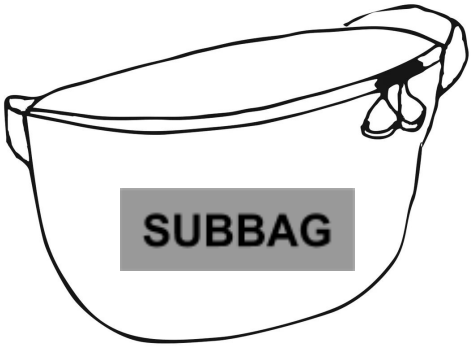
TRAN Kinh-Vinh
LI Zixuan
Sabri M'BARKI
Nathan KREMER
Thomas CARDONNE
Master 1

Internet of Things - Project



Date of Delivery : 17/04/2019

Professor : Mr. HAMIDI
Promotion 2020

OUTLINE

Introduction

Nowadays, we are 7.5 billion in this whole planet. And what we thought English would be the universal language, it's only spoke by 1,3 billion of people in the world. Communication is now in the center of each relation we have between others individuals or communities. It's the core to break the glass and make friends, convince others in order to make them follow your opinions or stand out for what you think is right. However, what happened if this communication is going on one side only because the others can't understand you ?

We thought about this problematic with the barrier of the language and the misunderstanding as we encountered this situation already once in our life. And tried to find out what would be the best solution to it.

From the different labs we did and also the state of the art for our idea, we finally agreed to build an IoT object in order to make the communication better in any language. That's how we created Subbag.

In this report we will go through the different steps we've been through in order to create this object, what we based ourselves from what has been done already and what can be our further implementation, optimisation of our object.

I/ Context

--- Find an universal communication tools

We based ourselves with what we're doing in our daily lives : Watching Series,Movies. And as pure movies amateurs, we always want our series to be in the original voices because we don't appreciate having a french voiceover that make the situation in the movie losing their intensity. That's why we always put the subtitles in order to still understand the context but also feel the non verbal communication.

We wanted to transpose this idea in our IoT object. As Speaking permits to show directly our feelings, Reading permits to understand the situation better, to get to know what is going on clearly if we don't get to understand what it's been saying.

II/ State of the Art

Before starting directly our project, we had to really analyze what has been done already in this field. Lots of resources we found from our researches that really helped a lot into the realisation of our SubBag.

First, finding a way to transcribe what we're saying. We found an application that directly transcript what we saying and translating it, it's an Web application taking as input our voice and as an output and transcript that is translated in the language of our choice :

http://www.authot.com/fr/2015/11/26/traduction-en-direct/

After we really wanted to know how does Transcription can work and what are the context of using it :

https://surdifrance.org/info-par-theme/accessibilite/237-metier-transcripteur-en-simultane

From this community of transcriptors for deaf people, we really agreed on doing transcription as it can solve a lot situation where the person is deaf or have some malfunctions to listen.

To follow up with the context of Transcription, we did researches to know how to get a voice with the resources that we have : an ESP 32.

We find some good API in order to transcript our voices :

- Microsoft API :

https://www.nexmo.com/blog/2018/03/14/speech-voice-translation-microsoft-dr/

- Google

https://cloud.google.com/speech-to-text/

https://cloud.google.com/translate/

In our researches, we find a Voice Recognition device created with arduino :

https://www.developpez.net/forums/d1753780/general-developpement/programmation-systeme/embarque/arduino/utiliser-reconnaissance-vocale-reconnaitre-bruits/

Basically they re analyzing your voice saying a color and the same color is lighted up. We can use the different schema and the way they connect the arduino to the PC in order to make it work.

After we were wondering how can we do the Connection to Wi-Fi :

https://letmeknow.fr/blog/2013/08/26/tuto-realiser-une-liaison-wifi/

In this article, we get to know how to connect an arduino to Wifi, but at the end we tried to get from the ESP32 as it has already a Wifi Card.

- Connection to the Phone - How to connect to your phone (with Wifi) :

https://www.youtube.com/watch?v=CV48QxsH0HQ

https://www.makeuseof.com/tag/6-easy-ways-connect-arduino-android/

- Needs to search for the different sensors compatible with arduino :

https://robokits.co.in/sensors/voice-recognition-module-arduino-compatible

https://fr.rs-online.com/web/p/products/1743250/?grossPrice=Y&cm_mmc=FR-PLA-DS3A-_-google-_-PLA_FR_FR_CATCHALL-_-Catch+All+Ad+Group-_-PRODUCT_GROUP&matchtype=&pla-293946777986&gclid=Cj0KCQiAh9njBRCYARIsALJhQkG4rD6eXcgCew_XKoE7RBULQ1WSJ5tNEnOQQEnK7YpJerlOkPOg_DsaAq6VEALw_wcB&gclsrc=aw.ds/

- Tutorial to how we can do it :

https://www.imagesco.com/articles/hm2007/SpeechRecognitionTutorial01.html

https://www.youtube.com/watch?v=Ur1tzMDP97g&vl=en

Need to choose how much time per second word length to how much every phrase are gonna be (.96 second word length (40 word vocabulary) or the 1.92 second word length (20 word vocabulary)

III/ Description of SubBag

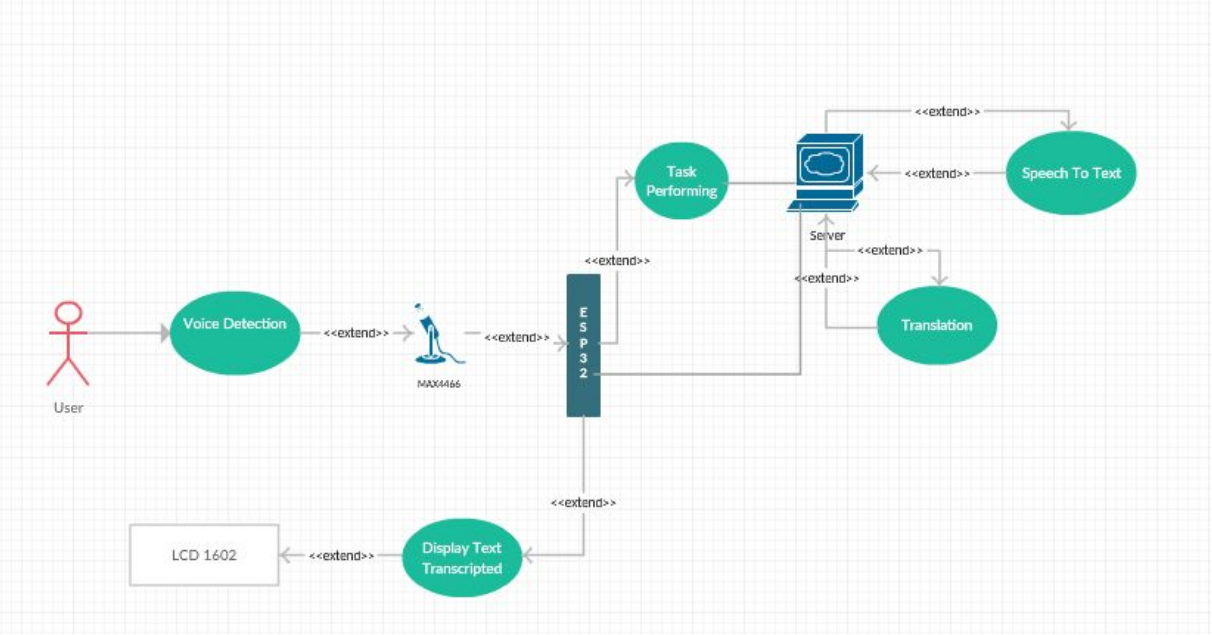
-- An innovative way for communication

SubBag is the new innovative way to communicate with strangers. Currently, SubBag allows people to communicate with different languages, how? Simply thanks an audio recorder, and a screen. The SubBag will register your sentence and translate in the language you want, for finally display it, transcript it on your SubBag screen. As what we talked about before, SubBag just like a subtitle on your bag, help others who speak different language to understand you and also help people who are deaf or has audition problems.

SubBag is also really respectful of the environment, cause is done with ecological tissues. In fact, SubBag has to be implemented to a handy items that we can bring in our daily life. We first thought about choosing a T-Shirt but the T-shirt has to be washed quite often and with our device, it's too complicated because it's not waterproof. That's why we choose to take a Bum bag, trending nowadays, to really make it as much ergonomic as possible.

-- Functional analysis

We chose to describe the functional analysis with a UML Diagram.



First we have the voice detection from the MAX4466, after that it send directly to the esp32 that then send to the server in Python using the API google to Speech to Text our sounds. Then it does another Request to do the traduction and the output goes to the ESP so it can be display with the LCD 1622.

IV/ Implementation

-- Introduction

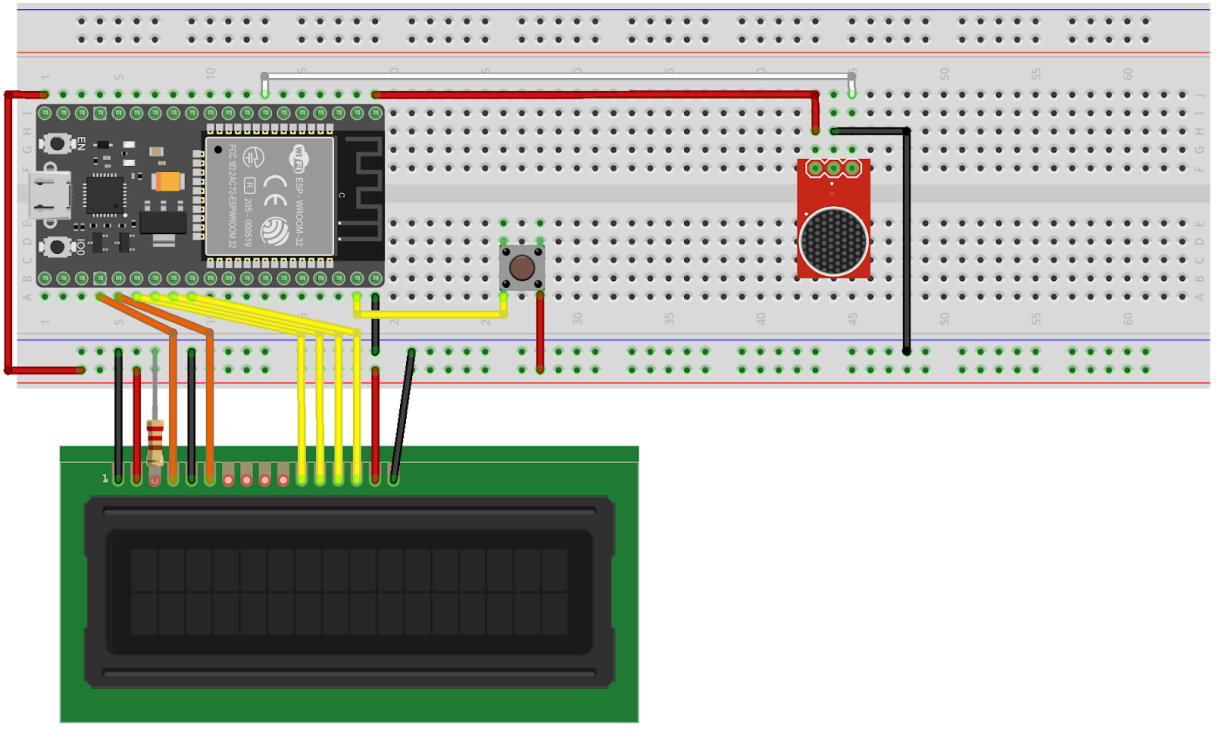
Our strategy to implement this project was to try separately each component and provide some kind of interface to interact with it. For example, we tried the LCD display without other components and we built functions to print some text in the LCD.

This allowed us to create single-responsibility files. For instance, the display.ino file is in charge of handling all LCD operations and provides a setup function and a display function. This permits everyone in the group to print something in the LCD without knowing how it works internally.

The other important point in the implementation details, is that we decided to use a server, more or less for the same reasons : the server handles all the “heavy” operations and provides a convenient HTTP interface for making speech recognition and text translation.

-- Board Development

At the hardware level, we have the ESP32 which is connected to an LCD 16x2 display, a push button and a microphone.



We developed the ESP32 using the ESP32 Arduino IDE Plugin.

Main Structure

The subbag.ino contains the backbone of the project and glues all components together.

The implementation is pretty straightforward, and the only tricky part of this file is the handling of events.

We allow the computer to send events via the Serial Port, using the 115200 baudrate.

```
/**
 * Listens for incoming events on SerialPort.
 * The event is stored in the eventString variable and the flag eventReceived is raised when the event is available for processing.
 */
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    eventString += inChar;
    // if the incoming character is a newline, set a flag so the main loop can
    // do something about it:
    if (inChar == '\n') {
      eventReceived = true;
    }
  }
}
```

This function reads the serial input and fills any characters received in the eventString variable until the EOL character (\n). When the line is finished, the eventReceived flag is set to True.

In the loop() we can check if the variable eventReceived is true, and if yes, we can simply parse the input according the the event number received :

```
/**
 * Processes a received event and takes necessary actions
 */
void processEventReceived() {
  switch(getEventValue(eventString, 0).toInt()) {
    case EVENT_TRANSLATE: {
      String rtn = apiTranslate(getEventValue(eventString, 1), getEventValue(eventString, 2), getEventValue(eventString, 3));
      displayString(rtn);
      Serial.println("Translate event processed");
      break;
    }
    default:
      Serial.println("Unknown event received.");
    break;
  }
}
```

LCD Display

The LCD component is straight-forward to use with the built-in LiquidCrystal library. On the other hand, the wiring is complicated because it required soldering. Also, the pin-out documentation we first took wasn't for the ESP32 we had, and we found later that some pins we used was dedicated to the flashing process, thus making it unusable.

The interface we built to use the LCD is composed of two functions : displaySetup and displayString.

The displaySetup simply initialize the LCD component using pin numbers.

The displayString computes a given string in order to print it on the LCD. Initially the method would print 16 chars in the first row, then 16 on the bottom one, wait one second, then print the 32 subsequent characters in the

same manner, until all characters has been displayed. It worked, but it wasn't user-friendly, so

we decided to only use the first row, and simulate a scroll of two characters with a delay between each movement. This makes the text a lot more readable for the final user.

Microphone

We decided to buy some components in order to record our voice. We bought the MAX4466 with a manual gain modular. We first follow the tutorial on Adafruit in order to get our component working. After a long set up to combine our esp32 to connect to our microphone. We successfully made the connection between the two components and got a result : we could've stream our voice directly from the max4466 and listen to it with a headphone connected via bluetooth. The only point is that it has a lot of parazyte noises making the sounds not clear and difficult to listen to. Even with an API that treating the sounds the the esp32 output. We couldn't get any good tracks. We decided to use the microphone of the computer in order to show as a Proof Of Concept how our device work.

WiFi

In order to connect to our API, we had to provide some connection to the ESP.

The easiest way was to use the WiFi feature, with the built-in library. The final user just needs to enable an access point on his phone to allow the ESP to get result from our apis.

The WiFi connection handle is available for all other components of the code, especially the apiserver component.

API Server

As we decided to use an external server, we built a file handling all the necessary function to send and receive information from the server.

There is two needs : the first one is to receive a translation of a text given the text, the source language and the destination language.

The other need was to transcript an audio recording providing the audio file and the source language.

The implementation details is available in the subsequent section (API Server).

The transcript endpoint needs to upload the file to the server. We tried the UDHttp library and tweaked some parameters (chunk size) but unfortunately the upload hangs at some point. So we statically stored audio files to simulate our PoC on the server-side.

-- API Server

To facilitate operations such as audio transcript and translation, we created a Python server to handle the work.

We used Docker to run the server on any of our development machine and ensure that it can run without any dependency issue.

The main benefit of using Python is that we have so many packages made available to us by the OSS community.

The server is built using the Flask-Restful framework, enabling us to create a HTTP RESTful API.

There is two endpoints : /translate and /speech-to-text, both using POST.

The translate endpoint expects a payload containing the string to translate, the source and destination languages using two letter codes (fr, en, es...).

The processing is made by the googletrans library, which uses the Web API of Google Translate. We need to note that this library is not the official one, and shouldn't be used in production. For our project it's completely fine, as we won't hit any rate limit issue given.

The other endpoint, for the audio transcript, we used the Uberi/speech_recognition library, which supports multiple transcription backends, such as Microsoft's Azure Cloud or Google.

V/ User Manual

-- ESP32

In order to flash the board with our code, you can use the Arduino IDE with the ESP32 plugin : https://github.com/espressif/arduino-esp32

Edit the secrets.h file with the API URL (more instructions in the next section) and WiFi credentials.

You can use the Monitor Serial feature to see output of the code, using 115200 as the baud rate.

The Monitor Serial allows us to send events to the board using the following format :

1:Bonjour:fr:en

The 1 tells it's a translation request, the Bonjour is a random string, fr is the source language, and en is the destination language.

This event will make a translate request to the API and print the result on the LCD screen.

-- Server

The board requires an API to be running. We use docker to run the server and we provide a simple Makefile to quickstart the project.

Use make install to build the image and install dependencies into the container.

Use make run to spin up the server on the 5000 port.

During development, you'll probably want the server to be accessible on a public endpoint, to allow the ESP to reach your local server. We used the ngrok program to tunnel the connection. After installing ngrok and after starting the server, open a new terminal and type ngrok http 5000. Make note of the given HTTP address ending in ngrok.io, this is what needs to be used in the secrets.h file of the ESP32.

Conclusion

To conclude in this project, we really got lot difficult times to connect the ESP32 and our computer. With a lot of times we were doing it right, with the right pins, we still got some problems to televerse the code .ino to the ESP32. Same problem with the Screen in order to show our results. After finding the right way to connect the different pins, we could've manage to do the transcription eventually with the computer in order to realize our Proof of Concept.

Through this project, we learn a lot with the different labs to have some knowledge about how can we use an ESP32. We also improve our development in Ino, C and Python, something that we didn't practice that much since our first year.

Of course, few optimizations can be done in our project as it's only a proof of concept. But we're really glad to make a first introduction to this area that has a lot of potential. Few discovery has been made in order to create THE ultimate universal Language, but why creating a new language if we can directly adapt our language with the technologies we have today ?