Voice Commander

Jan Jasiński (lider, numer albumu: 285390), Jan Wilczek (numer albumu: 285433)

Link do repozytorium: https://github.com/Jasinsk/VoiceCommander

---

Raport 1 (21.12.17-04.01.18)

1. Cel projektu

Celem projektu jest stworzenie systemu wspomagającego produktywność (oraz wygodę) interakcji z komputerem poprzez wykorzystanie komend głosowych. Pozwalałby on na wykonywanie pewnych czynności bez potrzeby odrywania się od obecnego miejsca pracy. Jednocześnie pozwalałby na zwiększenie wygody podczas rekreacyjnego korzystania z komputera. Dodatkowo system taki użyty może zostać przez osoby starsze lub niepełnosprawne aby usprawnić obsługę komputera.

2. Proponowana architektura projektu

1. Main - Łączy działanie wszystkich klas.
2. NameListener - Nasłuchuje na pojawienia się klucza aktywacyjnego.
3. Recorder - Nagrywa komendę i wysyła ją do rozpoznania. Zwraca otrzymane wyniki.
4. CommandRecognizer - Przyjmuje wyniki z rozpoznania i patrzy czy coś z tego mogło byc komenda i z jakim prawdopodobieństwem. Zwraca komendę, która należy wywołać lub informację o braku rozpoznania.
5. CommandHandler - Przyjmuje komendę, którą nalezy wykonać i wykonuje przypisane do niej czynności.
6. TTS - Łączy się z jakimś zewnętrznym ttsem, mówi mu co ma powiedzieć i pozwala na odtworzenie wyniku.
7. TTSHandler - Obsługuje co i w jakich momentach ma byc mówione przez system.
8. **Powyższa architektura nie jest sztywną wytyczną w naszych działaniach** - planujemy najpierw stworzyć działający prototyp, który będzie w stanie rozpoznać jedną komendę w trzech wariacjach i na jego podstawie otrzymać szkielet projektu, na którym bazowalibyśmy w dalszych pracach.

3. Kwestie związane z obsługą systemu

1. Język obsługi systemu - polski.
2. Imię systemu - Powinno ono być zwięzłe i wygodne do wymawiania, charakterystyczne z punktu widzenia językowego, żeby nie było mylone z komendami, przyjazne dla użytkownika (pewne niedociągnięcia systemów są łagodniej traktowane w momencie, kiedy system ma jakąś osobowość) i zawierające się w słowniku systemu rozpoznawania mowy, którego będziemy używać. Niestety nie udało nam się na chwilę obecną znaleźć odpowiedniego imienia i w związku z faktem, iż nie będzie ono potrzebne w fazie prototypowej poczekać z podjęciem tej decyzji.

3. Aktywacja systemu - We wstępnej wersji programu będziemy podawać komendę przy włączeniu programu, żeby zobaczyć czy cały system działa. Docelowo trzeba wymyślić wygodne i sprawne rozwiązanie tego problemu. Wykorzystanie skrótu klawiszowego trochę niweczy cały sens systemu, bo skoro i tak trzeba naciskać klawize to o wiele łatwiej i sprawniej jest zaprogramować własne skróty klawiszowe. Aktywacja głosowa systemu wydaje się najsprawniejszym obecnie rozwiązaniem, jednak jako, iż jest to standardowe wyjście dla wszystkich wirtualnych asystentów sterowanych mową warto byłoby sie zastanowić czy nie wymyslimy czegoś innego co działałoby sprawniej dla naszego systemu.

4. Lista przewidywanych komend

1. Otwórz program X
2. Rozpocznij dyktowanie
3. Zakończ dyktowanie
4. Wyżej (scroll up)
5. Niżej (scroll down)
6. Góra (home)
7. Rzuć X na lewo/prawo (screen split)
8. Kopiuj
9. Wklej
10. Zapisz
11. Cofnij
12. Otwórz stronę X (w nowej karcie)
13. Zgoogluj "coś"

5. Przewidywane problemy

1. Czas reakcji systemu - nagrywanie, łączenie się z serwerem, wysyłanie pliku audio, czekanie na wynik, rozpatrywanie czy coś jest komendą, wykonywanie komendy, wysyłanie tekstu do tts, otrzymywanie pliku audio, odtworzenie go - wszystkie te czynności będą zajmowały znaczącą ilosć czasu. Czas reakcji systemu nie może wynosić więcej niż kilka sekund jeżeli ma on działać sprawnie.
2. Text-to-speech - Byłoby to bardzo dobre rozwiązanie, jednak ciężko powiedzieć czy jego implementacja nie spowolni dodatkowo systemu. Możliwe, że dla tak ograniczonego systemu bardziej korzystne będzie nagranie wszystkich odpowiedzi i odgrywanie ich z plików.
3. Źródło audio - Konieczne będzie rozpoznanie jak jakość nagrywanego i wysyłanego audio wpływa na skuteczność i czas reakcji systemu rozpoznania mowy.
4. Aktywacja systemu - Jeżeli wykorzystana zostanie aktywacja głosowa systemu przy pomocy komendy startowej to trzeba rozplanować jak to zrobić aby jednocześnie działało to sprawnie, ale również nie wykorzystywało zbyt duzej ilości zasobów komputera podczas nasłuchiwania. Ciagłe nagrywanie audio, wysyłanie je na serwer i oczekiwanie na odpowiedź wydaje się bardzo nieoptymalnym rozwiązaniem. Pierwszym krokiem jaki trzeba wykonać to stworzenie bramki, która uruchamiać będzie analizę nagrywanego materiału. Najprostszym rozwiązaniem jest analiza natężenia dźwięku i analizowanie sygnału dopiero po wyraźnym jego skoku. Zastanawiamy się nad wykorzystaniem systemu, który stworzyliśmy przy okazji miniprojektu, w celu usprawnienia

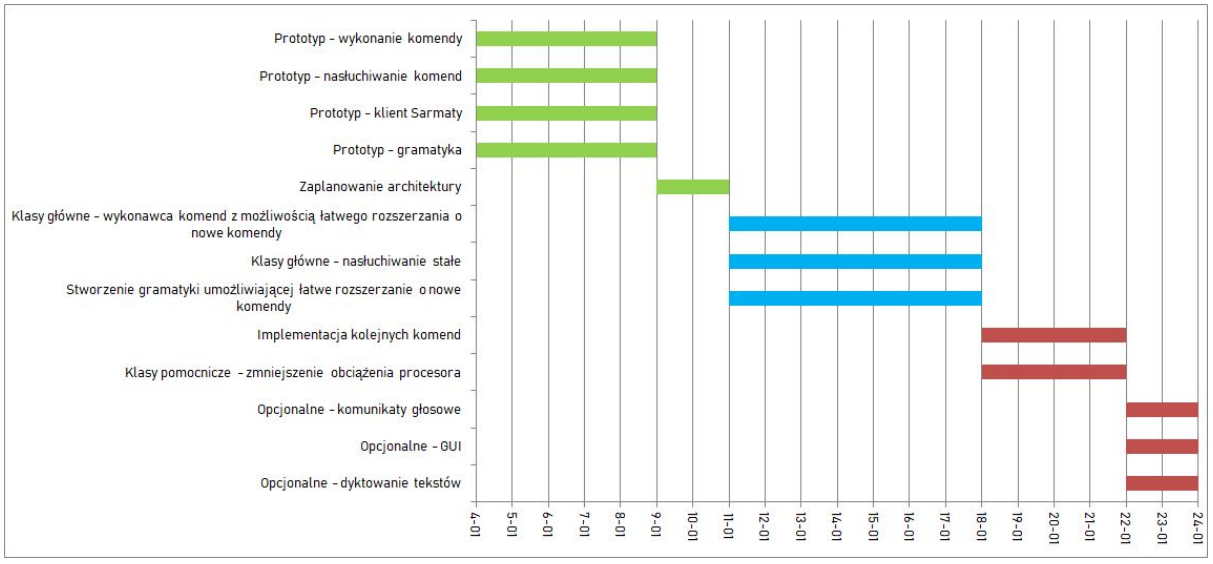
rozpoznawania komendy aktywizacyjnej. Stworzenie wewnątrz systemu programu, który wytrenowany byłby do rozpoznawania komendy startowej zwolniłoby nas od potrzeby wysyłanie danych do serweru przy każdej sytuacji. Dopiero po zatwierdzniu hasła startowego system zaczynałby wysyłanie nagrywanego audio w celu rozpoznania mowy.

6. Wykorzystywane systemy z uzasadnieniem

W naszym projekcie użyjemy głównie systemu Sarmata do rozpoznawania mowy z zadaną gramatyką, gdyż nasz system będzie rozpoznawał konkretne komendy, które użytkownik powinien znać, aby poprawnie korzystać z systemu. Dodatkowo planujemy użycie serwisu Dictation do dyktowania tekstu (na przykład do pliku tekstowego) oraz zastanawiamy się nad użyciem serwisu Trybun do syntezy mowy, aby przedstawiać komunikaty systemu także w postaci mowy.

7. Plan pracy

Jest to ramowy plan, który będzie uszczegóławiany w czasie trwania projektu. W kolejnym etapie prac zadania dotyczące klienta Sarmaty i gramatyki prototypu wykonywał będzie Jan Jasiński, a kod obsługujący nasłuchiwanie i wykonywanie komend napisze Jan Wilczek.



Raport 2 (04.01.18-11.01.18)

1. Postęp prac według wytycznych planu

Z powodzeniem zaimplementowano prototyp systemu, który po włączeniu programu rejestruje jedną komendę, którą może być "uruchom x", gdzie x jest jednym z następujących programów: Google Chrome, Eksplorator Windows lub notatnik, po czym włączany jest odpowiedni program. Prototyp pozwolił na skuteczne zaprojektowanie architektury (bardzo przypominającej tę zaproponowaną w ubiegłym tygodniu). Powstała wstępna gramatyka i wstępna implementacja

klasy CommandHandler. Porównując z rozpiską na diagramie Gantta plan na ten tydzień został wykonany w 100%.

2. Napotkane problemy

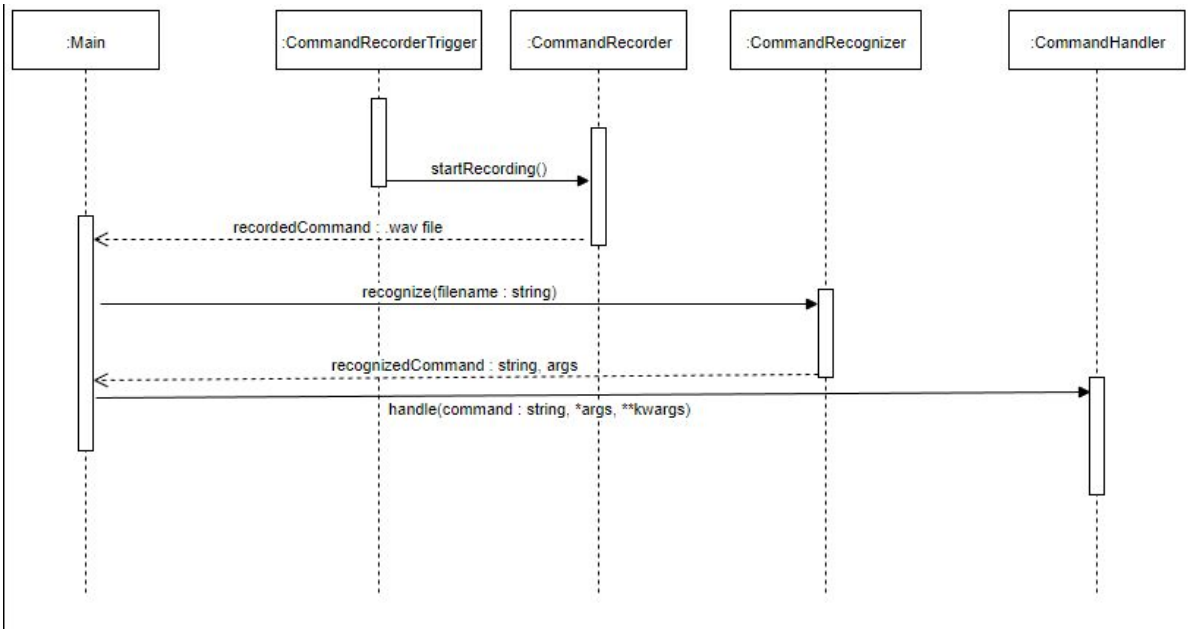
1. Biblioteka pywinauto - jest ona odpowiedzialna z obsługę systemu operacyjnego za pomocą skryptów Pythona. Podczas ładowania jej jako normalny użytkownik (na przykład po włączeniu wpisywania linijka po linijce w PowerShellu za pomocą polecenia python) proces nigdy się nie kończy. Dopiero uruchomienie polecenia jako administrator skutkuje pozytywnym rozwiązaniem. Próbowaliśmy to obejść stosując zewnętrzny skrypt, który w trakcie działania programu prosi użytkownika o dostęp do praw administratora (wyświetlane jest okienko podobnie jak podczas instalacji programu w systemie Windows), ale niestety bezskutecznie. Tymczasowym obejściem jest uruchomienie edytora PyCharm jako administrator.

3. Pomysły i sugestie

1. Hasło uruchamiające system - mogłoby być nagrane wcześniej przez użytkownika, a system rozpoznawałby je na zasadzie podobieństwa sygnałów, bez użycia ASR - dzięki temu można by znacznie skrócić czas reakcji systemu i obciążenie procesora.

4. Architektura systemu przedstawiona za pomocą diagramu sekwencji

Architekturę jest przedstawiona za pomocą diagramu sekwencji języka UML, gdyż najlepiej widać działanie naszego systemu w przypadku użycia - jeśli interakcję użytkownika określi się jako "komendę" diagram przedstawia praktycznie wszystkie możliwe use case'y.



5. Plan na następny tydzień

Po pierwszym tygodniu prac nie zauważyliśmy konieczności aktualizowania diagramu Gantta, zatem w następnym tygodniu będziemy pracować nad zorientowaną już obiektowo podstawową implementacją naszego systemu. Chcemy, aby system po kolejnym etapie prac był w stanie:

1. rozpocząć nagrywanie w wyniku komendy głosowej,
2. nagrać komendę i zapisać ją w pliku
3. poprawnie rozpoznać komendę używając zewnętrznego systemu ASR
4. wykonać komendę lub wyświetlić komunikat o nieznalezieniu odpowiedniej komendy

Oprócz tego mamy nadzieję rozwiązać wszystkie powstałe dotąd problemy.

Raport 3 (11.01.18-18.01.18)

1. Postęp prac

W całości zostały zaimplementowane klasy CommandRecorder i CommandHandler. Ta pierwsza może nagrywać w dwóch wariantach: albo krótkie nagranie o określonym czasie (komenda), albo nagrywanie ciągłe z możliwością przerwania (dyktowanie). Ta druga posiada obsługę następujących komend:

1. Włącz program
2. Powrót (home)
3. Pokaż pulpit
4. Scrollowanie w górę
5. Scrollowanie w dół
6. Zaznacz wszystko
7. Kopiuj
8. Wklej
9. Zapisz
10. Cofnij
11. Zamknij okno

Parę innych komend jest w stanie prac (na przykład dyktowanie albo googlowanie danego hasła). Poza tym główna pętla programu została wzbogacona o interfejs graficzny, który składa się z pola z komunikatami aplikacji oraz przycisku, który po naciśnięciu zaczyna cały proces nagrywania, rozpoznawania i obsługiwania komend. Zaletą tego rozwiązania (stworzonego przy pomocy pakietu tkinter) jest to, że program nie zużywa praktycznie w ogóle procesora, ale czeka na interakcję użytkownika. Jest to rozwiązanie tymczasowe, ale bardzo prawdopodobne, że końcowy program będzie z niego częściowo korzystał. Została również ustalona ostateczna struktura gramatyki systemu. Ostateczny jej zakres jednak jest wciąż opracowywany w związku wciąż zmiennym zbiorem obsługiwanych komend. Zostałe również udoskonalone klasy związane z klientem Sarmaty. Niestety w momencie ich kończenia pojawił się problem z łącznością z serwerem Sarmaty, który uniemożliwił nam ostateczną ocenę skuteczności tych zmian.

2. Problemy

Rozwiązaniem problemu nasłuchiwania może być użycie biblioteki AUDIo TOKenizer (https://github.com/amsehili/auditok), która w czasie rzeczywistym rozpoznaje zdarzenia akustyczne, a także pozwala zapisać je do pliku i następnie obsłużyć callbackiem. Możnaby mieć ustawiony "tryb pracy" z aplikacją, kiedy nasłuch byłby włączony i obsługiwane byłyby komendy na bierząco oraz "tryb bierny", między którymi można by się przełączać używając przycisku obecnego już w GUI. Nie powinno to być uciążliwe dla użytkownika, a dużo prostsze dla nas w implementacji.

Aby rozwiązać nasz problem z prawami administratora należy uruchomić główny skrypt jako administrator - wystarczy to do poprawnego działania aplikacji. Nie będziemy szukać innego rozwiązania w tej kwestii.

3. Ocena dotychczasowej pracy

Niestety w tym tygodniu prace nie posunęły się znacząco do przodu - głównie z powodu zbliżającej się sesji i dużej liczby zaliczeń. Aplikację będziemy prawdopodobnie kończyć na początku następnego tygodnia tak, aby podczas oddania działała ona prawidłowo w swojej podstawowej postaci. Niestety nie mając doświadczenia w podobnych projektach często jesteśmy niepewni, w którą stronę rozwinąć nasz system, co jest tak naprawdę ważne. Pewnym niepokojącym faktem jest obecnie duże rozproszenie systemu - przez opóźnienia pewne jego części są gotowe w większym stopniu niż inne, a sposoby ich komunikacji mogą być niekompatybilne, co łamie zasady inżynierii oprogramowania. Możemy mieć tylko nadzieję, że następny tydzień pozwoli nam na większe postępy i możliwość zaprezentowania gotowego produktu podczas oddania.

Nie wszystkie zadania z planu Gantta zostały zrealizowane w tym tygodniu.

4. Plan na następny tydzień

W kwestii pisania kodu największy nacisk będzie położony na rozpoznanie komendy przez system ASR (gramatyka i klient Sarmaty) oraz na poprawną interpretację komendy. Powodzenie w implementacji tych modułów zadecyduje o powodzeniu całego projektu. Oprócz tego musimy przygotować się na obronę projektu - prawdopodobnie przedstawimy głównie przypadki użycia naszego systemu, komentując pobieżnie jego architekturę.

Raport końcowy (18.01.18-25.01.18)

1. Ostatni etap prac

Podczas tego tygodnia udało rozwiązać się najbardziej palący problem - obsługę komend w czasie rzeczywistym wraz z ich wykrywaniem. Z pomocą przyszła biblioteka Auditok znaleziona w zeszłym tygodniu (Raport 3, punkt 2). Umożliwia ona nasłuchiwanie zdarzeń akustycznych na podstawie kryterium energetycznego i ich obsługę za pomocą callbacku. Co najważniejsze,

działa ona w sposób asynchroniczny, dzięki czemu program nie musi być aktywnym oknem, aby komendę nagrać i obsłużyć. Zwiększa to przyjemność w użytkowaniu naszego systemu jak i również jego praktyczność, w związku z faktem, iż wydawane komendy działają w kontekście w jakim zostaną wydane, niezależnie od tego nad czym obecnie pracuje użytkownik. Zastosowanie tej biblioteki wpłynęło również korzystnie na wydajność. Obecnie program, prowadząc stały nasłuch, zabiera jedynie 60-70 MB pamięci RAM, przy zużyciu procesora rzędu 0,2%. Jest to bardzo dobry wynik, z którego jesteśmy bardzo zadowoleni, zwłaszcza, że nie spodziewaliśmy się tak wielkich korzyści przy tak prostym rozwiązaniu.

Oprócz tego sukcesu udało nam się zintegrować cały nasz system, co również stanowiło pewne wyzwanie, gdyż do tej pory jego składowe były mocno rozproszone i zagrażały spójności. Nie przysporzyło to większych problemów, a raczej wręcz przeciwnie: modularność systemu okazała się jego największą zaletą. Została zachowana zasada ortogonalności, dzięki czemu składowe mogły być użyte wielokrotne. Należy to podkreślić, gdyż jest to dobrą praktyką w projektowaniu dużych systemów. Jednym z przykładów jest moduł logujący. Obecnie każdy komunikat do użytkownika jest do niego przekazywany, dzięki czemu można w przyszłości dowolnie zmieniać formę komunikatów, bez konieczności ingerencji w cały system.

Naszą pracę zwieńczyliśmy dodając kolejne komendy obsługiwane przez system. Okazało się to bardzo wygodne i proste, co ma w sobie duży potencjał na dalsze rozszerzanie programu.

2. Podsumowanie planu Gantta

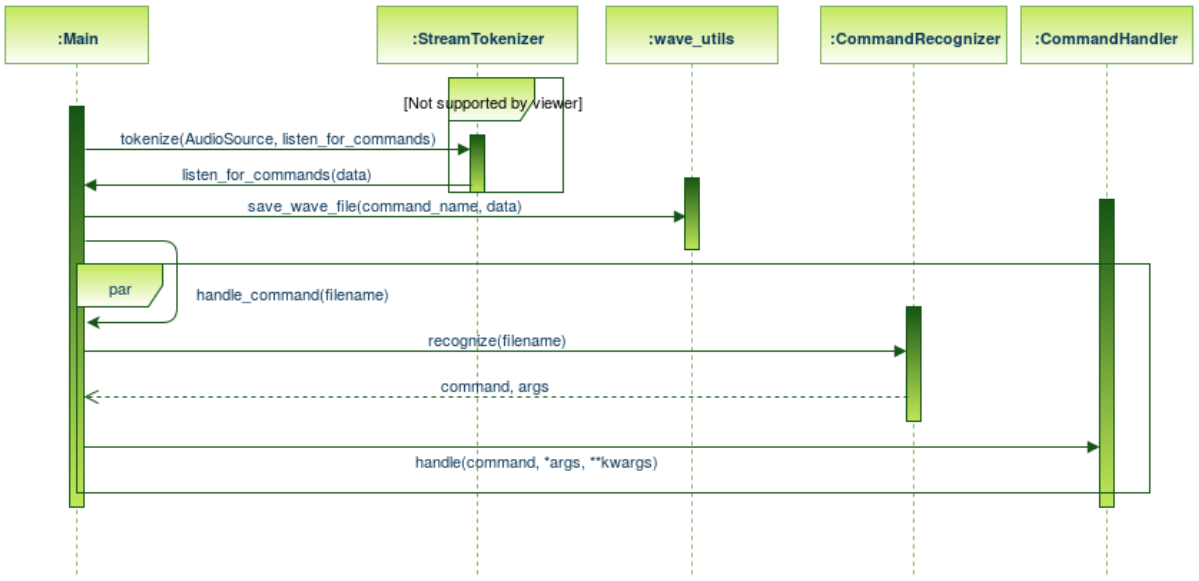
Dokończyliśmy zadania z poprzedniego tygodnia - implementację głównych klas. Powstała gramatyka, która w połączeniu z modułem odpowiedzialnym za wykonywanie komend pozwala na bardzo proste dodawanie nowych rozkazów (wprowadzenie nowej linijki do gramatyki wraz z aliasem komendy i kod źródłowy w jednym miejscu, obsługujący ją pod tym aliasem). Jeśli chodzi o zadania z obecnego tygodnia, to również wykonaliśmy całość założeń. Zmniejszyliśmy obciążenie procesora, wprowadziliśmy możliwość dyktowania 10-sekundowych tekstów, a także zaimplementowaliśmy obsługę kolejnych komend.

Nie stworzyliśmy ani komunikacji głosowej systemu, ani GUI nie z powodu braku czasu, ale dlatego, że oba te elementy wydały nam się zbędne. Program jest uruchamiany z konsoli a za komunikację z użytkownikiem odpowiada moduł logujący, który korzysta z niej do wyświetlania komunikatów. Przez pewien czas mieliśmy w naszym projekcie GUI na podstawie modułu tkinter, ale zrezygnowaliśmy z niego, gdyż występował problem z wątkowaniem, a konsola nadaje się do tego dużo lepiej - koniec końców chodzi o okienko z linijkami tekstu. Komunikacja głosowa wydała nam się zbędna, gdyż komunikaty typu "Rozpoznano komendę X" albo "Nie rozpoznano komendy" tylko użytkownika denerwują, spowalniając system, natomiast wynik komendy widoczny będzie w systemie, a transkrypcja z dyktowania i tak musi być gdzieś wyświetlona, zatem moduł ten wydał nam się zupełnie niepotrzebny.

Tym sposobem wykonaliśmy 100% planu Gantta. Patrząc na cały okres wykonywania projektu stwierdzamy, że praca była dobrze rozplanowana. Co prawda w zeszłym tygodniu opóźniła się ona nieco, ale w tym udało nam się nadrobić wszelkie zaległości i ukończyć wszystkie zadania z początkowej rozpiski.

3. Ostateczna architektura systemu

Tak jak poprzednio, architekturę najlepiej przedstawić na diagramie sekwencji.



4. Nieosiągnięte cele projektu i inne problemy

- Nie wszystkie komendy, o których była mowa w fazie planowania programu, został zaimplementowane. Było to jednak świadome działanie, wynikające z faktu, iż nie wszystkie komendy okazały się praktyczne do wykonywania przy pomocy poleceń głosowych. Uznaliśmy jednocześnie, iż obecny zasób komendowy w pełni wystarczy aby zaprezentować program.
- Ostateczną skuteczność systemu w rozpoznawaniu komend uznajemy za dobrą, jednak zdecydowanie byłby to aspekt, który wymagałby pracy w przypadku dalszego rozwijania systemu. Jesteśmy przekonani, iż wykorzystywanie zewnętrznego zewnętrznego mikrofonu, wraz z obróbką DSP sygnały przed wysłaniem do rozpoznania pozwoliłyby poprawić ten aspekt.
- Występują pewne problem z stabilnością i powtarzalnością systemu, związane głównie z buforem, który z obecnie nieznanego nam powodu czasami zapełnia się w sposób, który znacznie utrudnia pracę z systemem. Jest to jednak problem wystąpujący periodycznie, którego nie udało nam się dotychczas rozwiązać.
- Napotkaliśmy pewne ograniczenia związane z możliwościami biblioteki Pywinauto, które uniemożliwiły nam implementację pewnych komend i funkcjonalności. Ograniczenia te bardzo łatwo obejść wykorzystując dodatkowy zewnętrzny program (np. Autohotkey), w którym uruchamiane byłyby skrypty poprzez nasz program. Takie rozwiązanie dałoby nam praktycznie nieograniczone możliwości jeżeli chodzi o obsługę systemu Windows, jednak w celu zachowania prostoty naszego projektu nie zdecydowaliśmy się na implementację takiego rozwiązania.

5. Osiągnięte cele i końcowa ocena projektu

Główny cel projektu, jakim było stworzenie działającego programu umożliwiającego obsługiwanie pewnych czynności przy pomocy komend głosowych w celu ułatwienia współpracy z komputerem, uważamy za wykonany. Stworzyliśmy system, który prawidłowo rozpoznaje i wykonuje komendy głosowe, w sposób pozwalający zauważyć jego skuteczność oraz przydatność w praktycznym zastosowaniu przez potencjalnego konsumenta. Uważamy, iż prezentowany przez nas program może służyć jako prototyp, który po rozwinięciu miałby potencjał rynkowy. Zarówno koncepcję naszego systemu jak i jego implementację uznajemy za sukces.