**Функціональні вимоги**

Орієнтовані на процес:

1. Система має дозволяти запуск без запуску сторонніх програм.

2. Система має генерувати звук достатньої гучності і дозволяти зміну цієї гучності.

3. Система має дозволяти отримувати текст,який був згенерований в звук і звуку який був розпізнаний.

4. Система має працювати з різними вимовами.

5. (опція) Система має надавати можливість міняти голос яким буде виводитися інформацію.

6. Система повинна правильно обробляти інформацію та давати правильну відповідь.

Орієнтовані на інформацію:

1. Система має видавати інформацію в зрозумілому вигляді (текст + звук)

2. Система має правильно реагувати на неправильну відповідь.

3. Система має обробляти інформацію за прийнятний час.

**Нефункціональні вимоги**

1. Система має бути доступною після запуску roscore на довільній операційній системі Ubuntu або подібних.

2. Система повинна припиняти роботу при певній команді і правильно завершувати роботу при ручному вимкнені терміналу.

3. Система має працювати на довільних ОС типу Ubuntu за умови якщо на ній є ROS Lunar або вище.

4. Система має бути доступна для довільної людини яка володіє англійською мовою.

**JSON**

JSON — це один з найбільш зручних форматів даних на основі синтаксису JavaScript. Він дає можливість кодувати і декодувати інформацію в зручному форматі.

Дані в форматі JSON являють собою:

- JavaScript об’єкти {...}

- Масиви [....]

- Значення одного з типів:

- Текстовий рядок

- Число

- Логічне значення

- null

**json.dump**(obj, fp, \*, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, default=None, sort\_keys=False, \*\*kw) — Серіалізує obj як відформатований за JSON потік до fp.

**json.dumps**(obj, \*, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, default=None, sort\_keys=False, \*\*kw) - Серіалізує obj в відформатований за JSON рядок використовуючи спеціальні таблиці. Аргументи мають те саме значення що й в dump()

**json.load**(fp, \*, cls=None, object\_hook=None, parse\_float=None, parse\_int=None, parse\_constant=None, object\_pairs\_hook=None, \*\*kw)

Десеріалізує fp ( .read() - підтримуючий об’єкт-файл,що містить JSON документ) в Python об’єкт використовуючи спеціальні таблиці.

**json.loads**(s, \*, encoding=None, cls=None, object\_hook=None, parse\_float=None, parse\_int=None, parse\_constant=None, object\_pairs\_hook=None, \*\*kw)

Десеріалізує s (текстовий байтовий об’єкт,що містить JSON документ ) в Python об’єкт використовуючи спеціальні таблиці.

**XML**

XML — розширювана мова розмітки. Використовується для опису ієрархії структур даних.

XML-документ має ієрархічну логічну структуру, і може представлятись у вигляді дерева. Вузлами цього дерева можуть бути:

* елементи, фізична структура яких складається із:
  + коректної пари відкриваючого та закриваючого тегів (<*Назва-тега*>) та (</*Назва-тега*>), або
  + тега порожнього елемента (<*Назва-тега*/>),
* Атрибути, що мають вигляд пар ключ-значення (*назва атрибута*="*значення атрибута*") і знаходяться або у відкриваючому, або у порожньому тезі.
* Вказівки щодо обробки документа
* Коментарі
* Текст, або у вигляді простого тексту, або фрагментів CDATA

XML-документ повинен мати лише один кореневий елемент. Решта елементів є піделементами цього кореневого елемента.

Деякі веб-браузери здатні безпосередньо відображувати XML-документи. Це може досягатись шляхом застосування таблиці стилів. Вказані у таблиці стилів операції можуть призводити до перетворення XML-документа в інший, відмінний від XML формат.

**HTML parser**

HTML parser служить для як база для парсингу текстових файлів які формовані в HTML і XHTML.

Даний парсер містить наступні методи:

HTMLParser.**feed**(data) — Віддає дані на обробку парсеру. Обробка здійснюється доти доки є інформація для обробки. Опісля викликається метод close()

HTMLParser.**close**() - Обробка всі збережених даних,примусова,подібна до читання файлу до позначки його закінчення.

HTMLParser.**reset**() - Заново створює об’єкт. Втрачає всі неопрацьовані дані.

HTMLParser.**getpos**() - Повертає номер поточного рядка і положення в ньому.

HTMLParser.**handle\_starttag**(tag, attrs)

HTMLParser.**handle\_endtag**(tag)

HTMLParser.**handle\_startendtag**(tag, attrs)

Методи,які викликаються для обробки початку тегу,кінця тегу та обох варіантів відповідно.

**Опис даних**

Система буде працювати з ROS API яка необхідна для створення будь-якого проекту цього роду. З даного API найважливішими є бібліотеки rospy та pocketsphinx.

Після забезпечення наступного імпорту:

import rospy

from std\_msgs.msg import String

Буде доступно використання всіх необхідних функцій. Також будуть використовуватися бібліотеки sound\_play.

Приклад використання в програмі:

**def** talker():

rospy.init\_node('talker', anonymous=True)

soundc = SoundClient()

rospy.sleep(1)

soundc.say('Hello Player. I want to play with you! You have no choice')

rospy.sleep(10)

Та для pocketsphinx:

**def** start\_recognizer(self):

# initialize pocketsphinx. As mentioned in python wrapper

rospy.loginfo("Initializing pocketsphinx")

config = Decoder.default\_config()

rospy.loginfo("Done initializing pocketsphinx")

# Hidden Markov model: The model which has been used

config.set\_string('-hmm', self.lm)

# Pronunciation dictionary used

config.set\_string('-dict', self.lexicon)

# Keyword list file for keyword searching

config.set\_string('-kws', self.kw\_list)

rospy.loginfo("Opening the audio channel")

Це частина коду тому вона не є функціональною.