglavlju detaljno je opisana razvijena desktop aplikacija i njene funkcionalnosti. Opisana je implementirana serijska komunikacija aplikacije. U četvrtom poglavlju objašnjena je struktura razvijene programske podrške za mikrokontroler i svaki od slojeva programske podrške. U zadnjem poglavlju dane su upute za korisnike, dan je primjer rada sustava i osvrnulo se na perfomanse sustave i moguća poboljšanja sustava.

2. Arhitektura sustava

Kroz ovo poglavlje prikazana je arhitektura sustava i opisana su razvojna okruženja i programski jezici korišteni za izradu desktop aplikacije i programske podrške za mikrokontroler.



Slika 2.1: Arhitektura sustava

Na slici 2.1 prikazana je arhitektura razvijenog sustava. Sustav čini osobno računalo i STM32F0 Discovery pločica koji komuniciraju serijskom vezom, odnosno UART-om (eng. *universal asynchronous receiver-transmitter*). Korištenjem naredbi iz XML datoteke desktop aplikacija izgrađuje rječnik koja joj omogućava prepoznavanje određenog skupa riječi. Prepoznate i potvrđene naredbe se šalju COM priključkom mikrokontroleru koji obrađuje podatke, šalje potvrdu primitka i ovisno o naredbi pali jednu od svjetlećih dioda odgovarajućom frekvencijom.

2.1. Desktop aplikacija

Desktop aplikacija razvijana je u Visual Studio Community 2017 razvojnom okruženju koristeći C++/CLI programski jezik i .NET razvojni okvir. Datoteka u kojem se nalaze naredbe je razvijena koristeći XML jezik.

2.1.1. C++/CLI

Kako bi se opisao C++/CLI programski jezik prvo je potrebno definirati C++ programski jezik. C++ je objektno orijentirani programski jezik opće namjene razvijen kao proširenje C programskog jezika. Kao takav, u njemu je uz objektno programiranje, moguće i proceduralno programiranje. Koristi se u cijelom nizu područja, a najviše u razvoju aplikacija, web preglednika i prevoditelja.

C++/CLI je programski jezik koji je razvio Microsoft kako bi omogućio korištenje C++ programskog jezika unutar zajedničke jezične infrastrukture (CLI, eng. *Common Language Infrastructure*) prema kojem je razvijen i .NET razvojni okvir. Korištenjem .NET razvojnog okvira omogućuje se upravljanje i izvršavanje aplikacija time omogućujući razvoj aplikacija C++ programerima. Kod napisan u C++/CLI jeziku je zapravo modificirani C++ kod u kojem se promijenjeni dijelovi koda, nazvani upravljani kod (eng. *managed code*), prevode i zatim izvršavaju unutar CLR-a, virtualnog stroja unutar .NET razvojnog okvira. Ostali nepromijenjeni dijelovi programa, odnosno neupravljani kod, se izvršavaju unutar C++ Runtime-a.

2.1.2. .NET

.NET razvojni okvir (eng. *.NET Framework*) je platforma otvorenog koda koja omogućava razvoj različitih aplikacija. Okvir uključuje ranije spomenuti CLR, razne programske jezike kao što su C#, F#, Visual Basic i C++/CLI te velik broj biblioteka klasa. CLR (eng. *Common Language Runtime*) je virtualni stroj u kojem se izvršava strojni kod i pruža razne usluge uključujući upravljanje memorijom, *garbage collection*, upravljanje iznimkama i upravljanje dretvama. Kako bi CLR mogao izvršiti strojni kod, programski kod se prvo mora prevesti u strojni. Programski kodovi, napisani u različitim jezicima za .NET razvojni okvir, se svaki preko svog prevoditelja prevode u CIL (eng. *Common Intermediate Language*) kod. Zatim CLR pomoću JIT (eng. *Just In Time*) prevoditelja prevodi CIL kod u izvorni strojni kod računala i izvršava ga.

U nastavku su navedene neke od biblioteka klasa .NET razvojnog okvira koje su korištene u izvršavanju ovog diplomskog zadatka.

System Speech biblioteka klasa sadrži imenske prostore (eng. *namespace*) koji omogućavaju korištenje mehanizama obrade govora u aplikacijama koje koriste upravljani kod. Više o ovoj biblioteci i pripadnim imenskim prostorima bit će riječ u kasnijim poglavljima.

Windows Forms je grafička biblioteka klasa koja omogućava izradu desktop aplikacija u grafičkom korisničkom sučelju. Windows forma je vizualni prostor na

kojem se nalaze elementi za prikaz ili unos podataka i omogućava komunikaciju između aplikacije i korisnika.

XML biblioteka klasa se nalazi unutar standardnih biblioteka zajedničke jezične infrastrukture i omogućava obradu podataka XML dokumenata.

2.1.3. XML

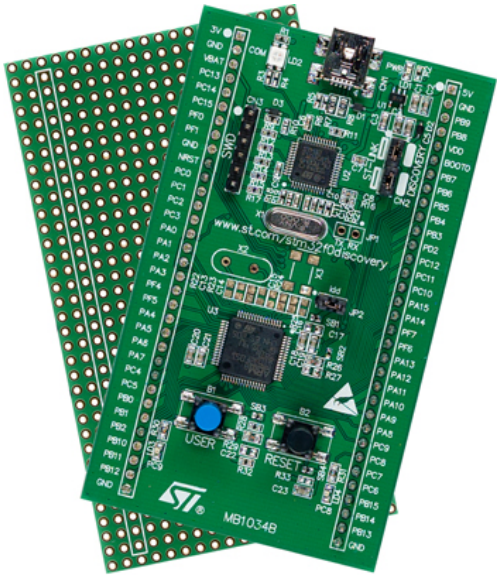
XML (eng. *Extensible Markup Language*) je proširiv jezik za označavanje podataka koji omogućuje stvaranje neograničenog broja elemenata. Podaci u XML dokumentu se zapisuju u tekstualnom obliku i njihovo prikazivanje je neovisno o operacijskom sustavu i programskoj podršci. Iz tog razloga se XML još opisuje i kao "tekst format". Glavna gradivna jedinica XML dokumenta je element koji je definiran oznakama. Element čine početna i završna oznaka i sadržaj između njih. Svi elementi unutar XML dokumenta sadržani su unutar jednog korijenskog elementa, a svaki pojedini elementi može sadržavati podelemente pri čemu se dobiva hijerarhijska struktura podataka. Pri stvaranju XML dokumenta potrebno je držati se određenog skupa pravila kako bi dokument bio "dobro formiran" odnosno da bi ga XML *parser* mogao obraditi. Skup pravila uključuje zahtjev za samo jednim korijenskim elementom i pravilna upotreba oznaka kod svakog elementa. [1]

2.2. Programska podrška za mikrokontroler

Programska podrška za mikrokontroler je razvijana u Eclipse razvojnom okruženju koristeći C programski jezik. Tijekom razvoja korišten je STM32F0 Discovery razvojni sustav.

2.2.1. STM32F0 Discovery

STM32F0 Discovery je razvojni sustav napravljen za STM32F051R8 mikrontroler s 32-bitnom ARM Cortex-M0 jezgrom radne frekvencije do 48 Mhz. Mikrokontroler sadrži 64 KB *flash* memorije i 8 KB RAM-a, a najviša radna frekvencija mu je 48 Mhz. Uz I2C, USART i SPI komunikacijska sučelja, mikrokontroler sadrži AD i DA pretvornik, brojila, RTC i DMA kontroler kao i izvode opće namjene. Sam razvojni sustav, uz tipku za resetiranje i svjetleće diode za indikaciju USB komunikacije i napajanja, sadrži dvije svjetleće diode i jednu tipku čije namjene je moguće programirati. Discovery pločica na kojem se nalazi razvojni sustav je napajana s 3.3V preko USB sabirnice koja



Slika 2.2: STM32F0 Discovery pločica

je povezana s osobnim računalom, a sam mikrokontroler napaja pločica. [2]

Pri razvoju u Eclipse razvojnom okruženju korišten je System Workbench za STM32 - Bare Metal Edition, platforma koja olakšava razvoj programske podrške za STM32 razvojne sustave. Unutar platforme se međuostalim nalaze uređivač koda (eng. *code editor*), alati za prevođenje (prevoditelj, asembler, linker), alati za traženje i otklanjanje grešaka u kodu (eng. *debugging*) i alati za programiranje mikrokontrolera (eng. *flash programming*). [3] Nakon razvijene programske podrške, podaci se šalju putem USB sabirnice na ST-LINK/V2 programator koji je ugrađen na samoj Discovery pločici i koji na temelju primljenih podataka programira mikrokontroler.

3. Prepoznavanje i sinteza govora

Od rane povijesti, govor je bio glavni način komuniciranja između ljudi. Komunikacija govorom, odnosno verbalna komunikacija, je zapravo proces razmjene informacija gdje svaki sugovornik ima sposobnost ljudskog govora i razumijevanja ljudskog govora. Danas se tu komunikaciju nastoji prenijeti i na komunikaciju između čovjeka i računala gdje se razumijevanje ljudskog govora implementira postupcima prepoznavanja govora, a simulacija ljudskog govora postupcima sinteze govora. U pozadini ovih postupaka je digitalna obrada govora i u ovom radu će se prepoznavanje i sinteza govora skupno nazivati obradom govora.

Prepoznavanje govora (eng. *speech recognition*) je interdisciplinarno područje koje se bavi razvojem metoda i tehnologija kojima računalo može obraditi govor. Kombinirajući područja jezikoslovlja, računarske znanosti i elektrotehnike omogućava se računalno prepoznavanje i razumijevanje govora, a posljedično i prevođenje istoga u tekst. Prepoznavanje govora se zato još naziva i *speech-to-text* i koristi se u situacijama kada korisnik nije u mogućnosti ili ne želi ručno pisati ili upravljati uređajima. Prepoznavanje govora se ne smije zamijeniti s prepoznavanjem glasa. Iako je često implementirano unutar sustava za prepoznavanje govora, prepoznavanje glasa (eng. *voice recognition*) se odnosi na identifikaciju govornika, a ne samog govora.

S druge strane, sinteza govora (eng. *speech synthesis*) je zapravo suprotnost prepoznavanja govora u smislu da se ljudski govor simulira. Govor se zapravo sintetizira iz teksta pa se zato sinteza govora još naziva i *text-to-speech*. Zbog toga sinteza govora veliku primjenu nalazi u pomoći osobama s oštećenjem vida ili poteškoćama u čitanju, a danas i u raznim glasovnim virtualnim asistentima.

Unutar Windows® operativnog sustava nalazi se SAPI, tehnologija koja omogućava korištenje prepoznavanja i sinteze govora svakom korisniku Windowsa kao i svakom razvojnom inženjeru za implementaciju u svojim aplikacijama.

3.1. SAPI

Godinama prije trenutno popularnih tehnologija prepoznavanja i sintetiziranja govora poput Sirija i Alexe, Microsoft je vidio potencijal u tehnologijama vezanih uz govor. 1993. godine Microsoft započinje s razvojem tehnologija prepoznavanja govora zapošljavanjem dio tima ljudi koji su razvili Sphinx-II sustav za prepoznavanje govora, tada smatran najnaprednijim sustavom za prepoznavanjem govora. Cilj tima je bio stvoriti tehnologiju koja bi bila precizna, ali i dostupna razvojnim inženjerima preko naprednog API-ja, nazvanog Speech API (SAPI). [4]

SAPI (eng. *Speech Application Programming Interface* je sučelje koje omogućava razvojnim inženjerima služenje gotovim funkcionalnostima mehanizama obrade govora (eng. *speech engines*) unutar Windows® operativnog sustava. Spomenuti mehanizmi obrade govora su prepoznavanje i sinteza govora.

Prije Windows Vista® operativnog sustava, Microsoft tehnologije vezane uz govor bile su dostupne razvojnim inženjerima pomoću odvojenog Speech skupa razvojnih alata (eng. *Software Development Kit*). Datoteke koje su omogućavale navedene tehnologije morale su se nalaziti uz svaku razvijenu aplikaciju i biti posebno instalirane na računalo korisnika kako bi se omogućilo prepoznavanje ili sinteza govora. U Windows Vista® operativnom sustavu prepoznavanje govora postaje integrirano u sam operativni sustav kao Windows Speech Recognition, a Speech skup razvojnih alata i SAPI postaju dio Windows skupa razvojnih alata čime je osigurano da sam operativni sustav omogućuje upotrebu tehnologije vezanih uz govor bilo kojoj razvijenoj aplikaciji. [5]

3.2. System Speech biblioteka

Unutar Speech skupa razvojnih alata nalazi se System Speech biblioteka koja komunicira s mehanizmima obrade govora pozivom preko SAPI-ja (sapi.dll) što znači da razvijene aplikacije i mehanizmi obrade govora ne komuniciraju direktno čime se potpuno razdvojilo aplikacije i mehanizme obrade govora i učinilo aplikacije manje ovisnom o samim mehanizmima. [5] Točnije, aplikacije pomoću SAPI-ja pozivaju željene naredbe, *runtime* DLL prevodi i obrađuje te iste naredbe, a zatim pomoću sučelja šalje potrebne pozive mehanizmima za obradu govora koji naposljetku izvršavaju željene naredbe. Isto tako, mehanizmi za obradu i sintezu govora generiraju događaje (eng. *event*) tijekom izvođenja pa je dijagram toka suprotan, kreće od mehanizama koji generiraju događaje, prolazi kroz runtime DLL i naposljetku dolazi u aplikaciju gdje se svaki događaj obrađuje u rukovatelju događaja (eng. *event handler*)

koji je pridjeljen za podignuti događaj (eng. *raised event*).

Budući da je u .NET razvojni okvir uključen Speech skup razvojnih alata, odnosno Windows skup razvojnih alata, razvojem aplikacija u .NET razvojnom okviru omogućeno je korištenje tehnologija obrade govora što je iskorišteno u ovom radu.

System Speech biblioteka koristi desktop verziju SAPI-ja i dizajnirana je da prima zvuk više kvalitete i osim naredbi i glasovnog upravljanja primjenjuje se i u diktiranju teksta. Kao dio Windows® operativnog sustava, System Speech biblioteka je instalirana je na svakom računalu s Windows Vista® operativnim sustavom ili novijim. Dobro je spomenuti da postoji i serverska verzija SAPI-ja, Microsoft Speech, koja prima telefonsku kvalitetu zvuka i potrebno ju je preuzeti s interneta i instalirati na računalo. [6] Kvaliteta zvuka i činjenica da se System Speech biblioteka može koristiti lokalno su razlozi odabira System Speech biblioteke u ovom diplomskom radu. Za upotrebu funkcionalnosti prepoznavanja govora potrebno je treniranje funkcionalnosti za svakog korisnika gdje se uzima u obzir izgovor korisnika. Ova biblioteka za prepoznavanje govora koristi rječnik unutar sustava za neograničeno diktiranje ili napravljene rječnike koji se stvaraju ovisno o svrsi aplikacije. [7] Samim mehanizmima za obradu govora pristupa se putem pripadajućih imenskih prostora, međuostalim System Speech Recogniton i System Speech Synthesis koji su korišteni i u ovom diplomskom radu.

3.3. Windows Speech Recognition

Windows Speech Recognition (u daljnjem tekstu WSR) je komponenta razvijena od strane Microsofta za Windows Vista® operativni sustav dostupna korisnicima od 2007. godine. [4] WSR omogućava glasovno upravljanje računalom i aplikacijama unutar sustava bez upotrebe tipkovnice ili miša. Koristeći glasovne naredbe, korisnik može, primjerice, diktirati e-mail, otvoriti željenu aplikaciju ili klikati mišem. Cijeli popis glasovnih naredbi dostupan je na službenoj Microsoft stranici za podršku korisnicima. [8]

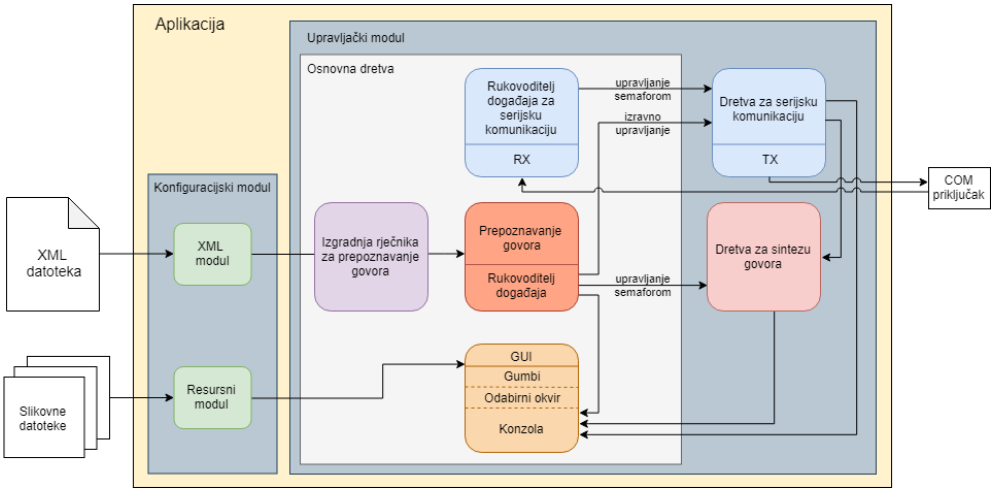
WSR stvara lokalni profil prepoznavanja govora kako bi spremio informacije o prepoznavanju glasa korisnika. Profil je moguće promijeniti tako da se prepoznaje drugi glas ili isti glas u drugačijem okruženju. Text to Speech komponenta podržava dva glasa, muški (David) i ženski (Zira). Jezici koje podržava dani su na službenim Microsoft stranicama [9]. Napravljen profil koriste i aplikacije razvijene koristeći System Speech bibilioteku, a neke informacije profila kao što je odabir glasa, brzina i glasnoća sintetiziranog govora je moguće i programski izmijeniti za pojedinu aplikaciju.

Iako među ranijim sustavima za prepoznavanje govora, Microsoft je u novije doba prestao s razvojem tehnologije za prepoznavanje govora pa je tako WSR u Windows 10® operativnom sustavu ostao više manje nepromijenjen u odnosu na verziju u Windows Vista® operativnom sustavu iz 2007. godine. Kad je riječ o točnosti, WSR je rangiran prilično nisko na ljestvici sustava za prepoznavanje govora. Bez treniranja, prilikom diktiranja, WSR ima točnost od 93.6% na uzorku od 1,028 riječi što je manja točnost od konkuretnih softvera. Microsoft s druge strane tvrdi da uz treniranje, WSR postiže točnost od 99%. [10]. Uz treniranje, na uzorku od 314 riječi ubrajajući u to i rečenične znakove WSR postiže točnost od 97.8% [11] što je i dalje lošije od konkuretnih sustava.

4. Razvoj desktop aplikacije

4.1. Arhitektura aplikacije

Arhitektura razvijene aplikacije sastoji se od dva glavna modula kao što je prikazano na slici 4.1.



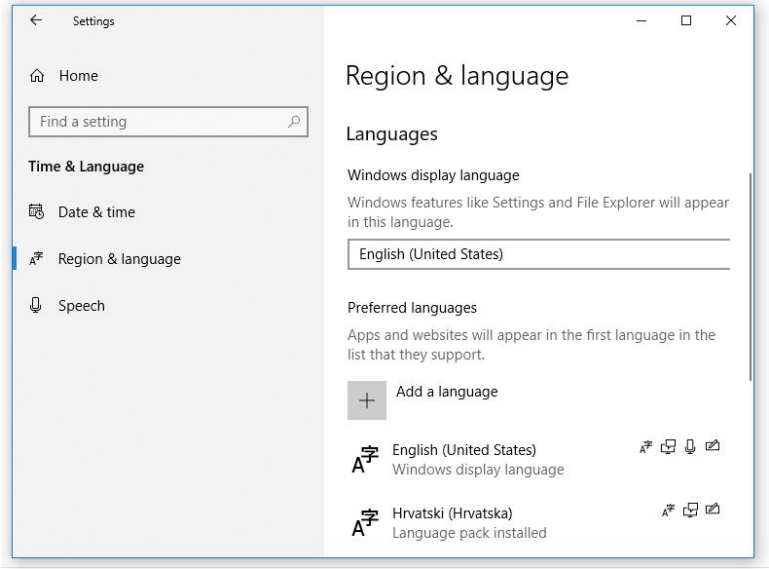
Slika 4.1: Arhitektura aplikacije

Konfiguracijski modul sastoji se od XML modula, koji obrađuje vanjsku XML datoteku i omogućava gradnju rječnika unutar upravljačkog modula, i resursnog modula koji pruža slikovne elemente unutar grafičkog korisničkog sučelja aplikacije. Unutar upravljačkog modula postoje tri dretve; osnovna dretva, dretva za serijsku komunikaciju i dretva za sintezu govora. Građenje rječnika, prepoznavanje govora i grafičko sučelje implementirano je u osnovnoj dretvi. Iako postoji dretva za serijsku komunikaciju, primanje i obrada podataka je implementirano unutar osnovne dretve dok je slanje podataka implementirano u samoj dretvi za serijsku komunikaciju. Obje komponente serijske komunikaciju imaju pristup i komuniciraju s COM priključkom koji omogućuje slanje i primanje podataka prema vanjskom uređaju.

U idućim poglavljima detaljnije su objašnjeni svi elementi arhitekture aplikacije.

4.2. Konfiguriranje Windows Speech Recognition značajke

Prilikom razvoja i korištenja aplikacije aplikacije potrebno je imati operativni sustav Windows 10® i konfiguriran Windows Speech Recognition značajku koja je dostupna unutar operativnog sustava. Upute o konfiguraciji dostupne su na službenoj Microsoft stranici za podršku korisnicima [12]. Za uspješno korištenje prepoznavanja govora potrebno je imati instaliran jedan od ponuđenih paketa prepoznavanja govora za engleski jezik. Aplikacija je razvijana s instaliranim paketom za prepoznavanje govora na američkoj inačici engleskog jezika. Prilikom konfiguracije značajke prepoznavanja



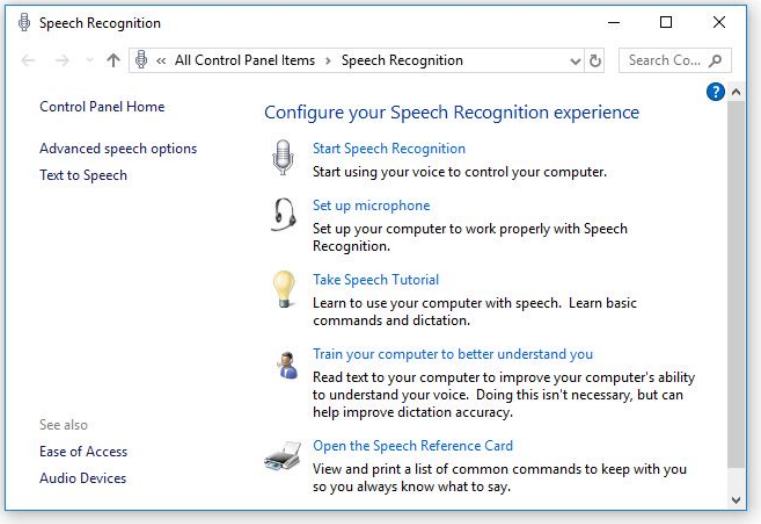
Slika 4.2: Instaliran potreban paket za prepoznavanje govora

govora, softver je kratko treniran kako bi se postigao što točniji glasovni profil. Tijekom razvoja aplikacije korištene su slušalice s mikrofonom kako bi se maksimalno eliminirao pozadinski šum odnosno da bi se postiglo preciznije prepoznavanje govora.

Na slici 4.3 nalazi se upravljačka ploča Windowsa 10® gdje je moguće konfigurirati, trenirati i pokrenuti WSR.

4.3. Postavljanje razvojnog okruženja

Nakon što su napravljeni koraci u prijašnjem odlomku potrebno je postaviti razvojno okruženje za System Speech biblioteku u Visual Studio razvojnom okruženju. Potrebno je preuzeti i instalirati Microsoft .NET Framework u kojem se nalaze System Speech



Slika 4.3: Windows Speech Recognition konfiguracija

imenski prostori koji razvojnim inženjerima omogućuju pristup tehnologijama prepoznavanja i sinteze govora unutar Windows® operativnog sustava. Za razvoj aplikacije u ovom radu korišten je .NET 4.6.1 Framework. Zatim se unutar Visual Studio projekta treba dodati referenca na System.Speech, a na početku koda treba uključiti System Speech biblioteku. Po izboru se mogu dodati *using* naredbi kojima se omogućava korištenje klasa unutar Speech Recognition i Speech Synthesis imenskih prostora. Ukoliko se ne dodaju *using* naredbe, pri svakom korištenju klase u kodu potrebno je ispred nje dodati oznaku kojom govorimo prevoditelju u kojem se imenskom prostoru ona nalazi. U svrhu preglednijeg koda unutar rada izabrano je korištenje *using* naredbi.

4.4. Stvaranje XML dokumenta i opis XML modula

Mehanizam prepoznavanja govora (eng. *Speech Recognition Engine*) kao prvi korak svog inicijaliziranja zahtijeva učitavanje rječnika iz kojeg on bira riječi koje je moguće prepoznati. Moguće je učitati cijeli rječnik čime značajka prepoznaje bilo koje riječi (*dictation*), a u ovom radu je cilj bio dobiti prepoznavanje određenog seta riječi, odnosno naredbi. Kako bi se aplikacija učinila u potpunosti modularnom odabrano je građenje rječnika čitanjem iz XML datoteke. U idućim potpoglavljima bit će opisan format XML dokumenta i njegovo učitavanje.

4.4.1. Formatiranje XML dokumenta

U ovom radu u XML datoteci se nalaze naredbe kojima se glasovno upravlja svjetlećim diodama na mikrokontroleru i čiji će s elementi koristiti za stvaranje rječnika kojim se odabire koje riječi aplikacija prepoznavati. Ova se datoteka mora nalaziti uz izvršnu datoteku i imati naziv *voice_commands.xml*. Također, datoteka mora biti strukturirana na način na koji je strukturirana XML datoteka korištena za izradu ovog diplomskog zadatka i koja se nalazi na slici 4.4. Korijenski element je nazvan `commandgroup` dok je svaki skup naredbi nazvan `command` i predstavlja podelement korijenskog elementa. Svaki podelement sadrži svoje podelemente oznaka `id` i `parameter` pri čemu `id` predstavlja glavnu riječ naredbe koja može stajati i samostalno, a `parameter` jedan od nastavaka naredbe. Pa primjerice imamo skup naredbi "stop" koji je ujedno i `id` i skup naredbi "step" koji ima parametre "up" i "down" što čini dvije naredbe; "step up" i "step down". Ovakva struktura je bitna za kasnije stvaranje rječnika gdje svaka naredba čini jednu cjelinu, odnosno moguće je prepoznati samo "step up", "step down", ali ne i "step" zasebno.

```
<command_group>
    <command>
        <id> stop </id>
    </command>
    <command>
        <id> auto </id>
        <parameter> down </parameter>
        <parameter> up </parameter>
    </command>
    <command>
        <id> step </id>
        <parameter> down </parameter>
        <parameter> up </parameter>
    </command>
    <command>
        <id> position </id>
        <parameter> one </parameter>
    </command>
</command_group>
```

Slika 4.4: XML datoteka s naredbama

4.4.2. XML biblioteka

XML imenski prostor (eng. *namespace*) pruža standardiziranu podršku za obradu XML dokumenata. Unutar njega nalazi se niz klasa kojima je omogućena potpuna obrada XML dokumenta. Ovdje će biti predstavljene samo klase koje su korištene u radu, a ostale klase je moguće vidjeti u službenoj Microsoft dokumentaciji [13].

4.4.3. XML modul

Nakon stvaranja XML dokumenta potrebno je analizirati podatke unutar datoteke i zatim ih pretvoriti u podatke iskoristive prevoditelju. To je napravljeno koristeći klase unutar System Xml imenskog prostora. Zatim je potrebno te podatke spremiti u memoriju kako bi mogli pomoću njih izgraditi rječnik. Budući da je aplikacija rađena u C++/CLI programskom jeziku logičan izbor bio je spremanje, odnosno parsiranje XML dokumeta, napraviti unutar klase. Napravljena je klasa `voiceCommands` koja sadrži dvije metode; jednu čija je funkcionalnost učitavanje i parsiranje XML datoteke i druge koja vraća grešku koja se može dogoditi tijekom parsiranja. Nadalje, klasa sadrži i dvije varijable; jedna je vektor u koju se sprema sadržaj XML dokumenta, a druga služi za vraćanje greške. Metoda `LoadAndParseCommandsFromXml()` kojom se

```
int VoiceCommands::LoadAndParseCommandsFromXml(std::string fileName)
{
    System::Xml::XmlDocument^ xml = gcnew System::Xml::XmlDocument;

    System::String^ fileNameManaged = msclr::interop::marshal_as<System::String^>(fileName);

    try
    {
        xml->Load(fileNameManaged);
    }
    catch (System::IO::FileNotFoundException const^ e)
    {
        return (loadParseCommandsError = ERROR_FILE_NOT_FOUND);
    }
```

Slika 4.5: Učitavanje XML dokumenta

učitava i parsira XML datoteka prima ime datoteke u obliku stringa, a vraća cjelobrojnu vrijednost greške. Prvi korak je stvaranje objekta pomoću konstruktora `XmlDocument` koji je dio Xml imenskog prostora. `XMLDocument` objekt zapravo predstavlja XML datoteku koju učitavamo koristeći metodu `Load()`. Navedena metoda kao argument prima URL, a u ovom slučaju to je putanja do XML datoteke koja je spremljena lokalno uz izvršnu datoteku aplikacije. S obzirom na to, argument je zapravo ime XML datoteke koje smo primili kao argument klase. Budući da metoda prima argument Stringtipa, potrebno je napraviti pretvorbu iz string tipa u managed string tip. Ukoliko učitavanje nije uspjelo javlja se iznimka koju interpretiramo kao grešku ERROR_FILE_NOT_FOUND. Navedena iznimka može se dogoditi u slučaju da korisnik nije stavio XML datoteku uz izvršnu datoteku ili je nazvao datoteku drugačije od onoga što je određeno u dokumentaciji. Zatim se svojstvom DocumentElement dohvaća `commandgroup`, korijenski element u XML datoteci, i njime se inicijalizira `XMLElement` objekt. Ovaj objekt sad sadrži informaciju o broju podelemenata (`ChildNodes`) koji je bitan za proširenje vektora, a također možemo pristupiti i njegovom prvom podelementu. Prvi podelement objekta `root` je prva

```
16      System::Xml::XmlElement^ root = xml->DocumentElement;
17      System::Xml::XmlNode^ node = root->FirstChild;
18
19      m_voiceCommands.resize(root->ChildNodes->Count);
```

Slika 4.6: Definiranje korijenskog i prvog podelementa

naredba unutar korištene XML datoteke i njime incijaliziramo objekt `node` koji nam služi za kretanje po XML datoteci odnosno njenim elementima. Pristupanje prvom podelementu činimo svojstvom `FirstChild`. Svojstvom `ChildNodes` se dohvaćaju svi podelementi trenutnog elementa, a svojstvom `Count` možemo znati koliki je broj podelemenata trenutnog elementa. Time je moguće vektor i vektore koji čine elemente tog vektora proširiti na odgovarajuću veličinu i pomoću informacije o početku i kraju vektora kretati se iteratorima u for petlji. Kretanje elementima i

```
21      for (std::vector<std::vector<std::string>>::iterator voiceCmdIt = m_voiceCommands.begin();
22                                              voiceCmdIt != m_voiceCommands.end(); ++voiceCmdIt)
23      {
24          int cmdContentCount = node->ChildNodes->Count;
25
26          if (!(node->FirstChild->Name->Equals("id")))
27          {
28              return (loadParseCommandsError = ERROR_INVALID_DATA);
29          }
30
31          System::Xml::XmlNode^ childNode = node->FirstChild;
32
33          voiceCmdIt->resize(node->ChildNodes->Count);
34
35          for (std::vector<std::string>::iterator voiceCmdStrIt = voiceCmdIt->begin();
36                                  voiceCmdStrIt != voiceCmdIt->end(); ++voiceCmdStrIt)
37          {
38              std::string value = msclr::interop::marshal_as< std::string >(childNode->InnerText);
39
40              if (value.empty())
41              {
42                  return (loadParseCommandsError = ERROR_EMPTY);
43              }
44              else
45              {
46                  if (voiceCmdStrIt == voiceCmdIt->begin() & childNode->Name->Equals("id"))
47                  {
48                      *voiceCmdStrIt = value;
49                  }
50                  else if (voiceCmdStrIt != voiceCmdIt->begin() & childNode->Name->Equals("parameter"))
51                  {
52                      *voiceCmdStrIt = value;
53                  }
54                  else
55                  {
56                      return (loadParseCommandsError = ERROR_INVALID_DATA);
57                  }
58              }
59
60              childNode = childNode->NextSibling;
61          }
62
63          node = node->NextSibling;
64      }
65
66      return (loadParseCommandsError = ERROR_SUCCESS);
67  }
```

Slika 4.7: Učitavanje elemenata XML datoteke u vektor

vektorima je implementirano koristeći dvije `for` petlje. Istovremeno se kreće parsiranom XML datotekom i vektorom u koji se spremaju naredbe iz datoteke. Vanjskom for petljom se kreće po elementima vektora `m_voiceCommands` u koji spremamo skupove naredbi iz XML datoteke. Uz to se kreće i elementima oznake

`command` koji predstavljaju pojedini skup naredbi. Unutar petlje povjerava se sadrži li prvi podelement oznaku `id`. Ukoliko to nije slučaj, metoda vraća grešku ERROR_INVALID_DATA. Unutarnjom petljom kreće se unutar pojedinog elementa vektora `m_voiceCommands` odnosno kreće se vektorom koji čini svaki pojedini element vektora `m_voiceCommands`. Uz to se kreće unutar skupa naredbe odnosno kreće se podelementima oznaka `id` i `parameter`. Ovdje se provjeravaju ispravan poredak podelemenata i ispravne oznake podelemenata. Metoda vraća grešku ERROR_INVALID_DATA ukoliko jedno od toga ne odgovara strukturi XML datoteke koji smo unaprijed odredili. Također se izvršava provjera vrijednosti podelementa, ukoliko je podelement prazan, metoda vraća grešku ERROR_EMPTY. Pristup tekstu podelementa omogućava svojstvo `InnerText` objekta `XmlNode`. Ovisno o ispunjenju `if` uvjeta njegova vrijednost se pomoću pokazivača pohranjuje u elemente vektora`m_voiceCommands`. Naposljetku je konstruiran vektor čije elemente čine drugi vektori koji sadrže sve naredbe iz XML datoteke.

Nakon svakog prolaska kroz element, odnosno podelement, isti se pomiče na sljedeći element jednake razine (eng. *next sibling*) dok ne dođe do zadnjeg postojećeg elementa, odnosno elementa nasljednika (linije 60 i 63 na slici 4.7). Metoda vraća ERROR_SUCCESS ukoliko je parsiranje uspješno provedeno.

Upotrebom vektora ne alociramo nepotrebno memoriju jer se vektor proširuje ovisno o tome koliko elemenata postoji u XML datoteci. Ovo također pridonosi fleksibilnosti, potpunoj modularnosti XML datoteke i iskorištenju svojstva XML-a koji omogućava neograničen broj elemenata, a u ovom slučaju naredbi.

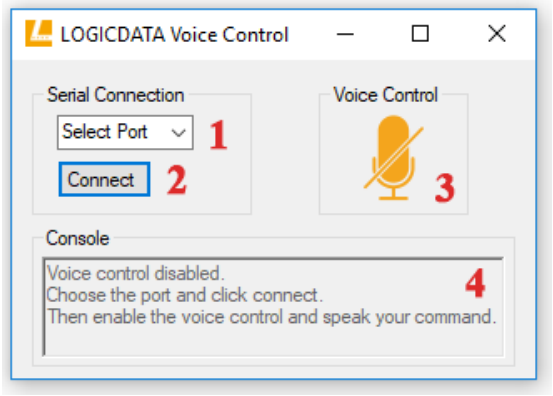
4.5. Osnovna dretva

Osnovnu dretvu stvara operacijski sustav kada se izvrši VoiceControl.cpp datoteka. Ova dretva stvara i upravlja svim komponentama grafičkog korisničkog sučelja kao i prepoznavanjem govora.

4.5.1. Inicijalizacija aplikacije

Desktop aplikacija rađena je u obliku Windows forme. Grafičko korisničko sučelje prikazano je na slici 4.8, a sastoji se od:

1. komponente kojom se omogućava odabir serijskog priključka
2. gumba za otvaranje odnosno zatvaranje serijskog priključka



Slika 4.8: Inicijalno stanje aplikacije

3. gumba za omogućavanje odnosno onemogućavanje prepoznavanja govora

4. konzole kojom se osigurava komunikacija s korisnikom

Sve grafičke komponente se inicijaliziraju unutar `VoiceControl` klase funkcijom `InitializeComponent()` koju generira Windows Form Designer kada se aplikacija dizajnira koristeći `Designer`. Uz grafičke komponente stvaraju se objekti i pozivaju metode potrebne za rad aplikacije. Instancira se klasa `SpeechRecognitionEngine` u obliku `recognizer` i događaju `SpeechRecognized` se dodjeljuje rukovoditelj događaja. Stvara se objekt klase `VoiceCommands` i poziva se metoda kojom se učitava i sprema XML datoteka u memoriju. Metodi je predano ime XML datoteke u obliku stringa. Stvaraju se dretve (eng. *thread*) za sintezu govora i serijsku komunikaciju i poziva metoda kojom se u komponentu `ComboBox` dodaju dostupni serijski priključci računala. Potpuni kod klase `VoiceControl` nalazi se na CD-u u prilogu.

4.5.2. Inicijalizacija mehanizma za prepoznavanje govora

Kao mehanizam za prepoznavanje govora unutar ovog rada korištena je `Speech Recognition Engine` klasa koja je dostupna unutar Speech Recognition imenskog prostora. `Speech Recognition Engine` je unutar procesni prepoznavatelj govora (eng. *speech recognizer*) što znači da ga koristi samo proces koji ga je instancirao. Time je omogućeno proizvoljno mijenjanje konfiguracije mehanizma za prepoznavanje govora i pripadnih operacija. Na slici 4.9 je prikazano stvaranje `Speech Recognition Engine` objekta s definiranim jezikom i kulturom koji se žele koristiti.

```
SpeechRecognitionEngine^ recognizer;
recognizer = gcnew SpeechRecognitionEngine(gcnew CultureInfo("en-US"));
```

Slika 4.9: Speech Recognition Engine inicijalizacija

4.5.3. Provjera grešaka tijekom parsiranja XML dokumenta

Prilikom parsiranja XML dokumenta osigurano je da aplikacija ne dopušta građenje rječnika pomoću nepravilno strukturiranog dokumenta. Tijekom učitavanja podataka iz XML dokumenta u memoriju provjeravaju se oznake, postojanje praznih elemenata i pravilan poredak elemenata. Ukoliko se javila neka od grešaka, funkcija `LoadAndParseCommandsFromXml()` vraća grešku kojom onda upravlja (eng. *handling*) funkcija `ParseErrorHandling()`. Ovisno o tipu greške, navedena funkcija ispisuje na konzolu tip greške i dodatna uputstva korisniku. U slučaju da je parsiranje uspješno izvedeno, odnosno nije došlo do pogreške, poziva se funkcija kojom se izgrađuje i učitava rječnik. U tablici 5.1 navedene su sve greške koje se mogu pojaviti i uzroka njihovog nastanka.

Tablica 4.1: Lista grešaka tijekom parsiranja i njihov uzrok

ERROR_FILE_NOT_FOUND	netočan naziv ili lokacija XML datoteke na disku
ERROR_INVALID_DATA	netočan poredak čvorova ili nepostojanje istih
ERROR_EMPTY	element ne sadrži tekst
ERROR_SUCCESS	uspješno parsiranje

4.5.4. Građenje i učitavanje rječnika

U kontekstu prepoznavanja govora, rječnik (eng. *grammar*) je skup pravila ili ograničenja koji određuje koje riječi ili izjave mehanizam za prepoznavanje govora prepoznaje kao smislene.

Unutar System Speech Recognition biblioteke izgradnju rječnika osigurava klasa `Grammar`. Prvi korak, stvaranje rječnika, moguće je napraviti programski ili pomoću odvojene datoteke. Zatim, ovisno o načinu na koji smo napravili rječnik, koristimo pripadni konstruktor kojim se rječnik ugrađuje u objekt klase `Grammar`. Navedeni objekt se zatim učitava u mehanizam prepoznavanja govora i na raspolaganju je aplikaciji za prepoznavanje govora tijekom njenog izvršavanja.

Stvaranje rječnika pomoću datoteke zahtijeva putanju datoteke koja sadrži opis rječnika u podržanom formatu. Podržani formati uključuju XML datoteke koje su u

skladu s W3C (eng. *World Wide Web Consortium*) SRGS (Speech Recognition Grammar Specification) specifikacijom (verzija 1.0) kao i rječnike koji su prevedeni u binarnu datoteku s .cfg datotečnim nastavkom.

S druge strane, programsko stvaranje rječnika zahtijeva stvaranje `GrammarBuilder` ili `SrgsDocument` objekta.

`GrammarBuilder` klasa pruža drugi način programskog stvaranja rječnika. Nakon stvaranja `GrammarBuilder` objekta potrebno mu je dodati ograničenja. To mogu biti `String` objekti, klasa `Choices`, klasa `SemanticResultKey` ili drugi `GrammarBuilder` objekti koji mogu definirati ograničenja za rječnik. `Choices` objekt predstavlja element izraza koji može imati više od jedne vrijednosti, odnosno on predstavlja skup alternativnih vrijednosti u ograničenjima rječnika. `SemanticResultKey` objektom se definira semantička struktura rječnika.

`SrgsDocument` klasa nalazi se unutar System Speech Recognition SrgsGrammar imenskog prostora koja sadrži klase pomoću kojih je moguće programski izraditi rječnike koji su u skladu sa SRGS specifikacijom (verzija 1.0).

Mehanizam prepoznavanja govora, kojeg koristi `SpeechRecognizer`, odnosno `SpeechRecognitionEngine` objekt, može učitati višestruke rječnike za prepoznavanje govora. Svaki od rječnika je objekt klase `Grammar` koji sadrži razna svojstva koja sadrže informaciju o samom objektu ili olakšavaju rukovanje višestrukim rječnicima. Pa je tako pomoću `Enabled` svojstva moguće uključiti ili isključiti pojedine rječnike, odnosno omogućiti ili onemogućiti njihovo korištenje u svrhu prepoznavanja govora. Nadalje, pomoću `Priority` svojstva moguće je postaviti prioritet koji pojedini rječnik ima u odnosu na ostale rječnike. Svakom rječniku se dodjeljuje naziv pomoću `Name` svojstva, a pomoću `Loaded` svojstva može se saznati je li rječnik učitan u mehanizam prepoznavanja govora.

Sve navedene i dodatne informacije o `Grammar` klasi dostupne su u službenoj Microsoft dokumentaciji [14].

4.5.5. Implementacija izgradnje rječnika za prepoznavanje govora

Stvaranje rječnika i učitavanje rječnika u `SpeechRecognitionEngine` napravljeno je u funkciji `BuildCommandGrammars()` čiji je kod prikazan na slici 4.10.

Budući da je željeni rječnik za ovu aplikaciju prilično jednostavan i može se smatrati kao skup lista, rječnik je stvoren programski koristeći `GrammarBuilder` i `Choices`

```
void BuildCommandGrammars(VoiceCommands &voiceCmd, SpeechRecognitionEngine^ recognizer)
{
    int commandNum = voiceCmd.m_voiceCommands.size();
    array<GrammarBuilder^>^ buildCommand = gcnew array<GrammarBuilder^>(commandNum);

    for (std::vector<std::vector<std::string>>::iterator voiceCmdIt = voiceCmd.m_voiceCommands.begin();
                                                         voiceCmdIt != voiceCmd.m_voiceCommands.end();
                                                                       ++voiceCmdIt)
    {
        int grammIndex = std::distance(voiceCmd.m_voiceCommands.begin(), voiceCmdIt);

        buildCommand[grammIndex] = gcnew GrammarBuilder();

        String^ commandName = "command_" + grammIndex;

        Choices^ cmdId;
        Choices^ cmdParams;

        for (std::vector<std::string>::iterator voiceCmdStrIt = voiceCmdIt->begin();
                                                voiceCmdStrIt != voiceCmdIt->end();
                                                                 ++voiceCmdStrIt)
        {
            if (voiceCmdStrIt == voiceCmdIt->begin())
            {
                cmdId = gcnew Choices();
                cmdId->Add(msclr::interop::marshal_as<String^>(*voiceCmdStrIt));

                buildCommand[grammIndex]->Append(cmdId);
            }
            else
            {
                if (std::distance(voiceCmdIt->begin(), voiceCmdStrIt).Equals(1))
                {
                    cmdParams = gcnew Choices();
                }
                cmdParams->Add(msclr::interop::marshal_as<String^>(*voiceCmdStrIt));
            }
        }

        if (voiceCmdIt->capacity() > 1)
        {
            buildCommand[grammIndex]->Append(gcnew SemanticResultKey(commandName, cmdParams));
        }

        array <Grammar^>^ commandGrammar = gcnew array <Grammar^>(commandNum);

        commandGrammar[grammIndex] = gcnew Grammar(buildCommand[grammIndex]);
        commandGrammar[grammIndex]->Name = "Command Grammar";
        recognizer->LoadGrammar(commandGrammar[grammIndex]);

        commandName = "";
    }

    Grammar^ disableGrammar = gcnew Grammar(gcnew GrammarBuilder("disable"));
    disableGrammar->Name = "Disable Grammar";
    recognizer->LoadGrammar(disableGrammar);

    Grammar^ confirmationGrammar = gcnew Grammar(gcnew GrammarBuilder("yes"));
    disableGrammar->Name = "Confirmation Grammar";
    recognizer->LoadGrammar(confirmationGrammar);

    Grammar^ rejectionGrammar = gcnew Grammar(gcnew GrammarBuilder("no"));
    disableGrammar->Name = "Rejection Grammar";
    recognizer->LoadGrammar(rejectionGrammar);
}
```

Slika 4.10: Izgradnja i učitavanje rječnika za prepoznavanje govora

klase. Kao što je spomenuto, `Choices` omogućava upotrebu alternativnih vrijednosti u ograničenjima rječnika. U napravljenoj aplikaciji alternativne vrijednosti su zapravo parametri naredbe. Prvo se stvara polje `GrammarBuilder` objekata. Veličina polja je određena brojem elemenata vektora koji je predan funkciji, odnosno vektora koji sadrži sve naredbe. Zatim se pomoću dvije `for` petlje kreće tim istim vektorom. Kao i u `LoadAndParseCommandsFromXml()` funkciji vanjskom `for` petljom se iteratorom kreće po elementima vektora. Svaki element vektora predstavlja skup naredbi. Za svaki element alocira se mjesto na pripadajućem indeksu u polju `buildCommand`. Unutarnjom `for` petljom kreće se unutar skupa naredbi. Zbog načina formatiranja i izvršene provjere ispravnosti formata XML datoteke poznato je da prvi element skupa

naredbi sadrži `id` naredbe. Za prvi element alocira se `cmdId` objekt i dodaje mu se vrijednost elementa na koju pokazuje trenutni iterator. Zatim se `cmdId` objekt dodaje `GrammarBuilder` objektu na pripadajućem indeksu polja. U slučaju da je riječ parametru za prvi parametar se alocira `cmdParams` objekt, a potom mu se za taj i svaki sljedeći parametar dodaje vrijednost tog istog parametra. Nakon izlaska iz unutarnje `for` petlje, ukoliko postoje parametri u trenutnom skupu naredbi, oni se dodaju objektu unutar `buildCommand` polja pripadajućeg indeksa. Dodaje se objekt `SemanticResultKey` čiji konstruktor prima dva argumenta; ime skupa naredbi i objekt koji predstavlja parametre danog skupa naredbi. Ovime se zapravo stvara lista prihvatljivih parametara za određeni `id`; primjerice za "step" su prihvatljivi parametri "up" i "down" što daje naredbe "step up" i "step down".

Nakon toga se stvori polje `Grammar` objekata. Veličina polja je određena brojem skupa naredbi. Zatim se za svaki skup naredbi gradi rječnik, dodjeluje mu se ime i naposljetku ga se učitava u prepoznavatelj, odnosno korišteni mehanizam prepoznavanja govora.

Naposljetku se izgrade tri dodatna rječnika koja sadrže fiksne naredbe "disable", "yes" i "no" pomoću kojih se upravlja tokom rada programa.

4.5.6. Omogućavanje prepoznavanja govora

Nakon što je učitan rječnik, dopušteno je omogućiti prepoznavanje govora. Omogućavanje govora implementirano je pritiskom mišem na prekriženi mikrofon u grafičkom sučelju aplikacije. Slika mikrofona se nalazi unutar komponente `PictureBox` koja je, kao i sve ostale korištene komponente, dio System Windows Forms imenskog prostora. Svaka komponenta sadrži događaje, metode koje se aktiviraju na akciju korisnika. Svakom događaju komponente se može pridjeliti rukovoditelj događaja koji bi se aktivirao u slučaju događaja kojeg primjerice izaziva pritisak mišem.

```
System::Void voiceControlEnableButton_Click(System::Object^  sender, System::EventArgs^  e)
{
    if (controlButtonStatus)
    {
        Sleep(100);

        try
        {
            recognizer->SetInputToDefaultAudioDevice();
        }
        catch (InvalidOperationException^ exception)
        {
            idleState = true;
            SendToVoiceControlConsole("Error: Audio device not detected.");
            return;
        }

        voiceControlEnableButton->Image = (cli::safe_cast<Drawing::Bitmap^>(resourceManager->GetObject("microphone_on")));

        idleState = false;
        SendToVoiceControlConsole("Voice control enabled.\nWaiting command...");

        try
        {
            recognizer->RecognizeAsync(RecognizeMode::Multiple);
        }
        catch (InvalidOperationException^ ex)
        {
            idleState = true;
            SendToVoiceControlConsole("Error: starting voice control.\nXml file not found.");
            AppendToVoiceControlConsole("\nXml file should be named as voice_commands.xml.");
            return;
        }

        speechRecognitionEnabled = true;
    }
    else
    {
        recognizer->RecognizeAsyncCancel();

        if (!serialCommunicationInit)
        {
            SerialCommunicationThread->Suspend();
        }

        voiceControlEnableButton->Image = (cli::safe_cast<Drawing::Bitmap^>(resourceManager->GetObject("microphone_off")));

        idleState = true;
        SendToVoiceControlConsole("Voice control disabled.\nChoose the port and click connect.");
        AppendToVoiceControlConsole("\nThen enable the voice control and speak your command.");
        speechRecognitionEnabled = false;
    }

    controlButtonStatus = !controlButtonStatus;
}
```

Slika 4.11: Omogućavanje prepoznavanja govora

Nakon pritiska mišem ulazi se u rukovoditelj događaja i provjerava se globalna varijabla `controlButtonStatus`. Ona sadrži informaciju je li aplikacija trenutno u stanju prepoznavanja govora i njome se implementira omogućavanje i onemogućavanje prepoznavanja govora. Ukoliko navedena varijabla ima vrijednost "true" aplikaciju se stavlja u stanje čekanja na 100 milisekundi. Ovo je napravljeno kako bi se aplikaciji dalo vremena u slučaju da korisnik neprestano pritišće gumb. Zatim je potrebno konfigurirati na koji način prepoznavatelj prima zvučni signal. `SetInputToDefaultAudioDevice()` metodom određeno je da se zvučni signal prima sa standardnog uređaja unutar Windows® operativnog sustava. Zatim se umjesto prekriženog mikrofona u komponentu stavlja slika mikrofona koja označava da je prepoznavanje govora omogućeno. Preostaje omogućavanje samog prepoznavanje govora korištenjem `RecognizeAsync()` metode. Ovisno o argumentu koji prima, `RecognizeMode::Single` ili `RecognizeMode::Multiple`, njome se omogućava jedna ili više asinkronih operacija prepoznavanja govora. Na ponovni pritisak mišem na mikrofon, metodom `RecognizeAsyncCancel()`

prepoznavatelju se onemogućuje prepoznavanje govora bez da se čeka na završetak trenutne operacije prepoznavanja. Uz to se i osvježava slika mikrofona na grafičkom sučelju aplikacije.

Nakon što je omogućeno prepoznavanja govora, u svrhu dobivanja rezultata asinkrone operacije prepoznavanja govora potrebno je događaju `SpeechRecognized` pridjeliti rukovoditelj događaja. Ovaj događaj pobuđuje prepoznavatelja svaki put kada uspješno završi operaciju prepoznavanja govora.

```
recognizer->SpeechRecognized += gcnew EventHandler<SpeechRecognizedEventArgs^>
        (this, &LDOfficeVoiceControl::VoiceControl::SpeechRecognized_Handler);
```

Slika 4.12: Pridjeljivanje rukovoditelja događaja

4.5.7. Obrada prepoznatog govora

Obrada prepoznatog govora vrši se u rukovoditelju događaja `SpeechRecognized_Handler()`. Ova funkcija kao argumente prima dva objekta. Prvi je objekt koji je pobudio događaj i koji je zatim aktivirao rukovoditelj događaja, u ovom slučaju taj objekt je `recognizer`. Drugi argument je objekt koji sadrži dodatne informacije o događaju i pomoću njega možemo pristupiti prepoznatom govoru koji je pretvoren u tekst. Pomoću svojstva `Result` pristupa se rezultatu prepoznatog govora koji je izazvao događaj. [15] Unutar klase `RecognitionResult` nalaze se `Grammar` i `Text` svojstva. Pomoću `Grammar` svojstva moguće je saznati koji rječnik je bio upotrijebljen za prepoznavanje govora dok `Text` svojstvo sadrži normalizirani tekst koji je generirao prepoznavatelj iz prepoznatog govora. [16]

S obzirom na sadržaj teksta ispunjava se jedan od nekoliko `if` uvjeta. Potpun kod ove funkcije dan je na CD-u u prilogu. Opisat će se dva bloka koja se izvršavaju unutar dva ključna `if` uvjeta. Na slici 4.13 prikazan je blok naredbi koji se izvršava u slučaju da je korisnik potvrdio da je prepoznata naredba zaista naredba koja je i izrečena. Prvo se vrijednost globalne varijable `receivingEnabled` postavlja u `true`. Ovom varijablom se upravlja logikom rukovoditelja događaja `SerialPort_DataReceived()`. Nakon toga pokreće se dretva unutar koje se upravlja slanjem podataka serijskom komunikacijom. Oboje će biti opisano u poglavlju 4.7. Na slici 4.14 prikazan je blok naredbi koji se izvršava u slučaju da je korisnik potvrdio prijašnju naredbu kojom želi onemogućiti prepoznavanje govora. Unutar

```
else if (e->Result->Text->Equals("yes") & commandPreviousStatus == CommandState::COMMAND_STATE_COMMAND_SPOKEN)
{
    receivingEnabled = true;
    commandStatus = CommandState::COMMAND_STATE_COMMAND_CONFIRMED;
    commandPreviousStatus = CommandState::COMMAND_STATE_COMMAND_CONFIRMED;

    if (serialCommunicationInit)
    {
        SerialCommunicationThread->Start();
        serialCommunicationInit = false;
    }
    else
    {
        SerialCommunicationThread->Resume();
    }
}
```

Slika 4.13: Potvrda naredbe

```
else if (e->Result->Text->Equals("yes") & commandPreviousStatus == CommandState::COMMAND_STATE_DISABLE_SPOKEN)
{
    commandStatus = CommandState::COMMAND_STATE_DISABLE_CONFIRMED;
    commandPreviousStatus = CommandState::COMMAND_STATE_DISABLE_CONFIRMED;

    voiceControlEnableButton->Image = (cli::safe_cast<Drawing::Bitmap^>(resourceManager->GetObject("microphone_off")));

    if (ENABLE_TEXT_OUTPUT)
    {
        idleState = true;
        SendToVoiceControlConsole("Voice control disabled.\nChoose the port and click connect.");
        AppendToVoiceControlConsole("\nThen enable the voice control and speak your command.");
    }
    controlButtonStatus = true;

    semaphoreSpeechSynth->Release();
}
```

Slika 4.14: Glasovno onemogućavanje prepoznavanja govora

bloka osvježava se slika mikrofona u grafičkom sučelju aplikacije koja označuje prestanak proces prepoznavanja govora. Zatim se semaforom ponovno pokreće dretva za sintezu govora unutar koje se onda onemogućuje prepoznavanje govora.

4.6. Dretva za sintezu govora

Zbog načina na koji je implementirana komponenta `CheckBox` u Windows formama i zbog implementacije istodobne sinteze govora i ispisa u konzolu pojavila se potreba za korištenjem višedretvenosti (eng. *multithreading*). Dodavanjem odgovora korisniku pomoću sinteze govora željela se postići responzivnost aplikacije i intuitivnije korištenje korisniku.

Unutar Speech Synthesis imenskog prostora dostupna je `Speech Synthesizer` klasa [17] koja omogućava korištenje *text-to-speech* funkcionalnosti unutar Windows® operativnog sustava. Sinteza govora implementirana je korištenjem te klase unutar dretve za sintezu govora. Metodama unutar imenskog prostora moguće je odabrati glas sintetizatora govora (eng. *SpeechSynthesizer*), a izmjenom svojstava sintetizatora govora, i brzinu (`Rate`) i jačinu govora (`Volume`).

Nakon stvaranja samog objekta sintetizatora govora, metodom

`SetOutputToDefaultAudioDevice()` se određuje gdje će sintetizator govora generirati govor, a u ovom se slučaju zvuk šalje na standardni uređaj.

Budući da je sintetizirani govor potrebno simulirati tek nakon što je naredba prepoznata, dretva za sintezu govora se ne mora izvršavati tijekom cijelog rada aplikacije. Time je potrošnja procesora koju aplikacija generira pri radu značajno smanjena. Upravljanje dretvom učinjeno je korištenjem semafora i metodama `Release()` i `WaitOne()`, a njezinim izvršavanjem upravlja dretva za serijsku komunikaciju i rukovoditelj događaja prepoznatog govora kada je potreban odgovor korisniku pomoću sintetiziranog govora. Budući da se istovremeno izvršava ispisivanje u konzolu i davanje izlaza u obliku sintetiziranog odgovora, bilo je nužno sinkronizirati rukovoditelj događaja s dretvom za sintezu govora. U tu svrhu definirana je `CommandState` enumeracija i dvije globalne varijable tog tipa, `commandStatus` i `commandPreviousStatus`. `CommandState` enumeracija sadrži identifikatore koji predstavljaju stanje u kojem se naredba nalazi. Varijable poprimaju neku od vrijednosti identifikatora i služe za upravljanje govornog izlaza u dretvi za sintezu govora. Tijekom procesa samog govora, prepoznavanje govora je onemogućeno kako bi se eliminirala mogućnost da prepoznavatelj sintetitzirani govor prepozna kao naredbu. Nakon što je sintetizator završio sa simulacijom govora, prepoznavatelju se opet omogućava prepoznavanje govora i u konzolu se ispisuje obavijest da je aplikacija ponovno spremna na prepoznavanje govora. Nakon što je dan odgovor u obliku sintetiziranog govora, dretva je blokirana dok se njeno izvršavanje ne pokrene ponovno.

4.7. Serijska komunikacija

Serijska komunikacija unutar aplikacije realizirana je komponentom `SerialPort` koja je dio System IO Ports imenskog prostora. Ovaj imenski prostor pruža API za pristupanje serijskim priključcima. [18] Primanje podataka realizirano je unutar osnovne dretve dok je slanje podataka realizirano unutar dretve za serijsku komunikaciju.

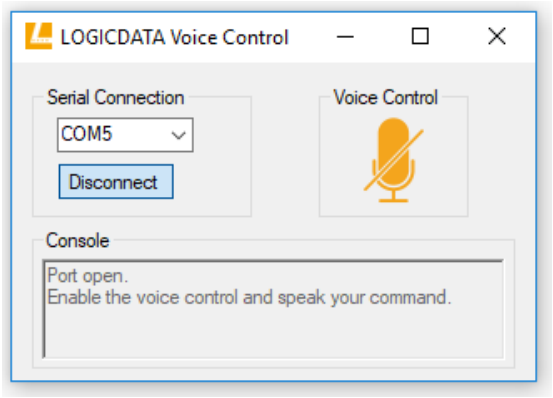
4.7.1. Dohvaćanje serijskih priključaka

Odabir serijskog priključka u aplikaciji implementirano je klasom `ComboBox` dostupnoj unutar System Windows Controls imenskog prostora, odnosno Presentation Framework biblioteke. Pomoću ovog objekta, odnosno komponente grafičkog sučelja moguće je izabrati jedan od ponuđenih priključaka unutar padajuće liste. Priključci se dohvaćaju pomoću metode `GetPortNames()` koja vraća polje serijskih priključaka

za računalo na kojem se program izvršava. Zatim se tim poljem dodaje raspon izbora unutar `ComboBox` objekta. Sve navedeno implementirano je unutar metode `SerialPortGetNames()` koja se poziva unutar klase `VoiceControl`. Potpun kod moguće je vidjeti na CD-u u prilogu.

4.7.2. Inicijalizacija serijske komunikacije

Inicijalizacija serijske komunikacije uključuje odabir priključka i otvaranje priključka pritiskom na gumb "Connect". Pritiskom na gumb "Connect" podiže se događaj koji onda podiže dodijeljeni mu rukovoditelj događaja. Unutar njega se objektu koji predstavlja serijski priključak dodjeljuje ime priključka koji je korisnik odabrao i isti se zatim otvara. Zatim se omogućava aktiviranje gumba kojim se upravlja uključivanjem prepoznavanja govora. Naposljetku se u konzoli ispisuje obavijest da je priključak otvoren i da je potrebno omogućiti prepoznavanje govora. Gumb "Connect" se osvježava postajući "Disconnect". Treba spomenuti i deinicijalizaciju koja se pokreće



Slika 4.15: Izgled aplikacije nakon pritiska na gumb "Connect"

ponovnim pritiskom gumba. Time se onemogućuje pritisak gumba kojim se upravlja uključivanje prepoznavanja govora i gumb se osvježava tako da označuje isključenje prepoznavanja govora (slika prekriženog mikrofona). Zatim se prepoznavatelju onemogućuje prepoznavanje govora i priključak se zatvara. Naposljetku se gumb "Disconnect" osvježava. Inicijalizacija i deinicijalizacija se ne mogu izvršiti u slučaju da se XML datoteka nije uspjela uspješno učitati ili obraditi. Navedeno je osigurano ispitivanjem varijable error u kojoj je spremljena informacija o uspješnosti `LoadAndParseCommandsFromXml()` metode.

4.7.3. Slanje podataka

Slanje podataka serijskom vezom unutar `SerialPort` klase moguće je implementirati korištenjem `Write()` metode. Ova metoda kao argument prima `String` kojeg potom šalje na serijski priključak. Slanje podataka u programu je implementirano u dretvi za serijsku komunikaciju. Dretva započinje odnosno nastavlja s izvršavanjem kada korisnik potvrdi prepoznatu naredbu, a pokretanje je izvršeno u rukovoditelju događaja prepoznatog govora opisanog u 4.5.7 poglavlju.

Tijekom izvršavanja dretve za serijsku komunikaciju prepoznavatelju je onemogućeno prepoznavanje govora. Unutar dretve prvo se podatak šalje na serijski priključak gdje je podatak potvrđena naredba koja je spremljena u globalnu varijablu. Zatim se semaforom čeka potvrda od mikrokontrolera da je primio podatak odnosno izvršio naredbu. Vrijeme čekanja potvrde primitka je određena makronaredbom `TIMEOUT_RX_DATA` i iznosi tri sekunde. Semafor dopušta nastavak izvršavanja dretve unutar rukovoditelja događaja primljenog podatka koji će se opisati u idućem poglavlju. Ukoliko je mikrokontroler poslao potvrdu da je primio naredbu, semafor zadužen za dretvu za sintezu govora pokreće tu istu dretvu koji govornim izlazom javlja korisniku da se naredba izvršava, a isto se isto ispisuje u konzoli. U slučaju da nije bilo potvrde primitka, prepoznavatelju se omogućava prepoznavanje govora, u konzolu se ispisuje obavijest da se čeka na naredbu i dretva se zaustavlja.

4.7.4. Primanje podataka

Metodama i događajima unutar `SerialPort` klase moguće je serijskom komunikacijom primiti podatke. Naime, prilikom slanja na serijski priključak podiže se događaj `DataReceived` koji ukazuje na to da su primljeni podaci preko priključka koji je predstavljen kao `SerialPort` objekt. Ovom događaju dodjeljuje se rukovoditelj unutar kojeg se čitaju primljeni podaci. Čitanje podataka izvršava se metodom `ReadByte()` koja sinkrono čita jedan bajt iz ulaznog međuspremnika (eng. *buffer*). Svakim čitanjem dekrementira se vrijednost svojstva `BytesToRead`. Kada vrijednost dosegne nulu, pročitan je cijeli podatak, odnosno svi bajtovi podatka. U rukovoditelju `SerialPort_DataReceived()` čita se bajt po bajt podatka i sprema u string varijablu. Zatim se ispituje je li podatak jednak riječi za koju se odredilo da označava primitak i izvršenje naredbe i semaforom se pokreće dretva za serijsku komunikaciju.

5. Razvoj programske podrške za mikrokontroler

Struktura datoteka razvijene programske podrške sastoji se od dva sloja, aplikacijskog sloja i sloja koji čine moduli za upravljačke programe (eng. *driver*). Struktura datoteka prikazana je na slici 5.1. Unutar aplikacijskog sloja implementirana je sama

Slika 5.1: Struktura datoteka razvijene programske podrške

funkcionalnost aplikacije dok drugi sloj čine moduli za upravljačke programe korištenog sklopovlja. Moduli za upravljačke programe uključuju funkcije za upravljanje pinovima opće namjene, realizirane u gpio.c/h datotekama, i funkcije za upravljanje serijskom komunikacijom realizirane u uart.c/h datotekama. Navedeni moduli služe kao poveznica između aplikacijskog sloja i samih upravljačkih programa.

5.1. Aplikacijski sloj

Unutar aplikacijskog sloja implementirano je primanje i slanje podataka serijskom komunikacijom te prikaz izvršavanja pojedine naredbe. Prikaz izvršavanja pojedine naredbe je implementirano paljenjem odgovarajuće svjetleće diode frekvencijom koja odgovara pojedinoj naredbi. U nastavku su navedene funkcije koje se nalaze unutar main.c datoteke koja predstavlja aplikacijski sloj. Na početku je definirana konstanta trajanja izvršavanja naredbe u milisekundama (`COMMAND_DURATION_MS`) koja je za potrebe ovog rada postavljena na 5000.

- `SysTickConfig` - funkcija za konfiguraciju Systick Base vremena za prekid na trajanje od jedne milisekunde
- `USART_CbRx` - funkcija za spremanje primljenih bajtova podataka u međuspremnik, kao ulazni parametar prima 8-bitni podatak iz registra i pokazivač na međuspremnik
- `GPIO_LED_Config` - funkcija za konfiguraciju pina svjetleće diode, kao ulazne parametre prima GPIO port i pin svjetleće diode kojom se želi upravljati
- `GPIO_LED_Toggle` - funkcija za paljenje i gašenje svjetleće diode, kao ulazne parametre prima GPIO port i pin svjetleće diode kojom se želi upravljati i varijablu koja predstavlja odgodu paljenja odnosno gašenja svjetleće diode u milisekundama. Ovom varijablom se regulira frekvencija paljenja svjetleće diode za pojedinu naredbu.
- `UART_Communication` - funkcija za serijsku komunikaciju, ovdje se postavljaju željeni parametri za serijsku komunikaciju i poziva se funkcija `UART_AutoReceive` kojom je implementirano primanje podataka
- `UART_ParseCommand` - funkcija za obradu primljenog podatka. Elementi podatkovnog niza se čitaju iz međuspremnika dok se ne naiđe na oznaku kraja niza, a kada se pročita kraj niza, podatak se uspoređuje s mogućim naredbama i nakon uspješne usporedbe, serijskom vezom se šalje potvrda primitka naredbe. Zatim se poziva funkcija `GPIO_LED_Toggle` kojoj se predaju argumenti ovisno o naredbi koja je primljena. Tijekom obrade primljenog podatka daljnje primanje podataka serijskom komunikacijom je onemogućeno. Funkcija kao ulazne parametre prima pokazivač na međuspremnik i UART port.

U tablici 5.1 navedene su implementirane naredbe i način prikaza naredbi za korišteni mikrokontroler. Korištene svjetleće diode su ujedno i jedine programski upravljive diode na korištenoj Discovery pločici, a riječ je o plavoj i zelenoj svjetlećoj diodi kojima odgovaraju pinovi PC8 i PC9.

5.2. Moduli za upravljačke programe

5.2.1. Modul za upravljanje pinovima opće namjene

Driverski modul za upravljanje pinovima opće namjene služi kao poveznica između driverskih funkcija upravljanja pinova opće namjene i aplikacijskog sloja. Modul je napravljen za specifično sklopovlje gdje je razvojnom inženjeru omogućeno indirektno

Tablica 5.1: Popis implementiranih naredbi i način prikaza

naredba	svjetleća dioda	frekvencija [Hz]
STOP	zelena	0
STEP UP	zelena	0.6
STEP DOWN	plava	0.6
AUTO UP	zelena	2
AUTO DOWN	plava	2
POSITION ONE	plava	0

korištenje driverskih funkcija, odnosno funkcija upravljačkog programa, kojima se upravljaju pinovi opće namjene. Uz funkcije, driverski modul sadrži nekoliko enumeracija i struktura. Razvijene funkcije i strukture su navedene u nastavku dok se enumeracije mogu vidjeti u na CD-u u prilogu gdje se nalazi potpuna programska podrška razvijena za korišteni mikrokontroler.

- `GPIO_PinConfiguration` - struktura koja sadrži parametre potrebne za konfiguraciju pina
- `GPIO_PortPinId` - struktura koja sadrži parametre GPIO pina i porta
- `GPIO_PinConfig` - funkcija za konfiguraciju GPIO pina, omogućavanje i onemogućavanje AHB (eng. *Advanced High-perfomance Bus* periferijskog izvora takta i inicijalizaciju i deinicijalizaciju GPIO periferije; kao ulazne parametre prima pokazivač na `GPIO_PinConfiguration` strukturu podataka i varijablu tipa `bool` koja određuje konfiguraciju ili dekonfiguraciju porta
- `GPIO_PinStateSet` - funkcija za postavljanje ili resetiranje GPIO pina; kao ulazne parametre prima pokazivač na strukturu `GPIO_PortPinId` i varijablu tipa `bool` koja određuje postavljanje ili resetiranje pina
- `GPIO_PinStateGet` - funkcija za dohvaćanja stanja GPIO pina; kao ulazni parametar prima pokazivač na strukturu `GPIO_PortPinId`, a kao izlazni parametar vraća informaciju o stanju pina
- `GPIO_PinStateToggle` - funkcija za naizmjenično postavljanje i resetiranje GPIO pina (eng. *toggle*); kao ulazni parametar prima pokazivač na strukturu `GPIO_PortPinId`.

5.2.2. Modul za upravljanje serijskom komunikacijom

Modul za upravljanje serijskom komunikacijom povezuje funkcije upravljačkog programa za serijsku komunikaciju i aplikacijski sloj. Korištenjem modula omogućeno je jednostavno korištenje UART komunikacije za primanje i slanje podataka u aplikacijskom sloju. Uz pozivanje odgovarajućih funkcija, u aplikacijskom sloju se postavljaju parametri potrebni za serijsku komunikaciju kao što je baud rate, korišteni serijski priključak, broj podatkovnih i paritetnih bitova, odabrani tx i rx pinovi i ostalo. U nastavku su navedene bitnije funkcije modula i jedna struktura, a potpunu programsku podršku je moguće naći u na CD-u u prilogu.

- `UART_Config` - struktura koja sadrži parametre za konfiguraciju sklopovlja i parametre *callback* funkcije serijske komunikacije
- `UART_PinConfig` - funkcija za konfiguraciju pina za serijsku komunikaciju; kao ulazni parametar prima pokazivač na strukturu `UART_Config`, a kao izlazni parametar vraća 0 ako je port uspješno konfiguriran, a -1 ukoliko nije
- `UART_SetConfig` - funkcija za konfiguraciju sklopovlja serijske komunikacije; kao ulazne parametre prima pokazivač na strukturu `UART_Config` i pokazivač na strukturu koja sadrži parametre potrebne za konfiguraciju USART upravljačkog programa; kao izlazni parametar vraća 0 ako je konfiguracija bila uspješna, a -1 ukoliko nije
- `UART_PortConfigure` - funkcija za konfiguraciju porta za serijsku komunikaciju; kao ulazni parametar prima pokazivač na strukturu `UART_Config`, a kao izlazni parametar vraća 0 ako je port uspješno konfiguriran, a -1 ukoliko nije
- `USART_Handler` - funkcija za rukovođenje prekida uslijed primanja podatka; ovdje se vrši čitanje primljenog podatka direktno iz registra i spremanje istoga u međuspremnik
- `UART_Send` - funkcija za slanje podatka veličine 8 ili 16 bitova; kao ulazne parametre prima port za serijsku komunikaciju, pokazivač na međuspremnik i broj elemenata podatka unutar međuspremnika te kao izlazni parametar vraća 0 ako je podatak uspješno poslan, a -1 ukoliko nije
- `UART_Receive` - funkcija za primanje podatka veličine 8 ili 16 bitova; kao ulazne parametre prima port za serijsku komunikaciju, pokazivač na međuspremnik i broj elemenata podatka unutar spremnika te kao izlazni parametar vraća 0 ako je podatak uspješno primljen, a -1 ukoliko nije

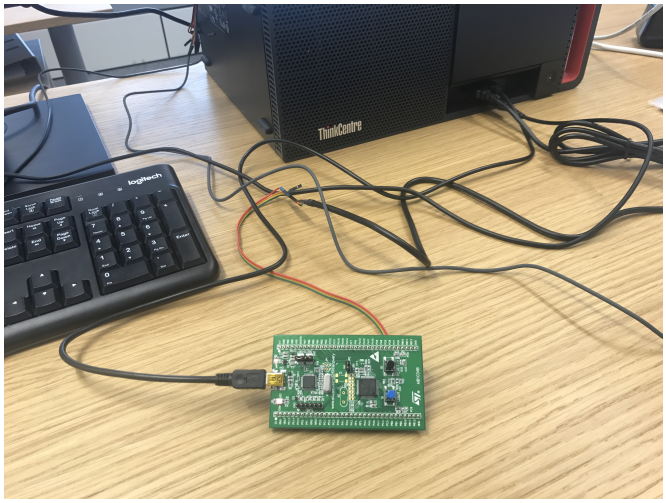
- `UART_AutoReceive` - funkcija za inicijalizaciju automatskog primanja podataka; omogućuje se prekidna rutina za serijsku komunikaciju; kao ulazne parametre prima port za serijsku komunikaciju, *callback* funkciju i njezin argument koji je pokazivač na međuspremnik u koji se spremaju primljeni podaci; funkcija kao izlazni parametar vraća -1 ukoliko je došlo do greške, a 0 ukoliko nije
- `UART_Control` - funkcija kojom se pokreće ili zaustavlja primanje podataka; kao ulazne parametre prima port za serijsku komunikaciju i operaciju koja se želi izvršiti (pokretanje ili zaustavljanje primanja podataka); kao izlazni parametar vraća 0 ako je operacija uspješno izvršena

6. Rad sustava

6.1. Zahtjevi sustava

Za korištenje aplikacije potrebno je imati operativni sustav Windows 10® i konfiguriran Windows Speech Recognition značajku koja je dostupna unutar Windows® operativnog sustava. Upute o konfiguraciji dostupne su na službenoj Microsoft stranici za podršku korisnicima [12]. Za uspješno korištenje prepoznavanja govora potrebno je imati instaliran jedan od ponuđenih paketa prepoznavanja govora za engleski jezik.

Tijekom korištenja aplikacije preporučuje se, a za desktop računala zahtijeva, korištenje slušalica s mikrofonom zbog što veće eliminacije pozadinskog šuma odnosno preciznijeg prepoznavanja govora, a time i ugodnijeg iskustva korisnika. Pri konfiguriranju Windows Speech Recognition značajke se preporučuje kratko trenirati samu značajku kako bi se omogućilo stvaranje što točnijeg glasovnog profila korisnika.

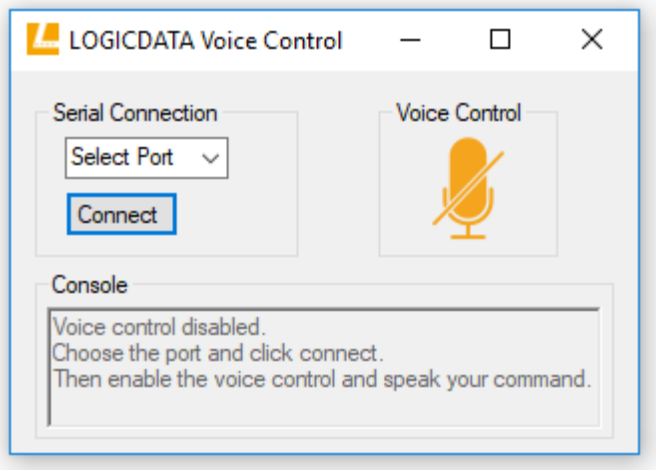


Slika 6.1: Discovery pločica spojena na osobno računalo

STM32F0 Discovery pločicu s programskom pogreškom je potrebno spojiti na USB priključak osobnog računala koristeći UART na USB čip, ili TTL-234X-5V USB na UART kabel koji je korišten u razvoju ovog sustava.

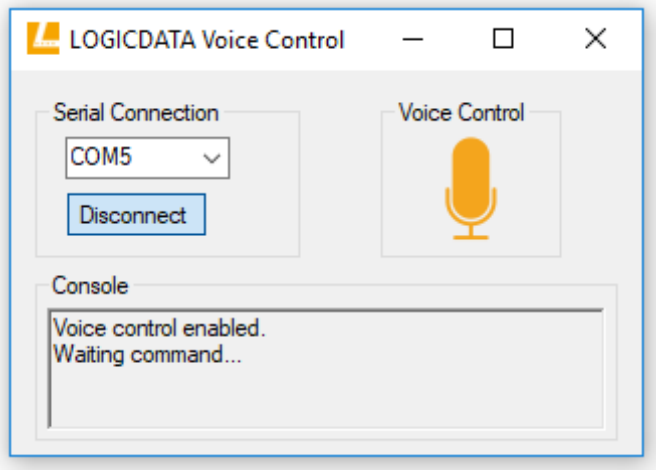
6.2. Pokretanje i primjer rada sustava

Sustav se pokreće klikom na izvršni program aplikacije uz koji se nalazi XML datoteka koja sadrži naredbe.



Slika 6.2: Inicijalno stanje aplikacije

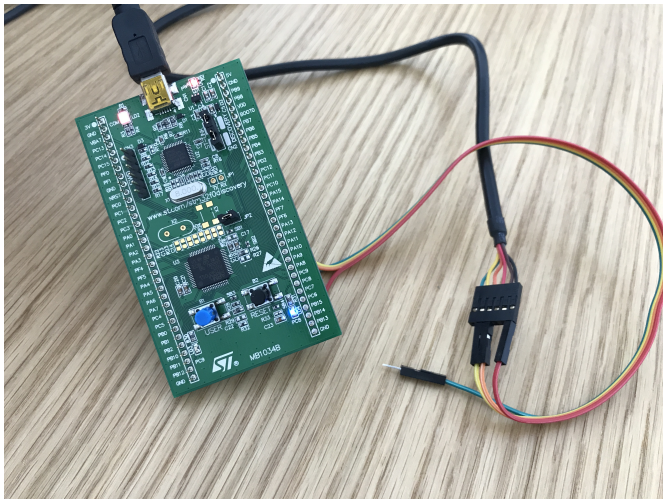
Nakon pojave GUI aplikacije potrebno je u padajućem izborniku izabrati serijski priključak preko kojeg je osobno računalo povezano s pločicom i kliknuti na gumb "Connect" kako bi se otvorio priključak. Klikom na gumb na kojem je prikazan mikrofon sustav je aktivan i čeka na glasovnu naredbu.



Slika 6.3: Aplikacija u stanju čekanja

Nakon što sustav prepozna naredbu, aplikacija sintetiziranim govorom ispituje korisnika je li naredba koju je sustav prepoznao zaista ona koja je bila izgovorena. Nakon potvrde korisnika naredba se serijskom vezom šalje na pločicu gdje se pali odgovarajuća svjetleća dioda, aplikaciji se šalje potvrda primitka i istovremeno s

izvršavanjem naredbe se sintetiziranim govorom javlja da se naredba izvršava. Nakon



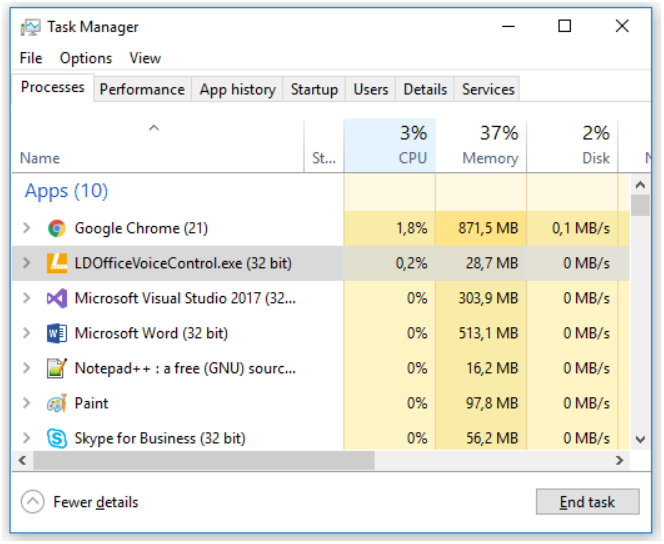
Slika 6.4: Izvršavanje "position one" naredbe, upaljena plava svjetleća dioda

izvršenja naredbe aplikacija se vraća u stanju čekanja. Ukoliko korisnik ne potvrdi naredbu, naredba se ne izvršava i aplikacija se vraća u stanje čekanja, a ukoliko korisnik opovrgne naredbu, aplikacija sintetiziranim govorom upućuje korisnika da ponovi glasovnu naredbu. Daljnji postupak je analogan prvom izgovaranju naredbe. Ukoliko se želi trenutno onemogućiti prepoznavanje govora, to je moguće napraviti ponovnim klikom na gumb mikrofona ili glasovnom naredbom "disable".

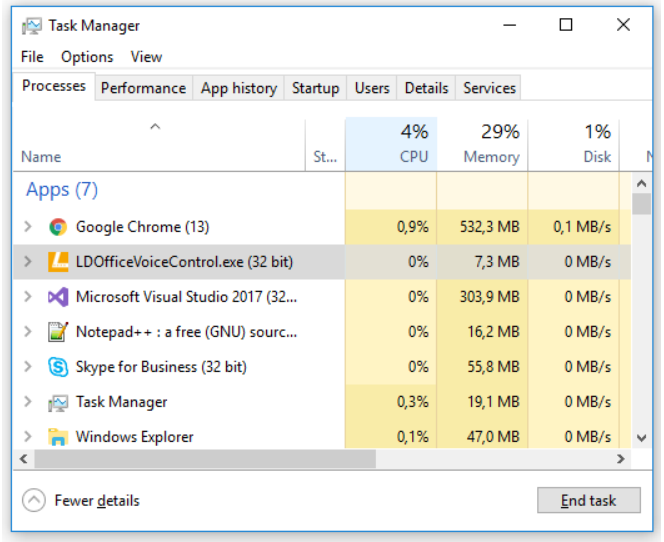
6.3. Performanse sustava

Upotrebom više dretvi postignuto je da potrošnja procesora tijekom rada aplikacije iznosi najviše 1%, gdje je tipična vrijednost prikazana na slici 6.5 dok potrošnja tijekom stanja čekanja iznosi 0%. Odgovarajuće dretve su aktivne samo tijekom rada, a ostatak vremena su privremeno zaustavljene što smanjuje nepotrebnu potrošnju procesora.

Točnost prepoznavanja izrečenih naredbi je vrlo visoka ukoliko se korisnik drži uputa za korištenje opisanih u poglavlju 6.1. Međutim, glavni nedostatak ove aplikacije je prepoznavanje naredbi ukoliko one nisu izrečene. Ovdje se referira na slučajno prepoznavanje riječi iz razgovora koje aplikacija prepozna kao naredbu zbog fonetske sličnosti. Ovaj nedostatak kao i slučaj prepoznavanja riječi iz razgovora koje definiraju naredbu su riješeni ugradnjom provjere izrečene naredbe gdje korisnik mora potvrditi naredbu koja je prepoznata. Na ovaj način se naredba izvršava tek kada je potvrđena čime se sprječava neželjeno izvršavanje naredbi, ali i produžuje vrijeme čekanja na izvršenje naredbe.



Slika 6.5: Maksimalna potrošnja procesora tijekom rada aplikacije



Slika 6.6: Potrošnja procesora tijekom stanja čekanja aplikacije

6.4. Moguća unaprijeđenja sustava

U okviru ovog diplomskog rada upravljanje vanjskim uređajem je demonstrirano paljenjem i gašenjem svjetlećih dioda na Discovery pločici gdje je zbog ograničenja sklopovlja implementirano vizualizacija 6 naredbi. Korištenjem naprednijeg sklopovlja ili dodatkom komponenata na postojeći broj naredbi se može proširiti. Krajnji cilj je razviti modul za obradu primljenih naredbi u podatke koje može iskoristiti određeni vanjski sustav kao što je primjerice kontrolna kutija pametnog stola. Također se za komunikaciju između desktop aplikacije i vanjskog uređaja mogu razviti dodatni moduli za komunikaciju koristeći druga sučelja umjesto UART-a. Bluetooth

komunikacijsko sučelje se, zbog toga što je riječ o bežičnoj komunikaciji, nameće kao logičan izbor za nadogradnju.

7. Zaključak

Koristeći tehnologiju prepoznavanja govora unutar Windows® operativnog sustava razvijena je desktop aplikacija koja prepoznaje glasovne naredbe korisnika i pretvara ih u naredbe koje mikrokontroler može obraditi. Naredbe se šalju serijskom komunikacijom mikrokontroleru koji ovisno o dobivenoj naredbi pali jedne od svjetlećih dioda frekvencijom koja odgovara primljenoj naredbi. Postignuto je da je XML dokument potpuno modularan s aplikacijske strane odnosno da je broj naredbi u dokumentu neograničen pod uvjetom da je format XML dokumenta zadovoljen. Uz minimalnu potrošnju procesora osobnog računala, uspješno je postignuto izvršenje naredbi na sklopovlju čime je ispunjen zadatak upravljanja vanjskim uređajem glasovnim naredbama. S druge strane, tijekom razvoja aplikacije pokazalo se da je potrebno uvesti provjeru prepoznate naredbe budući da aplikacija povremeno prepoznaje kao naredbe prepoznaje riječi koje su im fonetski slične. Također, kao problem se pokazuje i činjenica da korisnik može izgovoriti neke od skupina riječi koje čine naredbu u razgovoru koju aplikacija onda prepoznaje kao naredbu. Uvođenjem provjere ovaj problem se uklonio, ali je sam proces izvršavanja naredbi postao sporiji.

LITERATURA

[1] D. Kirasić, XML tehnologija i primjena u sustavima procesne informatike, 28th International Convention MIPRO, 2005.

[2] STMicroelectronics, STM32F051x8 Datasheet, 2015.

[3] About System Workbench for STM32, URL: https://www.openstm32.org/About+System+Workbench+for+STM32, pristupano: 2019-06-25

[4] R. Brown, Exploring New Speech Recognition and Synthesis APIs in Windows Vista, MSDN Magazine, 2006.

[5] Microsoft, The Windows Vista and Windows Server 2008 Developer Story: Speech, URL: https://docs.microsoft.com/en-us/previous-versions/bb875962(v%3dmsdn.10), pristupano: 2019-06-25

[6] J.McCaffrey, *Speech Recognition with .NET Desktop Applications*, MSDN Magazine, 2014.

[7] Microsoft, Microsoft Speech API (SAPI) 5.3, URL: https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms723627(v=vs.85), pristupano: 2019-06-25.

[8] Microsoft, Windows Speech Recognition Commands, URL: https://support.microsoft.com/en-us/help/12427/windows-speech-recognition-commands, pristupano: 2019-06-25

[9] Microsoft, Language and Region Support for the Speech Services, URL: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service
/language-support, pristupano: 2019-06-25.

[10] M. Hachman, The Windows Weakness No One Mentions: Speech Recognition, URL: https://www.pcworld.com/article/3124761/the-windows-weakness-no-one-mentions-speech-recognition.html, pristupano: 2019-06-25.

[11] L. Wood: Windows Speech Recognition vs Dragon NaturallySpeaking, URL: http://anewdomain.net/lamont-wood-windows-speech-recogniti

on-vs-dragon-naturallyspeaking-shootout/
[12] Microsoft, Use Voice Recognition in Windows 10, URL: https://support.microsoft.com/en-us/help/4027176/windows-10-use-voice-recognition, pristupano: 2019-06-25
[13] Microsoft, System Xml Namespace, URL: https://docs.microsoft.com/en-us/dotnet/api/system.xml?view=netframework-4.8, pristupano: 2019-06-25
[14] Microsoft, Grammar Class, URL: https://docs.microsoft.com/en-us/dotnet/api/system.speech.recognition.grammar?view=netframework-4.7.2, pristupano: 2019-06-25
[15] Microsoft, Speech Recognized Event Args Class, URL: https://docs.microsoft.com/en-us/dotnet/api/system.speech.recognition.speechrecognizedeventargs?view=netframework-4.8, pristupano: 2019-06-25
[16] Microsoft, Speech Recognition Result Class, URL: https://docs.microsoft.com/en-us/dotnet/api/system.speech.recognition.recognitionresult?view=netframework-4.8, pristupano: 2019-06-25
[17] Microsoft, Speech Synthesizer Class, URL: https://docs.microsoft.com/en-us/dotnet/api/system.speech.synthesis.speechsynthesizer?redirectedfrom=MSDN&view=netframework-4.8, pristupano: 2019-06-25
[18] Microsoft, Sytem IO Ports Namespace, URL: https://docs.microsoft.com/en-us/previous-versions/windows/embedded/hh423145%28v%3dmsdn.10%29, pristupano: 2019-06-25

Glasovno upravljanje vanjskim uređajem pomoću osobnog računala

Sažetak

U okviru ovog diplomskog rada razvijena je aplikacija za osobno računalno s operativnim sustavom Windows® 10 kojom se putem govornih naredbi upravlja vanjskim uređajem. Aplikacija je razvijena koristeći Microsoft tehnologiju za obradu i prepoznavanje glasa unutar Windows® 10 operativnog sustava u C++/CLI programskom jeziku i .NET okruženju. Realiziran je ugradbeni računalni sustav temeljen na razvojnom sustavu za STM32F0 mikrokontroler koji prima tekst s osobnog računala preko serijskog UART sučelja i potom izvršava i vizualizira odgovarajuću naredbu.

Ključne riječi: glasovno upravljanje, prepoznavanje govora, UART, Windows Speech Recognition, C++/CLI

Speech Control of External Device Using Personal Computer

Abstract

In this master thesis, a desktop application for Windows® 10 was developed that controls an external device using voice commands. The application was developed using Microsoft Speech Recognition technology available within the Windows® 10 operating system. Development was done in the C++/CLI programming language and .NET environment. Embedded system which receives a text from a personal computer via a serial UART interface and then executes and visualizes the corresponding command was developed. Development of the embedded system was done on the development board for the STM32F0 microcontroller.

Keywords: voice control, speech recognition, UART, Windows Speech Recognition, C++/CLI