

CENTRE DE FORMATION PROFESSIONNELLE TECHNIQUE

TRAVAIL DE DIPLÔME

RaspiHome

Documentation technique

Auteur :
Salvi Cyril

Encadrant :
M. BONVIN PASCAL

T.IS-E2B

2016 - 2017

1 Table des matières

| | | |
|----------|---|-----------|
| 1 | Table des matières | 1 |
| 2 | Résumé | 5 |
| 3 | Abstract | 6 |
| 4 | Remerciements | 7 |
| 5 | Cahier des charges | 8 |
| 5.1 | But | 8 |
| 5.2 | Description | 8 |
| 5.3 | Spécifications techniques | 8 |
| 5.4 | Restrictions | 8 |
| 5.5 | Environnement de développement | 8 |
| 5.6 | Livrables | 9 |
| 6 | Étude d'opportunité | 10 |
| 6.1 | Présentation de l'existant | 10 |
| 6.1.1 | Google avec "Google Home" | 10 |
| 6.1.2 | Amazon avec "Amazon echo" | 11 |
| 6.1.3 | Apple avec "Homekit" et "Siri" | 13 |
| 6.1.4 | Gatebox | 14 |
| 7 | RaspiHome | 15 |
| 7.1 | Introduction | 15 |
| 7.2 | Matériel utilisés | 16 |
| 7.2.1 | Raspberry Pi | 16 |
| 7.2.2 | Sense HAT | 17 |
| 7.2.3 | PiFace Digital 2 | 18 |
| 7.2.4 | Microphone MAX9814 | 19 |
| 7.2.5 | Microphone Sunfounder | 19 |
| 7.2.6 | Amplificateur classe D MAX98357A | 20 |
| 7.2.7 | Amplificateur classe D PAM8302A | 20 |
| 7.3 | Logiciels et technologies | 21 |
| 7.3.1 | Windows 10 Iot | 21 |
| 7.3.2 | Visual Studio 2015 et C# / CSharp | 21 |
| 7.3.3 | Universal Windows Platform | 21 |
| 7.3.4 | SpeechRecognition | 21 |
| 7.3.5 | SpeechSynthesizer | 22 |

| | | |
|----------|---|-----------|
| 7.3.6 | Sense HAT | 22 |
| 7.3.7 | PiFace Digital 2 | 22 |
| 8 | Analyse fonctionnelle | 23 |
| 8.1 | Serveur et client | 23 |
| 8.2 | Reconnaissance vocale | 24 |
| 8.3 | Synthèse vocale | 24 |
| 8.4 | Application tablette Windows | 25 |
| 9 | Analyse organique | 26 |
| 9.1 | Connexion entre les différents matériels | 26 |
| 9.2 | Serveur | 26 |
| 9.2.1 | Description | 26 |
| 9.2.2 | Écoute de la trame | 26 |
| 9.2.3 | Connexion et initialisation du client | 28 |
| 9.2.4 | Traitement de la commande reçue | 30 |
| 9.2.5 | Écriture dans la trame réseau du client | 33 |
| 9.2.6 | Déconnexion du client au serveur | 34 |
| 9.3 | Envoie et réception de commandes | 35 |
| 9.3.1 | Description | 35 |
| 9.4 | Client | 36 |
| 9.4.1 | Description | 36 |
| 9.4.2 | Initialisation du client | 36 |
| 9.4.3 | Lecture et écriture des données du client | 38 |
| 9.4.4 | Problèmes rencontrés et solutions | 40 |
| 9.5 | Carte électronique entrée et sortie audio | 41 |
| 9.5.1 | Description | 41 |
| 9.5.2 | Schéma électronique | 41 |
| 9.5.3 | Composants et méthodes utilisées | 42 |
| 9.5.4 | Problèmes rencontrés et solutions | 42 |
| 9.6 | Reconnaissance de commandes vocale | 43 |
| 9.6.1 | Description | 43 |
| 9.6.2 | Fonctionnement de la grammaire | 43 |
| 9.6.3 | Problèmes rencontrés et solutions | 45 |
| 9.7 | Synthèse vocale | 46 |
| 9.7.1 | Description | 46 |
| 9.7.2 | Fonctionnement de la synthèse | 46 |
| 9.7.3 | Initialisation de la synthèse vocale | 46 |
| 9.7.4 | Problèmes et résolution | 49 |
| 9.8 | Programme pour tablette | 50 |
| 9.8.1 | Description | 50 |

| | | |
|-----------|--|-----------|
| 9.8.2 | Interface graphique pour tablette | 50 |
| 9.8.3 | Fonctionnement de l'interface graphique | 51 |
| 9.8.4 | Envoi et lecture de données | 53 |
| 9.8.5 | Améliorations possible de l'application tablette | 55 |
| 10 | Application de gestion du Sense HAT | 56 |
| 10.1 | Description | 56 |
| 10.2 | Contrôle du Sense HAT | 56 |
| 10.3 | Gestion du Sense HAT | 57 |
| 10.4 | Envoi des valeurs formatées | 59 |
| 11 | Application de gestion du PiFace Digital 2 | 60 |
| 11.1 | Description | 60 |
| 11.2 | Contrôle du PiFace Digital 2 | 60 |
| 11.3 | Recherche et application de modification | 60 |
| 11.4 | Classe de gestion des lumières | 66 |
| 11.5 | Classe de gestion des stores | 69 |
| 11.6 | Améliorations possible de la gestion des stores | 75 |
| 12 | Problèmes rencontré et résolution | 76 |
| 12.1 | Composant reçu différent | 76 |
| 12.2 | Carte électronique pour ajouter un microphone | 76 |
| 12.3 | Échantillonnage du signal du convertisseur | 76 |
| 12.4 | Langue de synthèse par défaut de Windows Iot | 77 |
| 12.5 | Valeurs faussées du module Sense HAT | 77 |
| 12.6 | Communication avec le Sense HAT | 77 |
| 13 | Protocole de tests | 78 |
| 13.1 | Procédé de communication | 78 |
| 14 | Conclusion | 81 |
| 15 | Table des figures | 82 |
| 16 | Table des listings | 83 |
| 17 | Références | 84 |
| 17.1 | Références d'ordre général | 84 |
| 17.2 | Références STT et TTS | 84 |
| 17.3 | Références Sense HAT | 84 |
| 17.4 | Références PiFace Digital 2 | 84 |
| 17.5 | Références Tablette Windows | 85 |

2 Résumé

RaspiHome est un système de domotique, qui permet de simplifier des tâches quotidiennes. Ce projet utilise une application tablette Windows et un programme de reconnaissance vocale pour commander des objets de domotique. Le projet utilise un système qui synthétise la voix pour combler le manque de support visuel que la reconnaissance vocale n'offre pas.

Ce projet utilise plusieurs mini-ordinateurs de type Raspberry Pi, qui ont chacun leur propre rôle et qui communiquent tous par le biais du Wi-Fi. Les modules que l'on retrouve sur les Raspberry Pi sont soit des SenseHAT pour récupérer des informations météorologiques, soit des PiFace Digital 2 pour commander des objets électroniques. Ce projet utilise plusieurs bibliothèques telles qu'une bibliothèque qui permet de gérer la reconnaissance vocale et la synthèse vocale, ou encore une bibliothèque qui s'occupe de la communication entre les différents programmes à distance.

Chaque Rapsberry Pi se trouvent muni d'un système d'exploitation de type Windows 10 Iot, afin de pouvoir être utilisé avec le logiciel de développement Visual Studio 2015 et être programmé par ce dernier. La communication doit avoir un serveur qui va transmettre les bonnes informations aux bons Raspberry Pi. Les commandes sont transmises grâce à l'application de reconnaissance vocale et par l'application pour tablette.

Ce projet permet d'apprendre plusieurs types de technologies différentes, ainsi que de plusieurs façon d'utilisation d'une application universelle.

3 Abstract

RaspiHome is a home automation system that simplifies everyday tasks. This project uses a Windows tablet application and a voice recognition program to control home automation objects. The project uses a system that synthesizes the voice to fill the lack of visual support that speech recognition does not offer.

This project uses several mini-computers of the Raspberry Pi type, each of which has its own role and that all communicate via Wi-Fi. The modules found on the Raspberry Pi are either SenseHAT to retrieve meteorological information , Or PiFace Digital 2 to control electronic objects. This project uses several libraries such as a library that can manage speech recognition and speech synthesis, or a library that handles communication between different programs remotely.

Each Rapsberry Pi is equipped with a Windows 10 Iot operating system so that it can be used and programmed with Visual Studio 2015 development software. The communication must have a server that will transmit the right information to the good Raspberry Pi. The commands are transmitted through the speech recognition application and the tablet application.

This project allows to learn several different types of technologies, as well as several ways of using a universal application.

4 Remerciements

Je tiens spécialement à remercier toute ma classe pour l'attitude joyeuse qu'il y avait tout au long du travail de diplôme.

Je remercie mon encadrant, Monsieur Bonvain, qui m'a aidé pour l'installation des Raspberry Pi, ainsi que pour l'utilisation du microphone sur le Raspberry Pi.

Je remercie Monsieur Garcia, qui m'a aidé sur un problème de structure de données, et qui est resté avec moi jusqu'à trouver la solution. Je remercie aussi Monsieur Garcia de son soutien malgré le fait que je ne suis pas un de ses élèves à encadrer.

Je remercie Monsieur Marechal qui m'a fourni le matériel nécessaire, ainsi que du soutien qu'il a fait preuve tout au long de l'année pour ses élèves.

Je remercie mes amis qui m'ont soutenu, ainsi que ceux avec qui je pouvais dialoguer au sujet de mon projet, car ils étaient tout autant intéressés par le projet que moi.

Je remercie finalement ma famille qui m'a soutenu tout au long du projet ainsi que pour mes différentes formations. et qui, grâce à eux j'ai eu le courage de m'attaquer à plus dure que moi.

5 Cahier des charges

5.1 But

RaspiHome permet d'automatiser des tâches quotidiennes et simples, afin de permettre aux personnes qui l'utilise, de gérer leur espace de vie au quotidien.

5.2 Description

RaspiHome est un système de domotique commandé par la voix et par écran tactile. Cette application aide à simplifier les tâches quotidiennes de l'utilisateur à l'aide d'une simple parole et d'un simple touché pour piloter des objets tels que les lumières ou les stores, et cela à distance.

5.3 Spécifications techniques

L'application est écrite en C# 6.0 sous .Net 4.6.1. Le réseau Wi-Fi est sécurisé par clé WPA2, SSL et SSID privé. Les Raspberry Pi sont installés avec une image Windows 10 IoT Core.

- Ils sont programmés en C# .Net
- Ils sont équipés soit d'un Sense Hat, soit d'un Piface Digital 2.
- Un seul Raspberry Pi est équipé d'un module électronique PCB à concevoir avec microphone à gain réglable type MAX9814 et haut-parleur intégré.

La base de données est SQLite3, avec le connecteur C# System.Data.SQLite. La gestion de la base de données est contrôlée depuis SQLiteStudio et C#. La bibliothèque TextToSpeech est à trouver en libre de droits. La bibliothèque de reconnaissance vocale (Speech SDK ou bibliothèque de Microsoft) est à trouver en libre de droits. Les commandes vocales possibles sont limitées (ouvrir, fermer, allumer, éteindre, etc...) et associées par apprentissage dans l'application principale, il n'y a pas d'intelligence artificielle d'analyse vocale.

5.4 Restrictions

L'application n'intègre pas de reconnaissance de mouvements ou de reconnaissance d'images. Elle n'intègre pas non plus d'intelligence artificielle.

5.5 Environnement de développement

Cet environnement décrit les outils de base utilisés pour la création d'une application C# avec le matériel nécessaire au projet, ainsi que pour la rédaction d'une documentation et de la création d'un poster en français et en anglais :

- Ordinateur : type "Personal Computer"
- Mini-ordinateur : type Raspberry PI
- Système d'exploitation : Windows 10, Windows 10 IoT Core
- Langage utilisé : C#
- Matériel nécessaire : Raspberry PI 3, Microphone, Capteur de température, Capteur d'humidité, Capteur de pression atmosphérique, Capteur de lumière, Capteur magnétique, Interrupteur magnétique
- Traitement de texte : Google docs, Microsoft Word, Sharelatex
- Traitement d'image : Paint.NET 4.0.10
- Traitement d'image vectorielle : InkScape 0.91

5.6 Livrables

5 mai 2017

- Poster explicatif en français et en anglais
- Documentation intermédiaire

19 mai 2017

- Résumé intermédiaire en français et en anglais

12 juin 2017

- Code source
- Documentation technique
- Journal de bord
- Fiches techniques du matériel
- Listes du matériel
- Schéma PCB de la plaque audio
- Manuel utilisateur

6 Étude d'opportunité

6.1 Présentation de l'existant

6.1.1 Google avec "Google Home"

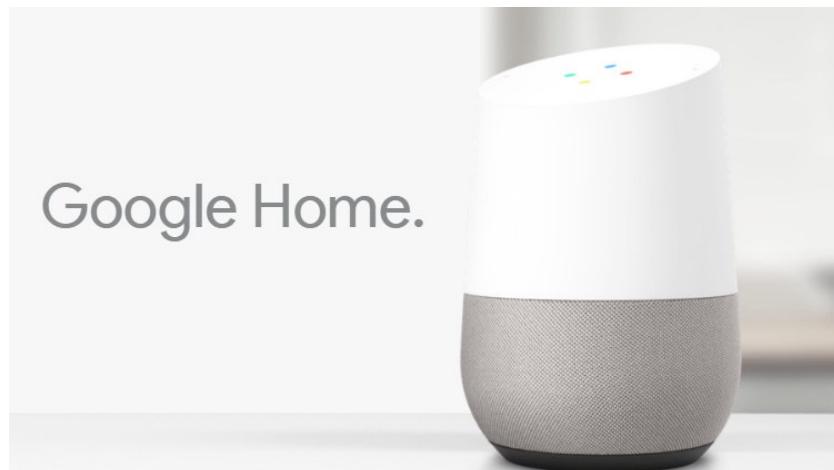


FIGURE 1 – Google Home - Image de présentation

Google Home est un assistant personnel intelligent fabriqué par l'entreprise américaine Google. Il est muni d'un haut-parleur ainsi que de deux microphones qui permettent à l'appareil de réagir aux commandes vocales des personnes se trouvant à proximité.

Google Home utilise la reconnaissance vocale pour activer des modules électroniques. Son prix de départ est de 129 dollars américains, et est compatible avec du matériel Wi-Fi, tel que des lampes Wi-Fi, ou des modules sur lesquels on y branche du simple matériel électronique qui n'est pas prévu pour la domotique à la base.

Points forts :

- Reconnaissance vocale
- Synthèse vocale
- Mise à jour automatique
- Vocabulaire dynamique et constamment à jour
- Contrôle de plusieurs objets à distance

Points faibles :

- Ordres que vocaux, aucun autre support
- Objets seulement compatible et pas adaptable
- Composants disponibles mais chère

6.1.2 Amazon avec "Amazon echo"



FIGURE 2 – Amazon Echo - Image de présentation

Amazon Echo¹ est l'oeuvre de la société Amazon, initialement spécialisé dans la vente en ligne, cette entreprise a développé et mis en place sur le marché leur nouveau produit "Amazon Echo". Cet appareil utilise les technologies de Google Home, ainsi que celles d'Amazon Alexa qui est un assistant personnel intelligent. Cette dernière technologie, développée par Amazon à été rendue populaire grâce à Amazon Echo. Ce dernier, est un haut-parleur intelligent, dit mains libres, qui est contrôlé au son de la voix et permet ainsi de commander l'appareil et des objets de domotique pour connaître les conditions météorologiques, les nouvelles dans le monde, et plus encore.

Amazon Alexa est un autre projet d'Amazon, développé par Lab126², qui est une partie d'Amazon qui recherche et développe des appareils électroniques de haut niveau pour le grand public.

Le réel point fort de cet appareil est Alexa qui est le cerveau, et surtout la partie la plus importante du produit final. Cette dernière est connectée à Internet, et devient de plus en plus intelligente. Amazon Echo s'adapte aussi aux modes de parole³, vocabulaire et préférences personnelles. En plus de cela, l'appareil est constamment mis à jour, sans que l'utilisateur n'est quoi que ce soit à toucher

-
1. https://en.wikipedia.org/wiki/Amazon_Echo
 2. <https://www.lab126.com/>
 3. <https://www.amazon.com/Amazon-SK705DI-Echo/dp/B00X4WHP5E>

étant donné que l'appareil est toujours connecté à Internet. En revanche Amazon Echo est restreint sur les appareils qu'il peut commander, il est vrai qu'il contrôle parfaitement la lumière ainsi que de contrôler un ventilateur ou de réchauffer la pièce, c'est là où s'arrête Amazon Echo.



FIGURE 3 – Amazon Echo - Utilitaire d'Amazon Echo

Points forts :

- Reconnaissance vocale intelligente
- Synthèse vocale intelligente
- Mise à jour automatique
- Constantement à jour
- Vocabulaire dynamique et constamment à jour
- Contrôle de plusieurs objets à distance

Points faibles :

- Ordres que vocaux, aucun autre support
- Objets seulement compatible et pas adaptable
- Composants disponibles mais chère

6.1.3 Apple avec "Homekit" et "Siri"



FIGURE 4 – Homekit - Image de présentation

HomeKit est un produit dérivé de la marque Apple, il s'agit d'une application compatible uniquement avec l'iPhone, l'iPad et l'iWatch, et qui communique avec des accessoires de domotique, à partir du moment où ils sont compatibles avec l'application (plus de 50 marques dans le monde proposent des accessoires compatibles)⁴. Pour que HomeKit fonctionne, il faut avoir en sa possession un module Apple TV pour pouvoir utiliser cette application. Une fois en possession de cet objet, il suffit simplement d'ouvrir l'application allouée à cet effet qui est installée d'office sur les appareils précédemment cités.

Le réel problème de HomeKit est qu'il est seulement compatible avec des produits Apple, ce qui restreint beaucoup de choses tels que le nombre de personnes à bénéficier d'appareil Apple à ce jour, ainsi que du nombre parmi ces personnes à disposer d'un module Apple TV, et enfin ces personnes doivent avoir les moyens nécessaire afin pouvoir installer de la domotique dans leur foyer

Points forts :

- Mise à jour automatique
- Fonctionne avec seulement un Apple TV
- Contrôle de plusieurs objets à distance

Points faibles :

- Application mobile seulement
- Besoin d'appareils Apple pour fonctionner
- Objets seulement compatible et pas adaptable
- Composants disponible mais chère

4. <https://www.apple.com/fr/ios/home/>

6.1.4 Gatebox



FIGURE 5 – Gatebox - Image de présentation

Gatebox⁵ est un appareil qui commande des objets connectés. Tout droit venu du Japon, cet appareil donne vie à la technologie actuelle en mélangeant hologramme, reconnaissance vocale, synthèse vocale et domotique. Il permet de commander des choses simples telles que la lumière.

Ce projet a été développé pour les personnes qui sont tout le temps seul chez eux, et ainsi empêcher ces personnes de commettre l'irréparable en se suicider avec la pression constante qu'ils subissent au travail et du fait qu'ils n'ont personne sur qui se reposer après avoir fini leur journée. Le Japon est un pays avec au grand taux de suicide par année dû aux raisons citées précédemment.

Points forts :

- Reconnaissance vocale
- Synthèse vocale
- Hologramme réactif et dynamique
- Chat à distance avec l'hologramme
- Contrôle de plusieurs objets à distance

Points faibles :

- Objets seulement compatible et pas adaptable
- Composants disponibles mais chère

5. <http://gatebox.ai/story/>

7 RaspiHome

7.1 Introduction

RaspiHome est une application C# dans le domaine de la domotique. Ce dernier, est un domaine qui est apparu au milieu des années 1980. Il s'agit d'une méthode qui rassemble l'électronique et l'informatique pour permettre d'automatiser un foyer. Il se charge de s'occuper des tâches quotidiennes telles qu'activer des appareils électroniques que le foyer contient. En passant de l'appartement, jusqu'à une grande surface de plusieurs hectares, la domotique s'implémente dans toutes circonstances et se trouve dans un marché en expansion depuis quelques années, car les demandes et les tendances sont, maintenant, étroitement liées.

Ce projet permet donc à l'utilisateur d'automatiser son foyer, et de quémander une tâche grâce à une tablette portable de type Windows, ou grâce à une commande vocale. Cette dernière est une méthode peu courante dans le domaine de la domotique actuelle, qui utilise majoritairement une tablette portable, un téléphone portable ou encore un écran tactile mural pour gérer le matériel de domotique. Malgré cela, elle est une méthode qui permet de se démarquer de ce domaine en tant que telle. En revanche sur ce marché, on commence à y trouver des entreprises telles que Google, Amazon ou encore Apple, qui se décident à rentrer dans ce marché et de sortir des appareils connectés et commandables par la voix.

Points forts :

- Application tablette
- Reconnaissance vocale
- Synthèse vocale
- Utilisation simple
- Composants adaptable
- Contrôle de plusieurs objets à distance

Points faibles :

- Commandes fixe et non adaptable (pas de mémorisation des précédentes commandes)
- Dispose de peu de commandes

7.2 Matériel utilisés

7.2.1 Raspberry Pi



FIGURE 6 – Raspberry Pi 3 Modèle B - Plaque électronique

Un Raspberry Pi est un mini-ordinateur. Sur le marché on trouve divers modèles de Raspberry Pi; des modèles A, des B et des Zéro. La version utilisée pour ce projet est le Raspberry Pi 3 Modèle B, sur lequel on peut y brancher une entrée vidéo, un clavier et une souris pour faire office d'un vrai ordinateur. On y trouve aussi une entrée RJ45 pour internet ainsi que d'un connecteur jack audio qui fonctionne seulement comme une sortie analogique. De plus sur cette plaque on y trouve un récepteur Wi-Fi et un récepteur Bluetooth directement intégrés à la plaque PCB. Le Wi-Fi permet de remplacer le câble RJ45.

RaspiHome est principalement basé sur le fait d'utiliser des Raspberry Pi pour gérer le matériel, dit connecté, que peut contenir un foyer. Ces appareils communiquent par le biais du Wi-Fi, ce qui enlève la restriction d'être câblé à un réseau fixe pour que chacun puissent communiquer entre eux.

Le projet utilise un Raspberry Pi pour transmettre les commandes, et c'est lui qui à son tour, traitera les informations qu'il aura reçues venant des autres Raspberry Pi. Chaque Raspberry Pi auront leurs propres rôles et leurs propres fonctionnalités : Exemple concret, ce projet fonctionne avec la détection de voix, donc un Raspberry Pi est alloué à cette tâche, et il contient un microphone pour enregistrer le son émis pas la voix. On peut y ajouter une sortie audio en plus du microphone qui lui est une entrée, pour permettre d'avoir un retour audible grâce à un système de synthèse de la voix.

7.2.2 Sense HAT

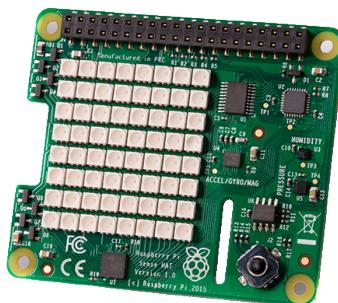


FIGURE 7 – Sense HAT - Plaque électronique

Le Sense HAT dispose de plusieurs capteurs de mesure tel que ; un capteur de température, un capteur de pression atmosphérique, un capteur d'humidité ainsi que d'un accéléromètre, d'un gyroscope et d'un magnétomètre.

Cette circuit imprimé se branche directement sur les PIN d'entrées et de sorties du Raspberry PI 3 Modèle B. Par la suite, il résulte du Sense HAT, des valeurs dérivées des capteurs, qui permettent connaître l'état de la pièce ou de l'extérieur tel que la température, l'humidité, la pression atmosphérique et plus encore, selon l'emplacement du Raspberry Pi.

La bibliothèque qui permet de faire fonctionner le Sense HAT est simple à implémenter et le tutoriel qui est fourni avec est simple d'utilisation aussi. On retrouve beaucoup de projets utilisant ce composant sur interne, et de ce point, on y trouve plusieurs manières de l'utiliser. Dans ce projet le Sense HAT n'a d'utilité que pour connaître la température, l'humidité ainsi que de la pression atmosphérique d'une pièce.

7.2.3 PiFace Digital 2

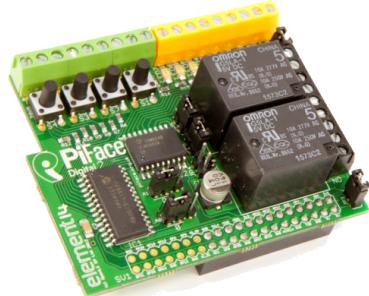


FIGURE 8 – PiFace Digital 2 - Plaque électronique

Le PiFace Digital 2 est une plaque qui se branche directement sur un Raspberry Pi 3 Modèle B. Cette carte est munie de 2 relais électriques qui permettent de commander et de commuter du matériel connecté à une source 230 Volts tels que ; des ampoules, le chauffage ou encore d'autres appareils qui fonctionnent avec en étant branché sur une prise secteur. Cet outil contient aussi 8 entrées "Digitales" ainsi que de 8 sorties "Open-Collector" et de 4 boutons interrupteur.

Les 2 relais sont chacun alimenté par une bobine interne de 5 Volts, qui permet de faire commuter l'interrupteur électrique contenu dans le relais. Sur ce dernier, il peut y être brancher des éléments à forte tension telle que des objets électriques sur une prise secteur ou des ampoules comme cité au-dessus. Les relais sont connectés aux sorties 0 et 1 du PiFace Digital 2 et sont pilotables par le Raspberry Pi lui-même. Les 8 sorties du PiFace Digital 2 ont une tension maximum qui va entre 3.3 à 4 Volts (valeur mesurée à l'oscilloscope lors de la mesure d'un signal) en sortie et les 8 entrées ont une tension de 3.3 Volts maximum admise.

Il n'existe pas de bibliothèque mise à disposition pour piloter le PiFace Digital 2 tel que la bibliothèque qui est utilisé pour piloter le Sense HAT. Pour que le PiFace Digital 2 fonctionne, il faut récupérer un programme que les développeurs du produit ont mis à disposition sur leur site. Le code fourni permet de piloter la composante principale du Piface soit le MCP23S17. Ce composant fait parties d'une famille d'interface série, le MCP23O17 qui est une interface I²C et celui utilisé par le PiFace Digital 2 qui est une interface SPI.

7.2.4 Microphone MAX9814

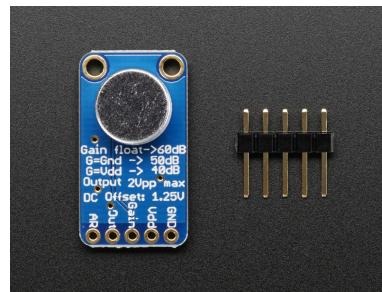


FIGURE 9 – MAX9814 - Plaque électronique

Le MAX9814 est un microphone de petite taille. Cette plaque est utilisée pour récupérer les sons émis par la voix. Il détecte cette dernière et retransmet dans un signal sinusoïdal variable, le signal qu'il génère en sortie. L'inconvénient est que le Raspberry Pi ne contient pas d'entrée analogique, donc il faut que le projet puissent convertir ce signal sinusoïdal en un signal digital, afin de pouvoir le traiter de façon à obtenir plusieurs valeurs binaires, qui peuvent par la suite être traitées par le programme de n'importe quelle façon.

7.2.5 Microphone Sunfounder

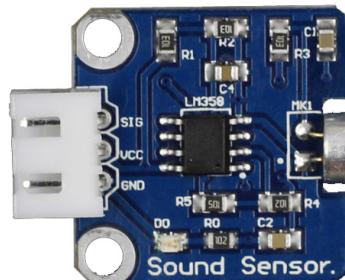


FIGURE 10 – Sound sensor - Plaque électronique

Le microphone de Sunfounder est un peu différent du microphone précédent, car il n'a qu'une seule PIN de sortie, vu que le gain est déjà intégré à la plaque. Par la suite l'utilisation reste la même, car le signal de sortie reste avant tout un signal sinusoïdal qu'il faut échantillonner.

7.2.6 Amplificateur classe D MAX98357A



FIGURE 11 – MAX98357A - Plaque électronique

Le MAX98357A est un amplificateur audio de classe D, soit une classe de très bonne qualité. Cette plaque est monté sur un circuit PCB qui peut par la suite être connecté au Raspberry PI. Cet amplificateur à en sortie une PIN "+" et une PIN "-" où un haut-parleur peut être connecté. Ce composant est utilisé pour donner une réponse suite à une commande demandée précédemment. En revanche, le Raspberry Pi comporte une sortie jack où un haut-parleur peut directement être câblé. Le point faible de cette façon de câblé est que tout est déjà implémenté, ainsi que le gain déjà intégré, du coup le Raspberry PI ne dispose pas d'un gain réglable, que l'amplificateur, lui dispose.

7.2.7 Amplificateur classe D PAM8302A

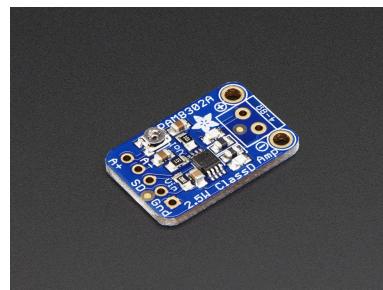


FIGURE 12 – PAM8302A - Plaque électronique

Le PAM8302A est un amplificateur classe D qui reçoit 2 entrée audio (gauche et droite), pour en ressortir les mêmes tout en ayant augmenté son signal en sortie. Celui-ci est plus simple d'utilisation comparé au précédent qui comporte plusieurs entrées, celui-ci n'en a que 2.

7.3 Logiciels et technologies

7.3.1 Windows 10 IoT

Windows 10 IoT (Internet of things), où Windows intégré, est un opérateur système de Microsoft qui a été designer pour des systèmes intégrés tels que des Raspberry Pi. Une fois installé sur le Raspberry Pi, ce dernier peut-être programmé à l'aide de Visual Studio 2015, soit en C#, en C++ ou en Javascript.

7.3.2 Visual Studio 2015 et C# / CSharp

Toutes les applications contenues dans le projet, sont programmées dans le langage C# à l'aide de l'outil de développement qui est Visual Studio 2015. Le projet use de toutes les bibliothèques de base qui sont disponibles sur Visual Studio 2015. Aucunes bibliothèques supplémentaire n'a dû être téléchargée pour faciliter le développement de partie complexe du projet, étant donné qu'il est possible de tout faire avec les bibliothèques de base, sauf pour le Sense HAT qui dispose d'une librairie à lui.

7.3.3 Universal Windows Platform

Pour créer un programme qui soit compatible avec un Raspberry Pi fonctionnant sous Windows 10 IoT, il faut cocher une case lors de l'installation de Visual Studio, qui stipule que Visual Studio est apte à pouvoir être utilisé sur des systèmes intégrés tel que ; des téléphones, des tablettes ou des Raspberry Pi. Le nom de l'application qui permet de créer ces programmes par la suite, est UWP qui est l'acronyme de Universal Windows Platform.

7.3.4 SpeechRecognition

Les Raspberry Pi installés avec Windows IoT ne peuvent accepter toutes les bibliothèques de reconnaissance vocale. Le réel problème est que la bibliothèque fonctionne avec une grammaire anglaise ce qui fausse la compréhension des mots issue de la langue française, par contre il peut y être ajouté une grammaire personnalisée qui contient autant de mots que besoin. Malgré tout, la reconnaissance vocale est possible grâce à cette bibliothèque, en revanche l'initialisation est différente de la version sur ordinateur car cette bibliothèque a été développé express pour les application UWP.

7.3.5 SpeechSynthesizer

Les Raspberry installé avec Windows Iot ne peuvent accepter toutes les bibliothèques de synthèse vocale. Cette bibliothèque est complexe à utiliser, étant donné que sur le Raspberry Pi, la langue installée de base n'est pas le français mais l'anglais américain, malgré la sélection de la langue lors de l'installation. Donc le résultat d'une commande écrite en français n'est pas le résultat souhaité, vu que le synthétiseur lit en anglais, alors que la syntaxe n'est pas la même. Ce qui résulte est du français avec une élocution purement anglaise, qui irrite passablement les oreilles et qui est pratiquement impossible à déchiffrer.

7.3.6 Sense HAT

Cette bibliothèque permet de contrôler le Sense HAT, et ainsi de manipuler ses divers capteurs comme la température, l'humidité ou la pression atmosphérique, mais aussi d'utiliser les valeurs de l'accéléromètre et du gyroscope ou encore de contrôler la matrice à LED ou le joystick que le module contient. Pour ce projet seule la température, l'humidité et la pression atmosphérique sont utilisées.

7.3.7 PiFace Digital 2

La programme qui permet de faire fonctionner le PiFace Digital 2 a été récupérer d'un projet⁶ où est utilisé le composant sur un Raspberry Pi 3 qui fonctionne sous Windows Iot. Cette bibliothèque définit simplement quels sont les ports d'entrées et les ports de sorties tout en initialisant chaque composants. L'utilisation est simple car il suffit d'écrire au moment de l'utilisation d'une PIN, son port sur lequel il est connecté, ainsi que son état s'il est à l'état haut "1" ou à l'état bas "0", pour actionner son système.

6. www.element14.com/community/community/raspberry-pi/blog/2015/08/03/piface-digital-ii-on-a-pi2-windows-10-iot-and-a-cool-demo

8 Analyse fonctionnelle

Le projet RaspiHome fonctionne avec plusieurs modules Raspberry Pi 3. Chacun ont leurs propres rôles et fonctionnalités, et ils communiquent tous par le biais d'un serveur TCP. La communication se fait à l'aide d'un Wi-Fi sécurisé et les différents modules qui communiquent entre eux grâce à ce serveur qui transmet les informations aux bonnes personnes. Les ordres donnés sont transmis soit par le biais d'une application sur tablette Windows ou alors commandés par la voix pour ensuite être synthétisé pour compenser le manque d'un support visuel.

8.1 Serveur et client

Un serveur est indispensable pour faire la communication entre plusieurs matériels connectés au réseau, et ainsi filtrer les clients. Il existe plusieurs moyens de communication, comme par exemple la lecture et l'écriture d'une base de données, ou encore la lecture et l'écriture directement dans la trame de communication du module, soit la trame réseau du Raspberry Pi. C'est cette dernière méthode qui a été retenue pour faire fonctionner le projet. Malgré qu'il s'agisse d'une partie complexe, le développement est facile à comprendre. Le point le plus complexe est de déterminer qui fait quoi et cela à quel moment et où, car chaque commande reçoit forcément une réponse, qu'elle soit visuelle ou audible, l'entrée doit rester constamment connectée à la sortie pour permettre un fonctionnement correct de n'importe quel projet utilisant la communication à travers un serveur.

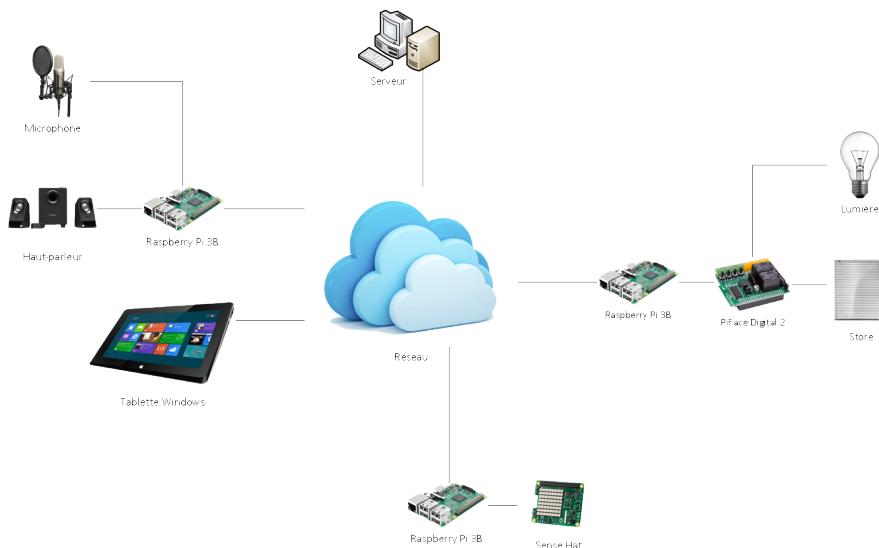


FIGURE 13 – Structure réseau du projet

8.2 Reconnaissance vocale

La reconnaissance vocale est une partie très importante et complexe dans ce projet, car elle utilise la technologie "SpeechToText" ou plus couramment "STT", et va retranscrire ce qu'elle entend à l'écrit, telle une dictée. Ce point à l'air simple, mais se révèle très complexe étant donnée qu'il faut prendre en compte la grammaire de la langue, ainsi que divers paramètres tels que ; les connexions entre chaque mot, l'orthographe complexe de la langue française, et surtout avoir une bonne élocution au moment de la commande et avant tout, ne pas mâcher ses mots, pour ainsi assurer un bon fonctionnement du système. Tous ces points peuvent fausser les commandes pré-enregistrées telles que « allumer » où lors de la retranscription, le système peut comprendre « alu mes ».

Pour que cette technologie fonctionne, il faut avant tout une entrée audio, peu importe le type. Que ce soit venant d'un microphone ou d'un fichier audio, le fonctionnement interne reste le même, seule la grammaire de la langue peut modifier la retranscription.

8.3 Synthèse vocale

Ce programme fonctionne avec la technologie "TextToSpeech" ou plus couramment "TTS". Cette technologie lit un message et le retranscrit dans la langue adéquate, soit le français. Le TTS utilise ensuite des haut-parleurs pour avoir un résultat audible. Dans ce projet, le synthétiseur fonctionne seulement pour les réponses directes, ou les commandes qui exigent une réponse à cause du manque d'un support visuel.

8.4 Application tablette Windows

L'application pour tablette Windows est un moyen de gérer la domotique contenue dans l'environnement. Cette application remplace l'élocution par le toucher, et n'a ainsi pas besoin d'utiliser les technologies de reconnaissance vocale et de synthèse vocale pour fonctionner.

L'application est développée pour les tablettes Windows à cause de la taille d'affichage qui est plutôt spacieuse, ce qui donne avant tout un meilleur rendu visuel et fatigue moins la vue.

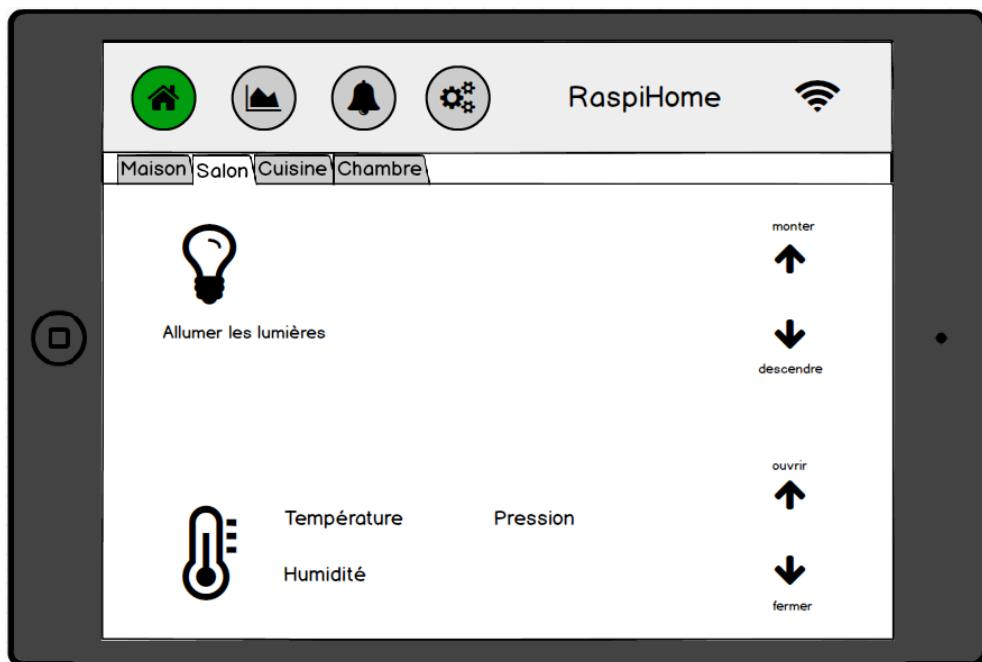


FIGURE 14 – Interface graphique schématisé de l'application pour tablette

9 Analyse organique

9.1 Connexion entre les différents matériels

Dans ce projet, il faut que les Raspberry Pi soient tous connecté au même réseau câblé ou Wi-Fi. Il faut aussi que le Wi-Fi soit toujours allumé pour éviter la perte de connexion entre les différents modules à cause du serveur qui contient tous les clients. RaspiHome ne se connecte pas sur un réseau à part, il se connecte sur le réseau principal du foyer. C'est par la suite que l'utilisateur choisit de le mettre sur un réseau externe.

9.2 Serveur

9.2.1 Description

La communication se fait grâce à un moyen de transmission par le biais du réseau (Wi-Fi ou câblé). Il faut donc créer un "serveur" TCP qui va s'occuper de lire les messages entrants et de les retransmettre aux bons clients. Pour cela, le serveur se doit d'être constamment actif et constamment à l'écoute. Une fois l'ordre reçut, le serveur recherche parmi tous ses clients lequel ou lesquels sont appelés. Il va rechercher en premier lieu la localisation de tous les Raspberry Pi, et ensuite il va rechercher chaque composant de chaque Raspberry Pi. S'il ne trouve pas le matériel désiré, le serveur ne va pas chercher plus loin et renvoie au demandeur une erreur telle que "le module souhaité n'existe pas dans ce lieu".

9.2.2 Écoute de la trame

Le premier état d'écoute du serveur, est le moment où ce dernier lit un message inconnu (client qui souhaite se connecter au serveur). Le client écrit sur sa trame un message avec un certain format, qui est le suivant "IPRasp={0};Location={1};Component={2}", et le serveur va ainsi procéder à une décomposition pour prendre les informations qui lui sont utiles. Le décodage de données passant d'un tableau de byte à une chaîne de caractères ce fait grâce à un composant de base de Visual Studio, qui est l'*Encoding.UTF8.GetString* que l'on peut voir dans les parties de programmes 1 et 2. Le code ci-dessous permet de lire le contenu de la trame réseau du client sélectionné.

```
1 // Creation of a new client at the connection
2 TcpClient newClient = this.Listener.AcceptTcpClient();
3
4 // Get the stream of the new client
5 NetworkStream netStream = newClient.GetStream();
6
```

```

7 // Set the buffer of the stream
8 newClient.SendBufferSize = _bufferSize;
9 newClient.ReceiveBufferSize = _bufferSize;
10
11 byte[] messageBuffer = new byte[_bufferSize];
12 int bytesRead = netStream.Read(messageBuffer, 0,
13     messageBuffer.Length);
14
15 if (bytesRead > 0)
16 {
17     // String result of the client stream
18     string messageRead = Encoding.UTF8.GetString
19         (messageBuffer, 0, bytesRead);
20 }

```

Listing 1 – Serveur - Premier état d'écoute de la trame

Le second état d'écoute du serveur, est un mode d'écoute constant à partir du moment où au moins un client est connecté au serveur. Le fonctionnement reste le même que la première méthode, à la différence que le client est déjà connecté et que le format du message qui est lu n'est plus spécifique, étant donné que la connexion a déjà été établie et le client déjà créé. Donc le message qui est lu est soit la transmission de commandes, soit le retour d'informations de mesure.

```

1 // Check the stream of every connected client
2 foreach (TcpClient client in this.Clients)
3 {
4     // Get the message if there is one
5     int messageLength = client.Available;
6     if (messageLength > 0)
7     {
8         byte[] messageBuffer = new byte[messageLength];
9         client.GetStream().Read(messageBuffer, 0,
10             messageBuffer.Length);
11
12         // New message from the client
13         string messageRead = String.Format
14             (Encoding.UTF8.GetString(messageBuffer));
15     }
16 }

```

Listing 2 – Serveur - Écoute constante de la trame

9.2.3 Connexion et initialisation du client

Lors de la connexion, le client qui veut se connecter envoi au serveur les informations nécessaires à l'initialisation d'un nouveau client dans le serveur. Ce dernier reçoit une chaîne de caractères spéciale, comme cité plus haut, qui permet de connaître le client, ainsi que ce qu'il fait. Cette chaîne de caractères regroupe les informations sur l'adresse IPv4 du client, sa localisation et ses composants. De ce point il est facile de créer un client en fonction des informations reçues.

```

1 /// <summary>
2 /// Initialize a raspberry pi with the information read
3 /// </summary>
4 /// <param name="rpiInformation"> Format of the string
5 // "IPRasp={0};Location={1};Component={2}" to read</param>
6 /// <returns> return the new client with all information of him
7 // </returns>
8 private RaspberryClient InitializeNewRaspberryClient(string
9 rpiInformation)
10 {
11     // Create an array of the actual string to get
12     // information
13     string[] rpiInformations = rpiInformation.Split(';');
14     string rpiIPv4 = "";
15     string rpiLocation = "";
16     string rpiComponent = "";
17
18     foreach (var information in rpiInformations)
19     {
20         switch (information.Split('=').First())
21         {
22             // Get the first value of the array
23             // IP of the actual Raspberry
24             case "IPRasp":
25                 rpiIPv4 = information.Split('=').Last();
26                 break;
27             // Get the second value of the array
28             // Location of the actual Raspberry (where to find
29             // the Raspberry)
30             case "Location":
31                 rpiLocation = information.Split('=').Last();
32                 break;
33             // Get the third value of the array
34             // Component of the actual Raspberry (what's the
35             // component used by the Raspberry)
36             case "Component":
37                 rpiComponent = information.Split('=').Last();
38                 break;
39         }
40     }
41 }
```

```
34     }
35
36     // Create a client with IPv4, Default port, localisation
37     // and new list of component
38     RaspberryClient client = new RaspberryClient(rpiIPv4,
39     DEFAULT_PORT, rpiLocation, new List<Component>());
40
41     try
42     {
43         // Get all components of the Raspberry
44         string[] components =
45         rpiComponent.Split('=').Last().Split(',');
46
47         // Get all class types in the project
48         Type[] types =
49         Assembly.GetExecutingAssembly().GetTypes();
50
51         // Check all components
52         foreach (var component in components)
53         {
54             // Check all class types
55             foreach (var type in types)
56             {
57                 if (type.Name == component)
58                 {
59                     // Instance a new component with the type
60                     result
61
62                     client.Components.Add((Component)Activator.CreateInstance(type));
63                     break;
64                 }
65             }
66         }
67     }
68     catch (Exception){} // Message unreadable by the program
69
70     // Add final client to the main list
71     this.RpiClients.Add(client);
72
73     // The actual client with all informations
74     return client;
75 }
```

Listing 3 – Serveur - Crédit de clients avec leurs propres informations

9.2.4 Traitement de la commande reçue

Une fois la commande, qui a pour thème d'être un ordre, est lue par le serveur. Ce dernier va faire une recherche intelligente, pour informer les bons Raspberry Pi de se mettre à jour. Le fonctionnement est simple. Un "client" a comme information son "nom", sa "localisation", son "adresse IPv4", ainsi que tous ses "composants". De ce point, il est facile de faire des recherches poussées. En premier lieu, le serveur va rechercher dans les informations des clients, tous les clients qui sont localisés dans le lieu mentionné dans la phrase, prenons exemple avec la phrase suivante : "allume la lumière du salon". Donc le mot à retenir pour localiser les éventuels Raspberry Pi est "salon". En restant avec ce même exemple, le programme va par la suite regarder si le client/les clients qui sont au salon, si le composant "lumière" est existant ou pas grâce à l'initialisation qui a permis de créer les clients et de connaître tous leurs composants. Et enfin, une fois le processus terminé, le serveur n'a plus qu'à informer les clients concernés pour qu'ils se mettent à jour, car il aura récupéré toutes les informations nécessaires pour savoir avec qui communiquer.

```

1 this.Sentence = RemoveDiacritics(paramSentence);
2
3 List<TcpClient> result = new List<TcpClient>();
4
5 string action = "";
6 string location = "";
7 string componentWithoutAction = "";
8
9 // Get the component word in the sentence
10 string component = this.GetComponentFromSentence(this.Sentence);
11
12 Type componentType = null;
13 string actionValue = "";
14
15 // Different usage of the order between an component with
   action and without one
16 // Writes values and send to the client the information or read
   values
17 if (component != "")
18 {
19     // Get the action word with in the sentence
20     action = this.GetActionFromSentence(this.Sentence);
21     // Get the property name with the action word
22     actionValue = ReadValueOfSelectedComponent(action);
23     // Get the class type founded with the component name
24     componentType = this.GetComponentType(component);
25 }
26 else

```

```
27 {
28     // Get the sensor component word in the sentence
29     componentWithoutAction =
30         GetIndependantComponentFromSentence(this.Sentence);
31     // Get the class type founded with the component name
32     componentType =
33         this.GetComponentType(componentWithoutAction);
34 }
35 // Check every clients
36 foreach (var rpiClient in paramRaspberryClients)
37 {
38     // Get the location word in the sentence
39     location =
40         this.GetSentenceLocationOrRaspberryLocation(this.Sentence,
41             rpiClient);
42
43     // Check every clients at this location
44     if (rpiClient.Location.ToLower() == location.ToLower())
45     {
46         // Check every clients at this location with this
47         // component
48         foreach (var itemType in rpiClient.Components)
49         {
50             // Check if the type is the same
51             if (itemType.GetType() == componentType)
52             {
53                 if (action != "")
54                 {
55                     // Write the new value string informations
56                     this.writeValue(itemType, action,
57                         itemType.GetType().GetProperty(actionValue));
58                     foreach (var name in paramClientsName.Keys)
59                     {
60                         if
61                             (paramClientsName[name].ContainsKey(rpiClient))
62                         {
63                             // Add the TCPClient inside the
64                             // dictionnay
65                             result.Add(paramClientsName[name][rpiClient]);
66                         }
67                     }
68                 }
69             }
70         }
71     }
72     else
73     {
74         foreach (var name in paramClientsName.Keys)
75         {
76             if
77                 (paramClientsName[name].ContainsKey(rpiClient))
78             {
79                 // Add the TCPClient inside the
80                 // dictionnay
81                 result.Add(paramClientsName[name][rpiClient]);
82             }
83         }
84     }
85 }
```

```
66             if
67             (paramClientsName[name].ContainsKey(rpiClient))
68             {
69                 // Add the TCPClient inside the
70                 // dictionnay
71                 result.Add(paramClientsName[name][rpiClient]);
72             }
73         }
74     }
75 }
76 }
77
78 return result;
```

Listing 4 – Serveur - Recherche des bons clients

9.2.5 Écriture dans la trame réseau du client

L'écriture dans la trame est facile. Une fois que le client est connu, il suffit simplement d'écrire dans sa trame de lecture/écriture des données dans un langage binaire, qui seront par la suite retranscrits par le client lui-même. Le message est écrit en format tableau de byte et 1 byte équivaut à 8 bits soit une valeur de 0 à 255 en chiffre numérique par valeurs dans le tableau.

```
1 /// <summary>
2 /// Convert the message in bytes and write in the stream of the
3 /// client
4 /// <param name="message"> message to send </param>
5 public void SendMessages(TcpClient clientToSend, string message)
6 {
7     // Encode the message
8     if (this.MessageQueue.Count != 0)
9     {
10         byte[] msgBuffer = Encoding.UTF8.GetBytes(message);
11
12         // Delai between each messages
13         Thread.Sleep(200);
14         clientToSend.GetStream().Write(msgBuffer, 0,
15             msgBuffer.Length);
16         Thread.Sleep(200);
17
18         Console.WriteLine(message);
19     }
20
21     this.MessageQueue.Clear();
21 }
```

Listing 5 – Serveur - Écriture dans la trame réseau du client

9.2.6 Déconnexion du client au serveur

Lorsqu'un client se déconnecte, il faut qu'il soit supprimé de la liste des recherches, car il est inutile de chercher un client inexistant. Ce qui est complexe, c'est le fait que le client peut se déconnecter pendant qu'une commande est reçue, mais le programme passera au-dessus de ce problème et n'écrira pas dans la trame réseau du client, ce qui évite des erreurs et des "crashes" potentiel du système. Mais la communication est assez rapide, donc pour que ce cas arrive il faut que le client se déconnecte en même temps que l'écriture dans la trame du client qui se déconnecte.

```

1 //> <summary>
2 /// Check for clients if someone is disconnected
3 </summary>
4 private void CheckForDisconnects()
5 {
6     // For every client
7     foreach (TcpClient client in this.Clients.ToArray())
8     {
9         if (this.IsDisconnected(client))
10        {
11            try
12            {
13                // Get info about the messenger
14                foreach (string name in this.ClientsNames.Keys)
15                {
16                    if
17                        (this.ClientsNames[name].ContainsValue(client))
18                    {
19                        this.Clients.Remove(client); // Remove
20                        this.ClientsNames.Remove(name); // Remove
21                        this.CleanupClient(client); // Cleanup
22                    }
23                }
24            catch (Exception) { }
25        }
26    }
27 }
```

Listing 6 – Serveur - Déconnexion du client

9.3 Envoie et réception de commandes

9.3.1 Description

Lorsque les commandes sont reçues, ces dernières sont défragmentées pour pouvoir communiquer avec la partie intelligente du serveur, et pour le mettre à jour lui ainsi que pour mettre à jour le reste du système connecté. De ce point, le code analyse les données stockées en recherchant d'abord tous les Raspberry Pi qui sont localisés dans la zone mentionnée lors de la commande. Une fois que les données sont trouvées, le programme va envoyer un message au Raspberry Pi qui contient l'objet mentionné dans la commande pour qu'il se mette à jour. Une fois l'ordre transmis, le Raspberry Pi qui est ciblé par le serveur, qui a reçu l'ordre de se mettre à jour. En retour il va lire le message et procéder tout seul à la propre mise à jour de ses données. En revanche, le fonctionnement est différent si la commande est pour récupérer des informations, telles que ; la température, l'humidité ou encore la pression atmosphérique. Car la commande est lue et renvoi directement les données traitées, qui ont été demandées par le client qui a fait la demande au serveur, tout dépend du but initial de la commande qui a été envoyé. Dans la structure schématisée du réseau ci-dessous, il y a des clients qui parlent et d'autres qui écoutent, mais malgré tout, aucun client ne connaît son voisin.

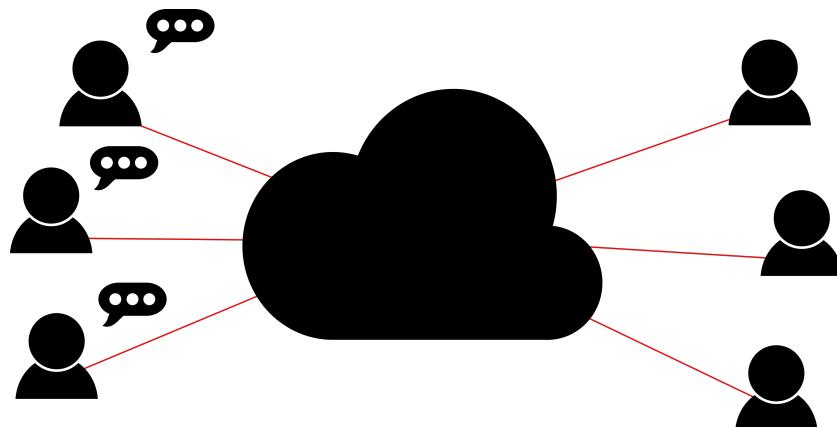


FIGURE 15 – Serveur - Représentation schématisé du réseau

9.4 Client

9.4.1 Description

Le client peut être un Raspberry Pi, ou une tablette Windows. Il donne ses informations lors de leur connexion au serveur. Ce dernier va traiter les informations reçues pour créer un objet en fonction du client, et lui donner un identifiant en fonction de son rôle. Tous les clients ne sont pas tous similaires, certains reçoivent l'ordre de mettre à jour des objets, et d'autres reçoivent l'ordre de renvoyer des informations, ainsi que certain qui servent à donner ses ordres.

Les clients utilisent tous le moyen de communication qui propre à ce type d'application. Le moyen de communication est celui de base d'une application de type UWP (Universal Windows Plateform). Il s'agit du système de communication propre à une application universelle de Visual Studio, qui utilise les "StreamSocket". Ce choix d'application compatible avec les Raspberry Pi ne peut communiquer avec les outils de base d'une application telle que de type WPF, qui utilise les bibliothèques "System", soit les "Socket" ainsi que les communications "TCP", qui ne fonctionnent pas dessus.

9.4.2 Initialisation du client

L'initialisation est différente en fonction du composant que le client possède, car les Raspberry Pi se retrouvent avec soit un PiFace Digital 2, soit un Sense HAT ou soit un système audio connecté dessus, mais le client peut aussi être une tablette Windows. Donc l'initialisation de chaque client est différente, étant donnée des technologies différentes utilisées pour le fonctionnement de chacun. En revanche, dans tous les cas, le dernier point de l'initialisation du client est la même soit l'envoi d'informations de connexion au serveur. Comme le montre le programme 7, l'initialisation de la communication avec le serveur se doit d'être après avoir généré les composants, car le programme d'initialisation créer dynamiquement la chaîne de caractères qui sera envoyée au serveur, alors que les parties de code qui suivent n'ont pas besoin de générer un quelconque composant étant donné que les objets qu'ils contiennent n'ont pas besoin de savoir quel type d'objet est branché dessus.

L'initialisation du programme qui gère le PiFace Digital 2 a absolument besoin de créer les différents composants avant de se connecter au serveur. Car le programme va générer automatiquement la chaîne de caractères en fonction des composants qui y sont initialisés. En revanche les constructeurs qui suivent n'ont pas besoin d'initialiser un quelconque composant. Donc dans l'initialisation des composants que contiennent les Raspberry Pi qui suivent celles du PiFace Digital

2, elles sont faites directement lors de la connexion sur le serveur, étant donné qu'il s'agit d'un objet défini de base dans la chaîne de caractères et qui reste fixe.

```

1 public ModelPiFaceDigital2(ViewPiFaceDigital2 paramView)
2 {
3     // Communication like Model-View
4     this.VPiFace = paramView;
5
6     // Initialize the components and add the components linked
7     // with the Raspberry
8     this.Components = new List<Component>();
9     this.Components.Add(new Light());
10
11    // Initialize the PiFace Digital 2
12    InitPiFace();
13
14    // Initialize the server communication
15    this.ComWithServer = new CommunicationWithServer(this);
15 }
```

Listing 7 – Constructeur de l'application de contrôle du PiFace Digital 2

```

1 public ModelSenseHAT(ViewSenseHAT paramView)
2 {
3     // Communication like Model-View
4     this.VSenseHAT = paramView;
5
6     // Initialize the Sense HAT (don't need to be initialized
7     // before the communication start because it's only a sensor)
8     InitializeSenseHat();
9
10    // Initialize the communication with the server
11    this.ComWithServer = new CommunicationWithServer(this);
11 }
```

Listing 8 – Constructeur de l'application de contrôle du Sense HAT

```

1 public Speicher()
2 {
3     // Initialize the synthetizer
4     this.RhSynt = new Synthetizer(this);
5     this.RhCommands = new Commands();
6
7     // Initialize the communication with the server
8     this.ComWithServer = new CommunicationWithServer(this);
9
10    // Create the speech recognition object
```

```

11     this._recoEngine = new SpeechRecognizer();
12
13     // Initialize the recognition
14     InitializeSpeechRecognizer();
15 }
```

Listing 9 – Constructeur de l'application de reconnaissance et de synthèse vocale

```

1 /// <summary>
2 /// Constructor: Initializer
3 /// </summary>
4 public MenuModel(MenuView paramView)
5 {
6     // Communication like Model-View
7     this.View = paramView;
8
9     // Initialize communication with server
10    this.ComWithServer = new CommunicationWithServer();
11 }
```

Listing 10 – Constructeur de l'application de la tablette Windows

9.4.3 Lecture et écriture des données du client

La lecture de données utilise un type d'événement asynchrone, qui agit sur le code une fois que le processus est fini. Ce dernier équivaut au *Thread.Sleep* de la bibliothèque *System.Threading* que l'on retrouve dans les applications de base de "Visual Studio" tel que les applications en mode *Console*, qui met le programme en mode "repos". Cette évènement quant à lui, il porte le nom de *await* et est souvent utilisé dans les applications universelles de Microsoft. L'évènement *await* est un moyen de mettre le programme en mode "attente d'information", donc en attente des informations venant du serveur pour ce projet. Dès que le client reçoit une quelconque modification dans sa trame réseau, le programme reprend à partir du *await*. De ce point, ressort des données contenu dans un tableau de byte, soit un tableau de données allant de 0 à 255 par valeur contenu dans le tableau.

Dans le cas ci-dessous, il s'agit de la méthode de base de l'utilisation de la lecture de trame. Une fois que la valeur en byte est récupérée, il est simple de la convertir dans n'importe quel type de valeur que ce soit du double à la chaîne de caractères. Le processus reste le même à partir du moment où les méthodes de conversion du *DataReader* existent. Dans ce cas-là, la conversion demandée est une chaîne de caractères.

```
1 // Get the actual stream
```

```

2 DataReader dataReader = new DataReader(socket.InputStream);
3 dataReader.InputStreamOptions = InputStreamOptions.Partial;
4 // Set the size of the message
5 var messageLength = dataReader.UnconsumedBufferLength;
6 uint stringBytes = messageLength;
7
8 try
9 {
10     // Read modification in the stream
11     stringBytes = await
12         dataReader.LoadAsync(MESSAGE_FULL_LENGTH);
13
14     // read message
15     string messageRead = dataReader.ReadString(stringBytes);
16 }
16 catch (Exception){}

```

Listing 11 – Client - Lecture de la trame du client

L’écriture de données fonctionne de la même façon en utilisant le même type d’évènement asynchrone. Ce qui diffère de la lecture, est que cette fois-ci, le programme doit convertir la valeur afin de l’écrire dans un tableau de byte, ce que le *DataWriter* permet de faire. Ce dernier contient plusieurs méthodes de conversion d’un type vers un tableau de byte, où il suffit d’inscrire la valeur avec la bonne méthode de conversion pour écrire dans la trame réseau. Par la suite, le serveur regarde si la trame du client est différente, si oui il la lit comme un nouveau message.

Dans le cas qui suit, il s’agit de la façon de base d’utilisation du *DataReader* pour écrire dans la trame. Pour ce projet, les données enregistrée dans la trame sont toutes des chaînes de caractères.

```

1 DataWriter dataWriter = new DataWriter(socket.OutputStream);
2 var len = dataWriter.MeasureString(message); // Gets the UTF-8
      string length.
3 dataWriter.WriteInt32((int)len);
4 dataWriter.WriteString(message);
5 var ret = await dataWriter.StoreAsync();
6 dataWriter.DetachStream();

```

Listing 12 – Client - Écriture sur trame du client

9.4.4 Problèmes rencontrés et solutions

La lecture de la trame m'a pris beaucoup de ressources, à devoir comprendre son fonctionnement et faire des recherches sur comment résoudre certains problèmes qui surviennent rapidement comme des "crashs" système.

L'écriture est fonctionnelle, mais une fois que le serveur lit la trame, des caractères s'ajoutent en début de chaîne. Donc il a fallu ajouter un filtre supplémentaire. J'ai essayé de contourner le problème, mais je ne suis pas resté dessus plus d'une heure. La façon de corriger ce problème est de simplement à ajouter un caractère spécial comme "@" qui permet de séparer l'utile de l'inutile au moment de la retranscription du message sur le serveur.

9.5 Carte électronique entrée et sortie audio

9.5.1 Description

Pour que ce projet soit totalement fonctionnel il faut avoir une entrée audio ainsi qu'une sortie audio. De ce fait le Raspberry Pi ne dispose que d'un port audio, et il s'agit seulement d'une sortie analogique pour y brancher des haut-parleurs. Donc ce projet demande de pouvoir développer un circuit imprimé qui vient se brancher sur les PIN du Raspberry Pi 3 modèle B, et qui fait office d'entrée audio grâce à une combinaison avec de l'électronique et de la communication en SPI. Le réel problème est que les entrées disponibles sur un Raspberry Pi, elles sont toutes des entrées digitales et non analogiques qui sont le format du signal de sortie d'un microphone. C'est pourquoi la carte se trouve munie d'un convertisseur Analogique à Digital pour pouvoir lire le signal venant du microphone.

9.5.2 Schéma électronique

Le schéma électronique n'est pas la dernière version étant donné qu'il a été abandonné à cause de diverses raisons citées plus tard. Malgré tout, le développement à quand même été fait, et l'avancement s'arrête au schéma.

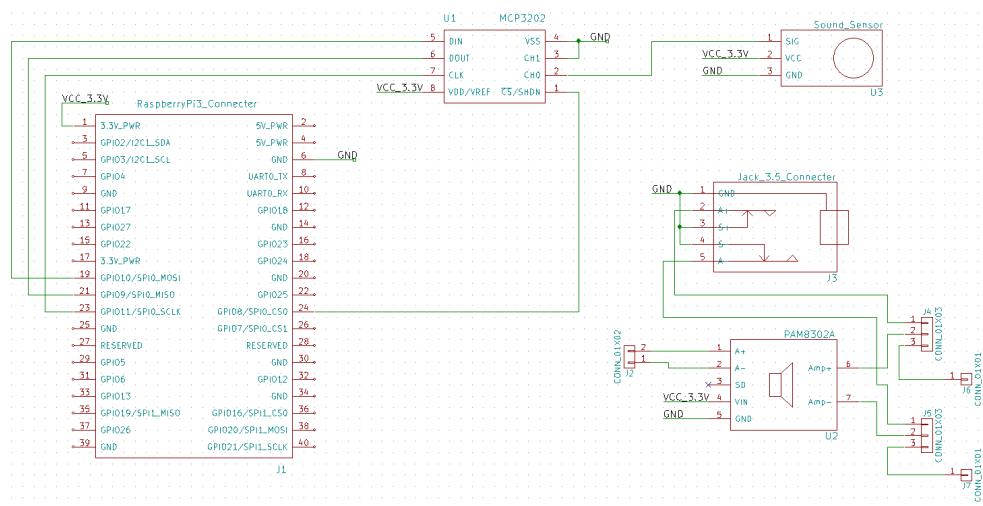


FIGURE 16 – Circuit imprimé - Premier schéma électronique

9.5.3 Composants et méthodes utilisées

- Convertisseur AD/DC - MCP3202⁷
- Amplificateur opérationnel - LM741⁸
- Connecteur 2x20 pôles
- Connecteur 1x3 pôles
- Connecteur 1x5 pôles
- Connecteur jack audio
- Interrupteur 2x3 pour commuter entre différente sortie audio

9.5.4 Problèmes rencontrés et solutions

Il était bien trop difficile de développer un circuit imprimé qui soit compatible avec le Raspberry Pi, et qui fasse office de convertisseur d'un signal audio en une chaîne de caractères, car il a fallu étudier plus profondément le tout, soit la conversion d'un signal analogique en un signal digital, tout en ayant de la précision afin de pouvoir avoir une retranscription précise pour que par la suite, le programme fasse office de reconnaissance vocale.

7. <http://www.farnell.com/datasheets/1669376.pdf>
8. <http://www.ti.com/lit/ds/symlink/lm741.pdf>

9.6 Reconnaissance de commandes vocale

9.6.1 Description

La reconnaissance vocale est une façon de donner des ordres au Raspberry Pi, le souci réel est la retranscription étant donné des risques d'erreurs qui peuvent se faire au moment de la retranscription dans la langue adéquate, soit le français. De nouveau, le Raspberry Pi ne peut contenir des bibliothèques tel que *SpeechLib* ou *System.Speech* qui sont les deux des bibliothèques de base de Microsoft. La solution est d'utiliser une nouvelle bibliothèque qui est fournie lors de l'installation des méthodes de programmation universelles de Visual Studio 2015.

9.6.2 Fonctionnement de la grammaire

Au cours de ce projet, il a fallu faire des recherches sur le fonctionnement de la grammaire française personnalisée, à implémenter dans un projet qui utilise la reconnaissance vocale comme technologie. Au début, la reconnaissance vocale fonctionnait avec des listes de mots pour filtrer l'indésirable, et de garder ce qui semblait être le mot qui se rapproche le plus du résultat souhaité. Après plusieurs recherches, et avec l'aide de plusieurs forums qui expliquaient comment fonctionne la reconnaissance vocale, la meilleure des solutions était de créer un fichier XML avec comme balise principale "grammar", pour par la suite créer sa propre grammaire avec dans quel ordre les mots doivent défiler. Pour ce projet, il a fallu apprendre à créer un fichier de grammaire et apprendre à utiliser les différentes balises contenues dans la balise principale.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <grammar version="1.0" xml:lang="fr-FR" mode="voice"
  root="toplevel"
3 tag-format="semantics/1.0"
4   xml:base="http://www.contoso.com/"
5   xmlns="http://www.w3.org/2001/06/grammar"
6
7     xmlns:sapi="http://schemas.microsoft.com/Speech/2002/06/SRGSExtensions">
8   <rule id="toplevel" scope="public">
9     <one-of>
10       <item>
11         <ruleref uri="#Raspi"/>
12       </item>
13       <item>
14         <ruleref uri="#Meteo"/>
15       </item>
16       <item>
17         <ruleref uri="#Action"/>
18       </item>
```

```
18      </one-of>
19  </rule>
20
21  <rule id="Raspi">
22    <example> raspi </example>
23
24    <item> raspi </item>
25  </rule>
26
27  <rule id="Meteo">
28    <example> quel est la temperature </example>
29    <example> quel est le taux d'humidite </example>
30    <example> quel est le niveau de pression </example>
31    <example> quel est l'état</example>
32
33  <one-of>
34    <item> temperature </item>
35    <item> humidité </item>
36    <item> pression </item>
37    <item> état </item>
38  </one-of>
39  <ruleref uri="#Localisation"/>
40 </rule>
41
42  <rule id="Action">
43    <example> allume la lumière</example>
44    <example> éteint la lumière</example>
45    <example> monte le store</example>
46    <example> descend le store</example>
47    <example> ouvre le store</example>
48    <example> ferme le store</example>
49
50  <one-of>
51    <item> allume </item>
52    <item> éteint </item>
53    <item> monte </item>
54    <item> descend </item>
55    <item> ouvre </item>
56    <item> ferme </item>
57  </one-of>
58
59  <ruleref uri="#Composant"/>
60 </rule>
61
62  <rule id="Composant">
63    <example> la lumière </example>
64    <example> les lumières </example>
65    <example> le store </example>
66    <example> les stores </example>
```

```
67
68    <one-of>
69        <item> lumiere </item>
70        <item> store </item>
71    </one-of>
72    <ruleref uri="#Localisation"/>
73 </rule>
74
75 <rule id="Localisation">
76     <example> du salon </example>
77     <example> de la cuisine </example>
78     <example> du bureau </example>
79     <example> de la chambre </example>
80     <example> des toilettes </example>
81     <example> de la salle de bain </example>
82     <example> de la maison </example>
83
84     <item repeat="0-1"> salon </item>
85     <item repeat="0-1"> cuisine </item>
86     <item repeat="0-1"> bureau </item>
87     <item repeat="0-1"> chambre </item>
88     <item repeat="0-1"> toilette </item>
89     <item repeat="0-1"> salle de bain </item>
90     <item repeat="0-1"> maison </item>
91 </rule>
92 </grammar>
```

Listing 13 – Reconnaissance vocale - Grammaire personnalisée de la reconnaissance vocale utilisée pour RaspiHome

9.6.3 Problèmes rencontrés et solutions

Ce projet a demandé de pouvoir utiliser la reconnaissance vocale. Mais le problème qui est arrivé plutôt rapidement, est la difficulté à lire un signal audio sur un Raspberry Pi qui n'a pas de composant prévu à cet effet directement implanté sur le circuit imprimé, soit une entrée analogique. La difficulté de ce problème m'a poussé à faire des recherches plus approfondie sur le sujet de la lecture d'un signal audio sur un Raspberry Pi. La majorité des réponses sont d'utiliser un microphone qui se branche en USB, car la méthode qui allait être utilisée pour ce projet était bien trop complexe et il y avait peu d'informations sur internet de comment faire un échantillonage de données sur un Raspberry Pi qui fonctionne avec Windows Iot.

9.7 Synthèse vocale

9.7.1 Description

Le projet contient une partie que l'on peut nommer "Intelligence Artificielle", mais qui n'en est pas une à proprement parler, car étant donnée que les commandes vocales sont disponibles, il n'y a aucun support sur quoi s'appuyer pour afficher des valeurs tel que la température qui est juste une mesure instantanée sur le moment présent et qui n'a pas besoin de support visuel hormis sur tablette où il est possible d'implémenter facilement une quelconque valeur. Donc pour le développement de ce projet et aussi pour une meilleure esthétique, RaspiHome traite les informations qu'il reçoit pour les retransmettre par la suite. La pseudo IA répond dès que l'on l'appelle, dès que l'on dit simplement "Raspi", et de ce point, le Raspberry Pi se met en mode écoute et est prêt à répondre à la demande, en bien si la demande peut être traitée ou en mal si la commande n'est pas juste.

9.7.2 Fonctionnement de la synthèse

Le fonctionnement de la synthèse est vraiment simple à comprendre, il s'agit de la technologie "TextToSpeech" ou "TTS". Cette technologie agit comme l'humain lorsqu'il doit lire un texte à haute voix. En premier lieu, il va lire le texte, et par la suite il va répéter ce qu'il a lu, mais cette fois-ci à haute voix. Donc le programme lui ne fait que d'écrire ce que le synthétiseur doit dire. Le point important de la synthèse, est lors de la lecture d'une longue phrase et qui contient des chiffres à virgule, il va les mentionner lors de sa synthèse, donc il n'y a pas besoin de retranscrire soit même un caractère phonétiquement.

9.7.3 Initialisation de la synthèse vocale

L'utilisation de la synthèse vocale sur une application de type universelle de Visual Studio est totalement différente de la version que l'on retrouve sur une application de base de Visual Studio 2015. Pour pouvoir travailler la synthèse, il faut utiliser la bibliothèque qui est fournie avec l'application qui est *Windows.Media.SpeechSynthesis*. Une fois implémentée au code, je me suis servi d'un exemple officiel pour initialiser correctement le synthétiseur vocal, ainsi que de plusieurs exemples trouvés sur l'aide officiel de Visual Studio pour générer cette synthèse.

```
1 using Windows.Media.SpeechSynthesis;  
2 using System.Threading;  
3 using System.Threading.Tasks;
```

```
4
5 namespace RaspiHomeSpeechNSynthetize
6 {
7     public class Synthetizer
8     {
9         #region Fields
10        #region Constants
11        // Change value when new update ("en" to "fr")
12        private const string LANGUAGE_SELECTION = "en";
13        private const double TIME_TO_WAIT = 3.0;
14        #endregion
15
16        #region Variable
17        private SpeechSynthesizer _voice = null;
18        #endregion
19        #endregion
20
21        #region Properties
22        #endregion
23
24        #region Constructor
25        /// <summary>
26        /// Constructor: Initialize
27        /// </summary>
28        public Synthetizer()
29        {
30            this._voice = new SpeechSynthesizer();
31        }
32        #endregion
33
34        #region Methods
35        /// <summary>
36        /// Allow the raspi to let her talk
37        /// </summary>
38        /// <param name="messageToSay"> sentence to say </param>
39        private async void RaspiTalk(string messageToSay)
40        {
41            // Get the output element (audio jack)
42            MediaElement mediaElement = new MediaElement();
43            SpeechSynthesizer synth = new SpeechSynthesizer();
44
45            // Set the default language
46            foreach (VoiceInformation vInfo in
SpeechSynthesizer.AllVoices)
47            {
48                if (vInfo.Language.Contains(LANGUAGE_SELECTION))
49                {
50                    synth.Voice = vInfo;
51                    break;
```

```
52         }
53     else
54         synth.Voice = vInfo;
55     }
56
57     SpeechSynthesisStream synthStream = await
58     synth.SynthesizeTextToStreamAsync(messageToSay);
59
60     mediaElement.SetSource(synthStream,
61     synthStream.ContentType);
62     // 0 = min / 1 = max
63     mediaElement.Volume = 1;
64     mediaElement.Play();
65
66     // Work like Thread.Sleep(TIME_TO_WAIT)
67     await
68     Task.Delay(TimeSpan.FromSeconds(TIME_TO_WAIT));
69   }
70 }
```

Listing 14 – Synthèse vocale - Initialisation du synthétiseur vocale et activation de la synthèse

9.7.4 Problèmes et résolution

Le plus gros problème du projet est que sur les Raspberry Pi la langue de la synthèse vocale par défaut est l'anglais américain. Donc le Raspberry Pi dispose seulement de trois types de voix différentes, mais aucune n'est française. La solution trouvée est de se connecter en PowerShell sur le Raspberry Pi depuis un ordinateur distant et d'ouvrir l'emplacement de fichier de la synthèse vocale sur les deux (ordinateur et Raspberry Pi). Par la suite il suffit de copier/coller le dossier "fr-FR" contenu dans le dossier C:/Windows/Speech_OneCore/Engines/TTS/ dans le même dossier, mais celui du Raspberry Pi. Par la suite il faut reproduire la même manipulation mais cette fois dans le dossier C:/Windows/System32/Speech_OneCore/Common/ et copier/coller le dossier "fr-FR" dans le dossier C:/Windows/System32/Speech_OneCore/ pour ensuite le déplacer à l'aide de la commande "mv" (move) de PowerShell, étant donné que Windows protège l'écriture de fichier dans le dossier *System32*.

Le second problème est que la langue française est bien implémentée, mais que le programme n'arrive pas à exécuter le code, car la langue utilisée ne peut pas être lue. La solution a été trouvée sur un forum officiel de Microsoft, où un administrateur du forum répond à une question en disant que plusieurs langues utilisées pour la synthèse vocale seront ajouté au cours des prochaines mises à jour de Windows IoT. Ce n'est pas une réelle solution donc pour le moment la seule synthèse vocale utilisable est l'anglais.

9.8 Programme pour tablette

9.8.1 Description

Pour que le projet soit complet, il faut pouvoir communiquer avec les Raspberry Pi, même si la commande vocale est désactivée ou ne fonctionne pas. La différence entre la reconnaissance vocale et l'application pour tablette Windows, est que toutes les informations sont déjà indiquées sur la tablette, il suffit donc d'appuyer pour avoir un résultat. En plus de ce point, l'application est utile pour les malentendants (sourd) et les aphasiques (muet), étant donné que le fonctionnement principal de la reconnaissance vocale et de la synthèse vocale, est le fait de ne pas utiliser de support visuel mais de tout faire par la voix et par l'ouïe, tandis que l'application utilise la vue et le toucher pour arriver aux même résultats.

9.8.2 Interface graphique pour tablette

L'interface graphique est dynamique, avec un affichage qui varie selon le menu sélectionné. Cette interface dispose de quatre boutons de base ; un qui est utilisé pour envoyer des commandes aux autres Raspberry Pi, un qui regroupe la configuration totale des objets qui sont connectés sur le serveur, un qui donne les informations sur les mesures de faite par les capteurs avec des graphiques, et enfin, un qui s'occupe de gérer certains réglages que l'utilisateur peut modifier lui-même.

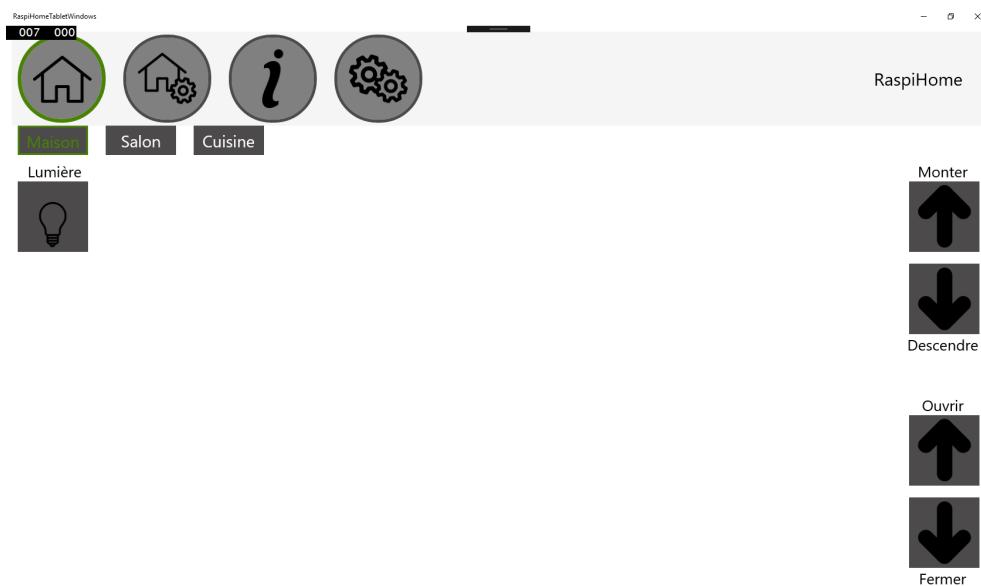


FIGURE 17 – Interface graphique - Gestion maison

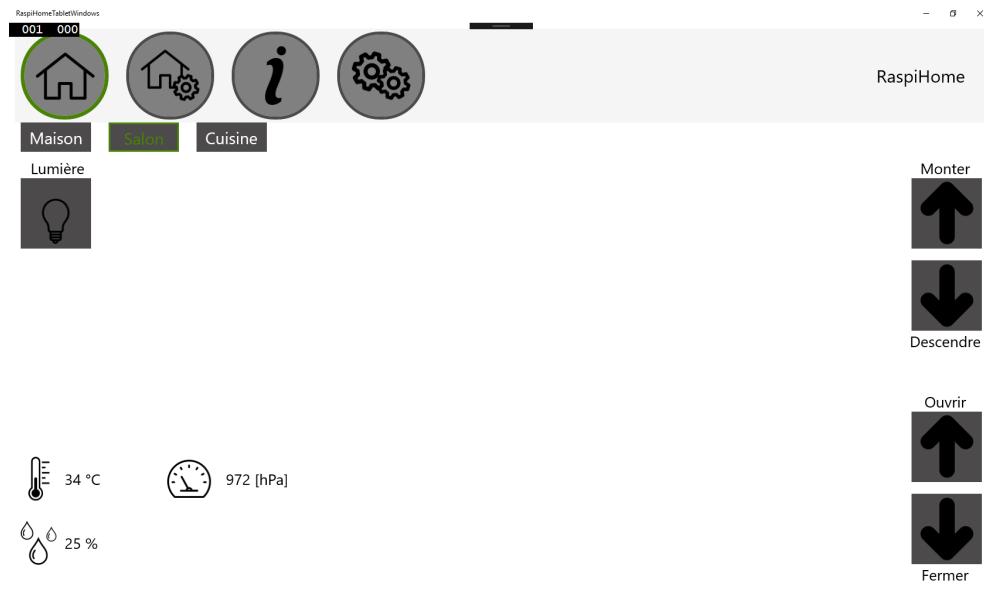


FIGURE 18 – Interface graphique - Gestion salon

9.8.3 Fonctionnement de l'interface graphique

Le fonctionnement de l'interface graphique a été développé pour que tout utilisateurs puissent l'utiliser. Il s'agit d'une interface avec un fonctionnement complexe en fond, mais avec une utilisation vraiment simple pour que n'importe quelle personne utilisant cette application, n'est pas à se demander à chaque fois quel bouton fait quoi.

L'application comporte en premier lieu un menu qui fait office de barre d'outils. Ce dernier permet de naviguer entre diverses fenêtres, toutes différentes les unes des autres.

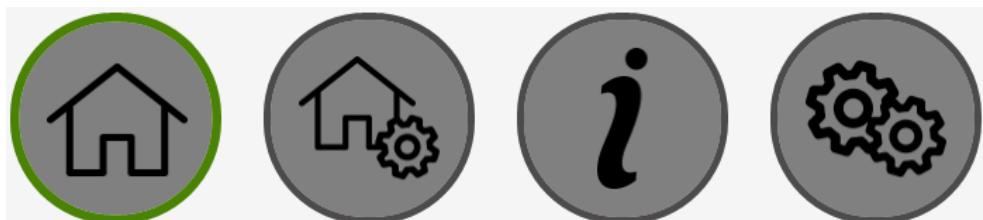


FIGURE 19 – Interface graphique - Barre d'outils de navigation

Une fois le premier bouton sélectionné, il va apparaître une seconde barre d'outils en dessous de la première, qui cette fois-ci permet de sélectionner la pièce avec

laquelle on veut interagir.



FIGURE 20 – Interface graphique - Barre d’outils de choix de localisation

De ce point, il est facile d’implémenter des boutons afin de d’activer ou de désactiver du matériel, tel que les lumières et les stores.

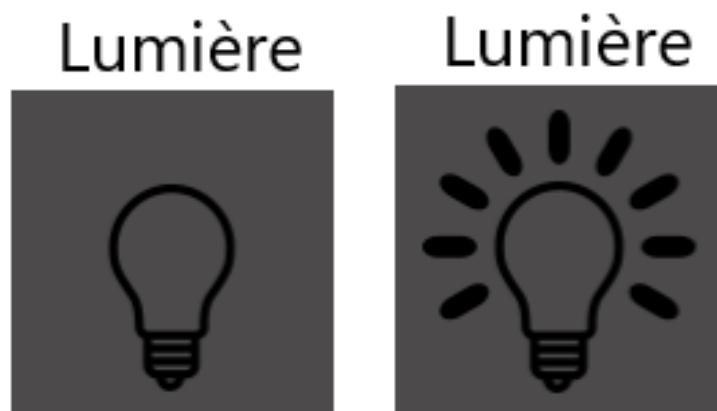


FIGURE 21 – Interface graphique - Bouton de contrôle de la lumière

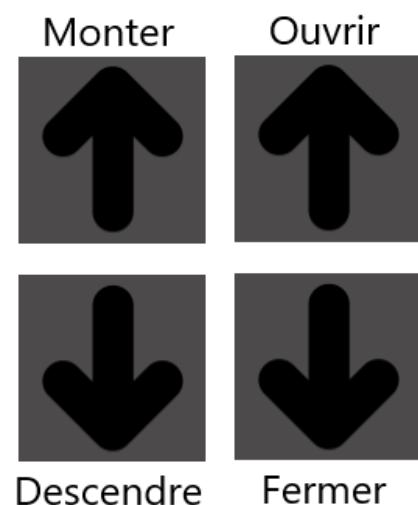


FIGURE 22 – Interface graphique - Boutons de contrôle du store

Le dernier point à prendre en compte est pour l'affichage des valeurs de température. Dans l'onglet maison, il n'apparaît pas car il est impossible de faire une moyenne des valeurs de tous les capteurs que le foyer contient. C'est pourquoi l'affichage de ces valeurs n'est disponible que dans les pièces où il y a une Sense HAT.

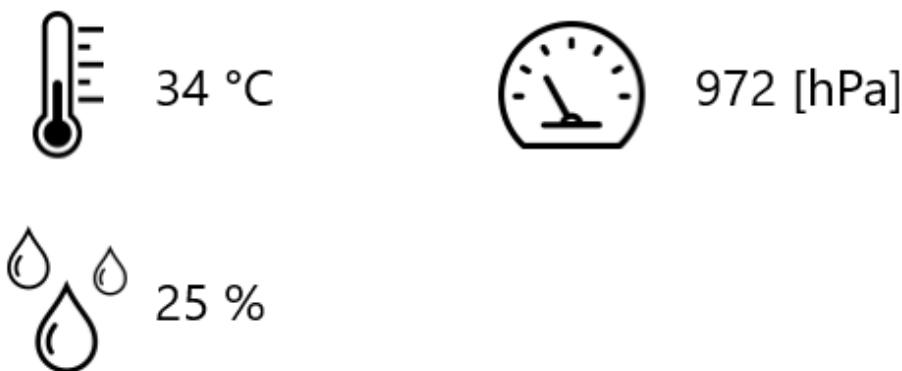


FIGURE 23 – Interface graphique - Affichage valeurs mesurées

9.8.4 Envoi et lecture de données

Le réel problème à surmonter avec cette application est la communication d'informations avec le serveur, sans créer et recréer un client à chaque fois que l'utilisateur change de menu. Donc pour que l'envoi soit fait entre deux classes qui ne se connaissent pas du tout, l'envoi et la lecture de données se font grâce à un procédé qui mélange la "sauvegarde de données" dans la mémoire locale et d'un minuteur qui va lire ces données sauvegardées toutes les tant de millisecondes.

```
1 Windows.Storage.ApplicationDataContainer localSettings =  
    Windows.Storage.ApplicationData.Current.LocalSettings;  
2  
3 /// <summary>  
4 /// Initialize at the start (check if the sense hat exist)  
5 /// </summary>  
6 private void InitializeState()  
7 {  
8     // Update state room  
9     var locationName =  
    localSettings.Values["NameButtonClicked"];  
10  
11     if (locationName != null)
```

```

12     localSettings.Values["SendMessageToServer"] = "etat " +
13         locationName;
14     else // Default reference
15         localSettings.Values["SendMessageToServer"] = "etat
16             salon";
17 }
```

Listing 15 – Interface graphique - Écriture dans la mémoire locale

La lecture de données a besoin d'avoir un évènement extérieur, tel que le "timer" pour forcer la lecture. Dès lors qu'il détecte une modification dans son espace de stockage local, il lit cette valeur et met dans une variable qui sera traitée.

```

1 Windows.Storage.ApplicationDataContainer localSettings =
2     Windows.Storage.ApplicationData.Current.LocalSettings;
3
4 /// <summary>
5 /// Constructor: Initializer
6 /// </summary>
7 public CommunicationWithServer()
8 {
9     Connect();
10
11     this._dTimer = new DispatcherTimer();
12     this._dTimer.Interval = new TimeSpan(10);
13     this._dTimer.Tick += _dTimer_Tick;
14
15     this._dTimer.Start();
16 }
17
18 private void _dTimer_Tick(object sender, object e)
19 {
20     if (localSettings.Values["SendMessageToServer"] != null)
21     {
22         var messageToSend =
23             localSettings.Values["SendMessageToServer"];
24         if(messageToSend.ToString() != "")
25             this.SendCommandToServer(messageToSend.ToString());
26         localSettings.Values.Remove("SendMessageToServer");
27     }
28 }
```

Listing 16 – Interface graphique - Lecture dans la mémoire locale

9.8.5 Améliorations possible de l'application tablette

Ajouter un système de sauvegarde de données courantes, tel que la sauvegarde des données météorologiques (température, humidité et pression) afin de générer des graphiques pour l'onglet information

Ajouter un système de mémorisation d'appareils connecté aux différents Raspberry Pi, tel que les lumières, les stores, les capteurs et les microphones ainsi que la localisation de chacun comme le salon, la cuisine ou autres.

Ajouter un espace de configuration pour les utilisateurs où ils peuvent modifier l'interface graphique, régler l'heure de la mise en veille du système et l'heure de réveil de chacun des occupants.

10 Application de gestion du Sense HAT

10.1 Description

L'application de gestion du Sense HAT est une partie importante, étant donné que le projet demande de pouvoir récupérer des informations telles que la température. Il est possible d'utiliser des composants externes et de développer des cartes électroniques pour avoir le même résultat, mais étant donné que le produit est déjà créé et que ce dernier est fonctionnel, il est plus bénéfique de l'utiliser. Ce programme transmet directement au serveur les informations qu'il mesure instantanément. C'est dernier qui s'occupe par la suite de renvoyer à la bonne personne les informations que le Sense HAT a mesurées.

10.2 Contrôle du Sense HAT

Pour contrôler le Sense HAT, il suffit d'inscrire la commande qui suit dans l'indice de commande prévu à cet effet, soit l'indice de commande NuGet :

```
PM> Install-Package Emmellsoft.IoT.RPi.SenseHat
```

Listing 17 – Sense HAT - Ajout de la bibliothèque du Sense HAT

Cette ligne de commande est à insérer au début d'un projet qui utilise un Sense HAT. Pour ce faire, il faut, une fois sur le projet créer, ajouter cette ligne de commande dans l'indice de commande de NuGet, que l'on retrouve dans l'onglet "Outils" qui est intégré à Visual Studio. À partir de là, il est simple d'utiliser le Sense HAT pour faire des mesures.

10.3 Gestion du Sense HAT

Une fois que le NuGet a fini de faire son installation, le Sense HAT est utilisable sur le Raspberry Pi. Pour le développement de l'application qui utilise le Sense HAT, je me suis servi d'un projet C# déjà existant comme référence⁹. Ce dernier récupère les données des capteurs pour simplement les afficher. Dans ce programme, le fonctionnement est le même. Le Sense HAT donne ses informations au programme, et ce dernier va ensuite les mettre en forme pour les envoyer au serveur dans une chaîne de caractères spécifique.

```

1 using Emmellsoft.IoT.Rpi.SenseHat;
2
3 namespace RaspiHomeSenseHAT
4 {
5     public class ModelSenseHAT
6     {
7         #region Variables
8         // Sense HAT librairy
9         private ISenseHat _senseHat;
10        private ISenseHatDisplay _senseHatDisplay;
11        SenseHatData _data;
12
13        // Set default color matrix to OFF
14        private Color _uiColor = Color.FromArgb(0, 0, 0, 0);
15        #endregion
16        #endregion
17
18        #region Constructors
19        /// <summary>
20        /// Constructor: Initializer
21        /// </summary>
22        /// <param name="paramView"></param>
23        public ModelSenseHAT(ViewSenseHAT paramView)
24        {
25            // Communication like Model-View
26            this.VSenseHAT = paramView;
27
28            // Initialize the Sense HAT (don't need to be
29            initialized before the communication start because it's only
30            a sensor)
31            InitializeSenseHat();
32
33            // Initialize the communication with the server
34            this.ComWithServer = new
35            CommunicationWithServer(this);

```

9. <https://www.hackster.io/laserbrain/windows-iot-sense-hat-10cac2>

```
33     }
34 #endregion
35
36 #region Methods
37 /// <summary>
38 /// Initialize the Sense HAT
39 /// </summary>
40 public async void InitializeSenseHat()
41 {
42     this._senseHat = await
43     SenseHatFactory.GetSenseHat();
44     this._senseHatDisplay = this._senseHat.Display;
45     this._senseHatDisplay.Fill(_uiColor);
46
47     SetValues();
48 }
49 /// <summary>
50 /// Set the value get with sensor
51 /// </summary>
52 public void SetValue()
53 {
54     // Update values
55     this._senseHat.Sensors.HumiditySensor.Update();
56     this._senseHat.Sensors.PressureSensor.Update();
57     this._senseHatDisplay.Update();
58
59     // Set values
60     this._data = new SenseHatData();
61     this._data.Temperature =
62         this._senseHat.Sensors.Temperature;
63         this._data.Humidity =
64             this._senseHat.Sensors.Humidity;
65             this._data.Pressure =
66                 this._senseHat.Sensors.Pressure;
67             }
68     }
69 }
```

Listing 18 – Sense HAT - Initialisation du Sense HAT et fonctionnement interne

10.4 Envoi des valeurs formatées

Lorsque le serveur informe le programme de se mettre à jour, ce dernier ordonne au Sense HAT de se mettre à jour à son tour. Pour que la commande soit complète, il ne reste plus qu'à générer une chaîne de caractères avec un certain format, et de l'envoyer avec les valeurs des capteurs qui ont été mesurée instantanément.

```
1 /// <summary>
2 /// Send the values with a special format:
3 ///   "TEMP=x;HUMI=y;PRES=z"
4 /// Values are rounded
5 /// </summary>
6 /// <returns></returns>
7 public string SendValues()
8 {
9     // Update values of sensors
10    SetValues();
11
12    return "TEMP=" +
13        Math.Round((decimal)this._data.Temperature) + ";" + "HUMI=" +
14        Math.Round((decimal)this._data.Humidity) + ";" + "PRES=" +
15        Math.Round((decimal)this._data.Pressure);
16 }
```

Listing 19 – Sense HAT - Format de l'envoi des valeurs des capteurs

11 Application de gestion du PiFace Digital 2

11.1 Description

L'application de gestion du Piface Digital 2 est tout aussi importante que l'application de gestion du Sense HAT. Cette application permet de gérer tout l'équipement électronique que le projet demande de pouvoir contrôler, tel que les lumières et les stores et plus encore. Chaque Raspberry Pi monter avec un PiFace Digital 2 peuvent contrôler toutes les lumières et tous les stores d'une même pièce. Le contrôle de la lumière ce fait grâce au relais qui sont implémentés sur le circuit imprimé et qui sur lesquels peuvent être branchés de l'équipement à forte tension telle que du matériel branché sur une prise secteur. Le contrôle des stores quant à lui, ce fait avec les sorties Digitales du PiFace Digital 2, étant donné que le fonctionnement d'un moteur store se doit pouvoir tourner dans les deux sens pour monter et descendre ces derniers.

11.2 Contrôle du PiFace Digital 2

Pour utiliser le PiFace Digital 2 sur un Raspberry Pi il faut avant tout être muni du code qui contrôle ce module. Ce code est disponible sur le site du fournisseur, et initialise le composant pour ne laisser place qu'à l'utilisation du produit. Ce programme initialise d'abord le MCP23S17 qui contient les méthodes de lecture et d'écriture sur les PIN du PiFace Digital 2. Le second élément d'initialisation est plus basé pour aider l'utilisateur car il regroupe les entrées sorties du PiFace Digital 2 de façon à ce que les entrées soient nommée INx, les sorties LEDx et enfin les relais A et B.

11.3 Recherche et application de modification

Une fois que le serveur a trouvé les bons clients à mettre à jour, ce même serveur écrit dans la trame du client la phrase qu'il a reçue. Cette phrase est lue par le client à qui le serveur a envoyé le message. Et ce client va appliquer à nouveau un filtre pour savoir ce que la phrase a pour but. En premier lieu, le programme va récupérer tous les mots importants tels que le composant et l'action qui va avec.

La décomposition de la phrase est basique. Le programme applique la mise en commun avec des listes de mots connus par ce programme.

¹ // Words know by the program

```
2 private List<string> _raspiHomeComponentKnown = new
3     List<string>()
4 {
5     "lumiere", "lumieres",
6     "store", "stores",
7     "television", "televisions",
8     "porte", "portes",
9     "fenetre", "fenetres",
10    };
11 private List<string> _raspiHomeActionKnown = new List<string>()
12 {
13     "allumer", "allume",
14     "eteindre", "eteins",
15     "monter", "monte",
16     "descendre", "descends",
17     "stopper", "stop",
18     "ouvrir", "ouvre",
19     "fermer", "ferme",
20     "stopper", "stop",
21    };
22
23 /// <summary>
24 /// Find location exist
25 /// </summary>
26 /// <param name="sentence"> sentence order</param>
27 /// <returns> return the action linked to the action word
28 //      </returns>
29 private string GetActionFromSentence(string sentence)
30 {
31     string result = "";
32     string[] words = sentence.ToLower().Split(' ');
33
34     foreach (var word in words)
35     {
36         if (this._raspiHomeActionKnown.Contains(word))
37         {
38             result = word;
39             break;
40         }
41     }
42
43     return result;
44 }
45 /// <summary>
46 /// Get the component called
47 /// </summary>
48 /// <param name="sentence"> sentence order </param>
```

```

49 /// <returns> return the component linked to the component word
50 </returns>
51 private string GetComponentFromSentence(string sentence)
52 {
53     string result = "";
54     string[] words = sentence.ToLower().Split(' ');
55
56     foreach (var word in words)
57     {
58         if (this._raspiHomeComponentKnown.Contains(word))
59         {
60             result = word;
61             break;
62         }
63     }
64
65     return result;
66 }
```

Listing 20 – Piface Digital 2 - Décomposition de la phrase

L'action va permettre de connaître quel type de propriété l'application doit viser pour appliquer la modification.

```

1 // KEY=[ACTION], VALUE[KEY=[PROPERTY], VALUE=[VALUE TO SET THE
2 private Dictionary<string, Dictionary<string, bool>>
3     _raspiBooleanCommandTranslation = new Dictionary<string,
4     Dictionary<string, bool>>()
5 
6     { "allume", new Dictionary<string, bool> { { "IsOn", true } }
7     },
8     { "allumer", new Dictionary<string, bool> { { "IsOn", true } } },
9     { "eteins", new Dictionary<string, bool> { { "IsOn", false } } },
10    { "eteindre", new Dictionary<string, bool> { { "IsOn", false } } },
11    { "monte", new Dictionary<string, bool> { { "IsUp", true } }
12    },
13    { "monter", new Dictionary<string, bool> { { "IsUp", true } } },
14    { "descends", new Dictionary<string, bool> { { "IsDown", true } } },
15    { "descendre", new Dictionary<string, bool> { { "IsDown", true } } },
16    { "stop",new Dictionary<string, bool> { { "IsStop", true } } },
```

```

13     { "stopper",new Dictionary<string, bool> { {"IsStop",true} }
14 },
15
16 /// <summary>
17 /// Read properties value of classes
18 /// </summary>
19 /// <param name="actionName"> name used to change the good
   property </param>
20 /// <returns> return the name of the property to change the
   value </returns>
21 private string ReadValueOfSelectedComponent(string actionName)
22 {
23     string result = "";
24
25     foreach (var actionKeys in
      this._raspiBooleanCommandTranslation.Keys)
26     {
27         if (actionKeys == actionName)
28         {
29             // Find the Value of the dictionary trough the
               inner dictionary to get the first value
30             result =
      this._raspiBooleanCommandTranslation[actionName].First().Key;
31             break;
32         }
33     }
34
35     return result;
36 }
```

Listing 21 – Piface Digital 2 - Recherche de la propriété à modifier

La recherche de toutes ces valeurs va permettre de pouvoir changer les valeurs des propriétés contenues dans les classes Light et Store, mais avant cela, il faut encore définir la classe à modifier. Pour cela, le procédé reste le même que précédemment, à la différence que le programme applique une traduction suivit d'une recherche des classes courantes contenu dans le programme pour trouver la bonne classe.

```

1 // Word translation
2 private Dictionary<string, string> _raspiLanguageTranslation =
   new Dictionary<string, string>()
3 {
4     { "lumiere", "Light" }, { "lumieres", "Light" },
5     { "store", "Store" }, { "stores", "Store" },
6 };
```

```

7
8 /// <summary>
9 /// Find all client who have the object in the sentence
10 /// </summary>
11 /// <param name="componentName"></param>
12 /// <returns>the object type</returns>
13 private Type GetComponentType(string componentName)
14 {
15     Type result = null;
16     Type[] types =
17         typeof(Component).GetTypeInfo().Assembly.GetTypes();
18     foreach (var typeOfComonent in types)
19     {
20         if (typeOfComonent.Name ==
21             this._raspiLanguageTranslation[componentName])
22         {
23             result = typeOfComonent;
24             break;
25         }
26     }
27     return result;
28 }
```

Listing 22 – Piface Digital 2 - Traduction et recherche du composant

Enfin, une fois que toutes ces valeurs ont été définies et que la classe à modifier le soit aussi, il ne reste plus qu'à faire un lien dynamique entre le code et la propriété de la classe à modifier, pour que le programme fasse le travail tout seul en fond.

```

1 /// <summary>
2 /// Set the value to be writed on the PiFace
3 /// </summary>
4 /// <param name="message"> message read from the server </param>
5 public void SetValue(string message)
6 {
7     // Initialize the message value
8     string sentence = this.RemoveDiacritics(message);
9     string action = this.GetActionFromSentence(sentence);
10    string actionValue =
11        this.ReadValueOfSelectedComponent(action);
12    string component = this.GetComponentFromSentence(sentence);
13    Type componentType = this.GetComponentType(component);
14    foreach (Component itemType in this.Components)
15    {
16        if (itemType.GetType() == componentType)
```

```
17      {
18          this.WriteValue(itemType, action,
19             itemType.GetType().GetProperty(actionValue));
20     }
21 }
```

Listing 23 – Piface Digital 2 - Traduction et recherche du composant

11.4 Classe de gestion des lumières

Cette classe est une partie maîtresse de l'application, car elle est le principe même de l'implémentassions d'un module de type PiFace Digital 2 pour commander du matériel électronique à forte tension telle que la lumière. Pour pouvoir commander ce type de matériel, il faut utiliser des relais électroniques tels que la figure qui suit, qui est la représentation d'un relais que l'on retrouve sur le module.

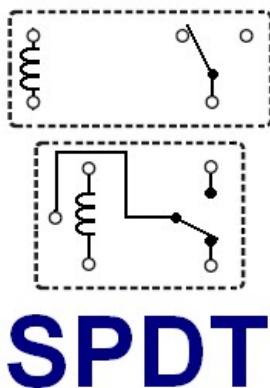


FIGURE 24 – PiFace Digital 2 - Type de relais utilisé

La classe *Light* permet de contrôler ce type relais à l'aide du procédé de recherche et d'écriture de propriété qui à été précédemment citée. Ce procédé va rechercher la propriété *IsOn* qui est contenue dans la classe *Light* et va la mettre à jour selon le rôle de l'action. L'écriture sur le PiFace Digital 2 ce fait directement dans les propriétés contenues dans *IsOn*.

```

1 #region Fields
2 #region Constant
3 // PiFace output
4 private const byte RELAIA = PiFaceDigital2.RelayA;
5 private const byte RELAIB = PiFaceDigital2.RelayB;
6
7 // PiFace State
8 private const byte OFF = MCP23S17.Off;
9 private const byte ON = MCP23S17.On;
10 #endregion
11
12 #region Variable
13 private bool _isOn = false;
14 private bool _isOnA = false;
```

```
15 private bool _isOnB = false;
16 #endregion
17 #endregion
18
19 #region Properties
20 public bool IsOn
21 {
22     get
23     {
24         return _isOn;
25     }
26
27     set
28     {
29         _isOn = value;
30         this.IsOnA = value;
31         this.IsOnB = value;
32     }
33 }
34
35 public bool IsOnA
36 {
37     get
38     {
39         return _isOnA;
40     }
41
42     set
43     {
44         _isOnA = value;
45         if (value)
46         {
47             // Turn ON the light
48             MCP23S17.WritePin(RELAIA, ON);
49         }
50         else
51         {
52             // Turn OFF the light
53             MCP23S17.WritePin(RELAIA, OFF);
54         }
55     }
56 }
57
58 public bool IsOnB
59 {
60     get
61     {
62         return _isOnB;
63     }

```

```
64      set
65      {
66          _isOnB = value;
67          if (value)
68          {
69              // Turn ON the light
70              MCP23S17.WritePin(RELAIB, ON);
71          }
72          else
73          {
74              // Turn OFF the light
75              MCP23S17.WritePin(RELAIB, OFF);
76          }
77      }
78  }
79 }
80 #endregion
```

Listing 24 – Piface Digital 2 - Classe de gestion des lumières

11.5 Classe de gestion des stores

La classe de gestion des stores est elle aussi une classe importante pour le projet. De la même façon que la classe de gestion des lumières, cette classe permet de contrôler les composants qui sont connectés dessus. En l'occurrence, cette classe permet le contrôle du store. Elle utilise un système de mémorisation pour savoir à quel niveau le store est, et ainsi de pouvoir repartir de cette valeur lorsqu'une nouvelle commande survient.

Avant toutes choses, pour piloter un moteur dans les deux sens, il faut utiliser un montage dis "pont en H". Ce montage est assez spécifique, car il a été conçu pour piloter les moteurs électriques dans les deux sens. Voici le fonctionnement d'un "pont en H". Ce montage demande à ce que les interrupteurs A et D soit connectés entre eux, de même pour les interrupteurs B et C afin de permettre de faire passer le courant dans les deux sens su moteur.

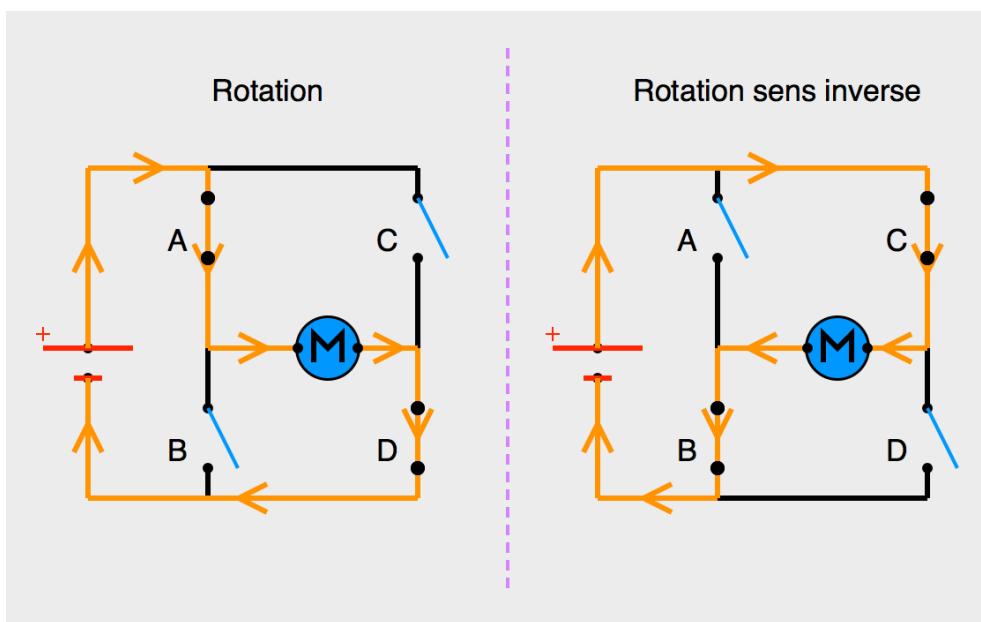


FIGURE 25 – PiFace Digital 2 - Fonctionnement du pont en H

Pour ce projet, j'ai d'abord fait un montage contenant quatre transistors NPN (interrupteur électronique), mais j'ai dû me confronter à un souci majeur car la tension de sortie du PiFace Digital 2 est de 3.3 Volts à 4 Volts et que pour commander le transistor que je disposais à sa base, il faut 5 Volts minimums. Du coup le montage était inutilisable. Mais la solution à ce problème est simple, il

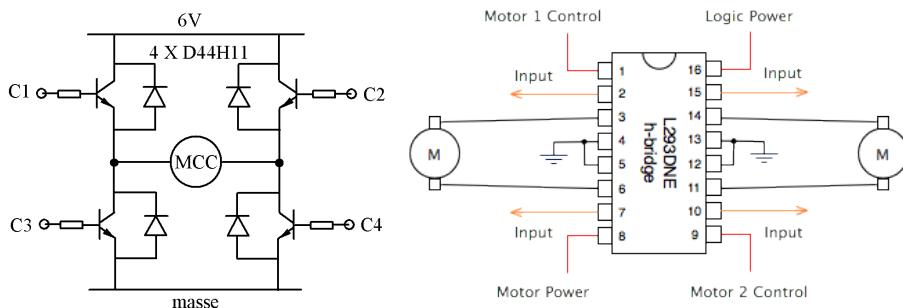


FIGURE 26 – PiFace Digital 2 - Montage transistor NPN et IC Pont en H

s'avère que je disposais par chance d'un circuit intégré de type L293DNE¹⁰, qui faisait le travail du "pont en H" que j'avais besoin pour faire fonctionner mes stores.

Le programme qui gère le moteur fonctionne de la même façon pour chaque montage. Le circuit intégré utilisé permet de contrôler un moteur allant jusqu'à 36 Volts. La classe de gestion des stores est modifiée de la même façon que la classe de gestion des lumières. Le procédé est que l'application va chercher les différentes propriétés et va mettre à jour les bonnes, toujours en fonction du rôle de l'action. Par la suite, l'écriture se fait dans les propriétés contenues dans la classe de gestion des stores.

```

1 #region Fields
2 #region Constant
3 // PiFace output
4 private const byte UP = PiFaceDigital2.LED4;
5 private const byte DOWN = PiFaceDigital2.LED3;
6
7 // PiFace State
8 private const byte OFF = MCP23S17.Off;
9 private const byte ON = MCP23S17.On;
10
11 // Max value for store (totally open)
12 private const int MAX_LEVEL = 200; // Time span total =
13 // 19seconds (raspberry latency)
14 // Min value for store (totally close)
15 private const int MIN_LEVEL = 0;
16 // Tick for timer
17 private const int TICKS = 10;
18 private const int TICK_SECOND = 1;
19 #endregion

```

10. Fiche technique du L293DNE - <http://www.ti.com/lit/ds/symlink/l293.pdf>

```
20
21 #region Variable
22 DispatcherTimer _dTimerUp = new DispatcherTimer();
23 DispatcherTimer _dTimerDown = new DispatcherTimer();
24
25 private bool _isUp = false;
26 private bool _isDown = false;
27 private bool _isOpen = false;
28 private bool _isClose = false;
29 private bool _isStop = false;
30
31 private int _counterStopped = 0;
32 #endregion
33 #endregion
34
35 #region Properties
36 public bool IsUp
37 {
38     get
39     {
40         return _isUp;
41     }
42
43     set
44     {
45         _isUp = value;
46
47         // Maximum level
48         if (value && this.CounterStopped < MAX_LEVEL)
49         {
50             this.SetLevel("IsUp");
51         }
52     }
53 }
54
55 public bool IsDown
56 {
57     get
58     {
59         return _isDown;
60     }
61
62     set
63     {
64         _isDown = value;
65
66         // Minimum level
67         if (value && this.CounterStopped > MIN_LEVEL)
68         {
```

```
69         this.SetLevel("IsDown");
70     }
71 }
72 }
73
74 public bool IsOpen
75 {
76     get
77     {
78         return _isOpen;
79     }
80
81     set
82     {
83         _isOpen = value;
84
85         if (value)
86         {
87             this.SetLevel("IsOpen");
88         }
89     }
90 }
91
92 public bool IsClose
93 {
94     get
95     {
96         return _isClose;
97     }
98
99     set
100    {
101        _isClose = value;
102
103        if (value)
104        {
105            this.SetLevel("IsClose");
106        }
107    }
108 }
109
110 public bool IsStop
111 {
112     get
113     {
114         return _isStop;
115     }
116
117     set
```

```
118     {
119         _isStop = value;
120
121         // Stop everything
122         if (value)
123         {
124             this._dTimerUp.Stop();
125             this._dTimerDown.Stop();
126             SetLevel("IsStop");
127             this.IsStop = false;
128         }
129     }
130 }
131
132 public int CounterStopped
133 {
134     get
135     {
136         return _counterStopped;
137     }
138
139     set
140     {
141         _counterStopped = value;
142
143         // Store manager
144         if (value == MAX_LEVEL)
145         {
146             this._dTimerUp.Stop();
147             SetLevel("IsStop");
148             _counterStopped = MAX_LEVEL;
149         }
150         else if (value == MIN_LEVEL)
151         {
152             this._dTimerDown.Stop();
153             SetLevel("IsStop");
154             _counterStopped = MIN_LEVEL;
155         }
156     }
157 }
158 #endregion
159
160 #region Constructor
161 public Store()
162 {
163     this._dTimerUp.Interval = new TimeSpan(TICK);
164     this._dTimerUp.Tick += _dTimerUp_Tick;
165
166     this._dTimerDown.Interval = new TimeSpan(TICK);
```

```
167     this._dTimerDown.Tick += _dTimerDown_Tick;
168 }
169
170 private void _dTimerUp_Tick(object sender, object e)
171 {
172     this.CounterStopped++;
173 }
174
175 private void _dTimerDown_Tick(object sender, object e)
176 {
177     this.CounterStopped--;
178 }
179 #endregion
180
181 #region Methods
182 /// <summary>
183 /// Set the level
184 /// </summary>
185 /// <param name="propertyName"></param>
186 private void SetLevel(string propertyName)
187 {
188     switch (propertyName)
189     {
190         case "IsUp":
191             this.IsDown = false;
192
193             MCP23S17.WritePin(DOWN, OFF);
194             MCP23S17.WritePin(UP, ON);
195
196             this.SetLevelUp();
197             break;
198         case "IsDown":
199             this.IsUp = false;
200
201             MCP23S17.WritePin(UP, OFF);
202             MCP23S17.WritePin(DOWN, ON);
203
204             this.SetLevelDown();
205             break;
206         case "IsOpen":
207             this.IsFalse = false;
208
209             this.SetLevel("IsUp")
210             await Task.Delay(TimeSpan.FromSeconds(TICK_SECOND))
211             this.SetLevel("IsStop")
212             break;
213         case "IsClose":
214             this.IsTrue = false;
215 }
```

```

216         this.SetLevel("IsDown")
217         await Task.Delay(TimeSpan.FromSeconds(TICK_SECOND))
218         this.SetLevel("IsStop")
219         break;
220     case "IsStop":
221         this.IsUp = false;
222         this.IsDown = false;
223         this.isOpen = false;
224         this.isClose = false;
225
226         MCP23S17.WritePin(UP, OFF);
227         MCP23S17.WritePin(DOWN, OFF);
228         break;
229     }
230 }
231
232 /// <summary>
233 /// Set upper the level of the store
234 /// </summary>
235 private void SetLevelUp()
236 {
237     this._dTimerDown.Stop();
238     this._dTimerUp.Start();
239 }
240
241 /// <summary>
242 /// Set downer the level of the store
243 /// </summary>
244 private void SetLevelDown()
245 {
246     this._dTimerUp.Stop();
247     this._dTimerDown.Start();
248 }
249 #endregion

```

Listing 25 – PiFace Digital 2 - Classe de gestion des stores

11.6 Améliorations possible de la gestion des stores

Il est possible d'améliorer le programme afin de gérer chaque composant indépendamment des autres. Étant donné que la plaque peut avoir un total de deux lumières contrôler séparément (je ne compte pas les montages parallèles qui peuvent être faits sur chacun des relais, je les compte comme une lumière par relais dans tous les cas). Ainsi que le contrôle de 3 stores possibles, vu que le moteur qui contrôle les stores n'a besoin que de deux informations pour contrôler le moteur.

12 Problèmes rencontré et résolution

12.1 Composant reçu différent

Pour ce projet, il a fallu commander deux composants ; un microphone MAX9-814 et un amplificateur audio classe D MAX98357A. Mais une fois les composants reçus, ils étaient différents des originaux, surtout l'amplificateur qui contient directement un gain réglable intégré au circuit. Donc il a fallu modifier les schémas électroniques, pour pouvoir rendre le matériel utilisable.

12.2 Carte électronique pour ajouter un microphone

Au cours de ce projet, il était question que je développe une plaque électronique utilisant un microphone, et qui soit compatible avec le Raspberry Pi 3 modèle B. Mais, je n'ai pas pensé à la réelle difficulté qui est le traitement des données que l'on reçoit à la sortie du microphone. Cette difficulté vient surtout du principe de faire une première conversion d'un signal analogique de 3.3 Volts, en un signal digital à l'aide d'un circuit électronique, qui convertit ce type de signal. De plus, la tension utilisée est trop basse pour pouvoir traiter correctement le signal et avoir de la précision sur l'échantillonnage. Pour cela, il faut augmenter le signal sinusoïdal que l'on obtient en sortie du microphone. La solution est d'ajouter un amplificateur opérationnel non inverseur juste après la sortie du microphone pour augmenter son signal de sortie, et ainsi avoir une meilleure précision sur le signal qui en sort.

12.3 Échantillonnage du signal du convertisseur

Le problème avec Windows IoT, est le fait que les bibliothèques varient entre une application de type console, à une application universelle de Visual Studio. Cette différence pose problème, car il y a peu d'informations utiles sur les applications UWP (Universal Windows Platform). On trouve des informations de base pour développer son application, mais pour faire de l'échantillonnage de données c'est beaucoup plus complexe, en plus du fait que peu de personnes n'est développées le sujet. En faisant diverses recherches, j'ai pu constater que beaucoup de forums parlent du fait d'utiliser un microphone qui se branche en USB, dans les ports alloués à cet effet sur le Raspberry Pi. Par la suite, le développement de l'application est plus simple grâce à cette solution. Malgré tout, je n'ai pas pu disposer d'un microphone USB pour développer la reconnaissance vocale.

12.4 Langue de synthèse par défaut de Windows Iot

Au cours de ce projet, il est demandé d'utiliser des haut-parleurs afin de développer une synthèse vocale, et d'avoir un résultat audible. Le souci vient de Windows Iot lui-même, car la langue par défaut que la synthèse vocale utilise, est de l'anglais américain. La première solution est de se connecter grâce à PowerShell sur le Raspberry Pi, ainsi que sur le partage réseau distant, pour pouvoir transférer directement les fichiers nécessaires à la synthèse vocale. Cette solution fonctionne, mais le programme n'arrive pas à générer la synthèse avec un synthétiseur français, et le programme "crash" à la moindre occasion. La seconde solution qui est la bonne mais qui n'est pas encore disponible, est le fait d'attendre la prochaine mise à jour de Windows Iot, une mise à jour qui va rajouter plusieurs langues différentes de synthèse vocale, telles que le français, l'allemand, l'espagnol, le portugais et plus encore. Cette solution-là a été trouvée sur un forum officiel de Microsoft, sur l'utilisation de la synthèse vocale allemande, pour un projet utilisant la même technologie que ce projet.

12.5 Valeurs faussées du module Sense HAT

Le problème avec le Sense HAT, est que tous les capteurs sont directement intégrés au circuit électronique. Ce point est pratique, cela évite d'avoir des capteurs qui partent dans tous les sens, mais ces capteurs, surtout celui de la température, il ne prend pas en compte la chaleur que le Raspberry dégage. Ce qui en résulte, est des chaleurs dépassant de plus de 10 degrés Celsius de la température ambiante. Malgré tout, je n'arrive pas à déterminer si tous les capteurs subissent le même problème, tel que le capteur d'humidité qui peut potentiellement en être affecté.

12.6 Communication avec le Sense HAT

Ce projet demande à ce que le Sense HAT renvoie les données qu'il mesure quand il l'est demandé. La communication se fait bien, mais après avoir transmis la même commande plusieurs fois, la communication avec le Sense HAT saute alors que le programme de communication est le même pour tous. Ce problème est plutôt dérangeant, mais en prêtant attention aux informations du serveur, on remarque que la connexion est toujours présente, donc que le client ne s'est pas déconnecté du serveur, et on remarque que rien ne se passe du côté du Sense HAT non plus.

13 Protocole de tests

13.1 Procédé de communication

| N° | Tests | État |
|----|---|------|
| 1 | Initialisation du serveur (Serveur) | OK |
| 2 | Création du serveur et démarrage de ce dernier (Serveur) | OK |
| 3 | Initialisation du client comportant le microphone (Client - Microphone) | OK |
| 4 | Connexion de ce client sur le serveur (Client - Microphone) | OK |
| 5 | Inscription de la chaîne de caractères sur la trame (Client - Microphone) | OK |
| 6 | Lecture d'un nouveau client sur le serveur (Serveur) | OK |
| 7 | Création d'un objet TCPClient pour le nouveau client connecté (Serveur) | OK |
| 8 | Lecture de la chaîne de caractères du client (Serveur) | OK |
| 9 | Création d'un objet RaspberryClient à l'aide des données de la chaîne de caractères lu par le serveur (Serveur) | OK |
| 10 | Le Client Microphone se met en mode attente d'informations venant du serveur (Client - Microphone) | OK |
| 11 | Le serveur parcourt tous les clients si ils sont déconnectés pour supprimer de la mémoire (Serveur) | OK |
| 12 | Le serveur parcourt tous les clients connectés pour un nouveau message (Serveur) | OK |
| 13 | Initialisation du client comportant le Sense HAT (Client - Sense HAT) | OK |
| 14 | Connexion de ce dernier sur le serveur (Client - Sense HAT) | OK |
| 15 | Inscription de la chaîne de caractères sur la trame (Client - Sense HAT) | OK |
| 16 | Lecture d'un nouveau client sur le serveur (Serveur) | OK |
| 17 | Création d'un objet TCPClient pour le nouveau client connecté (Serveur) | OK |
| 18 | Lecture de la chaîne de caractères du client (Serveur) | OK |
| 19 | Création d'un objet RaspberryClient à l'aide des données de la chaîne de caractères lu par le serveur (Serveur) | OK |
| 20 | Le Client Sense HAT se met en mode attente d'informations venant du serveur (Client - Sense HAT) | OK |
| 21 | Initialisation du client comportant le PiFace Digital 2 (Client - PiFace Digital 2) | OK |

| | | |
|----|---|----|
| 22 | Connexion de ce dernier sur le serveur (Client - PiFace Digital 2) | OK |
| 23 | Inscription de la chaîne de caractères sur la trame (Client - PiFace Digital 2) | OK |
| 24 | Lecture d'un nouveau client sur le serveur (Serveur) | OK |
| 25 | Création d'un objet TCPClient pour le nouveau client connecté (Serveur) | OK |
| 26 | Lecture de la chaîne de caractères du client (Serveur) | OK |
| 27 | Création d'un objet RaspberryClient à l'aide des données de la chaîne de caractères lu par le serveur (Serveur) | OK |
| 28 | Le Client PiFace Digital 2 se met en mode attente d'informations venant du serveur (Client - PiFace Digital 2) | OK |
| 29 | Envoi d'une commande au serveur (Client - Microphone), Message : Allume la lumière du salon | OK |
| 30 | Nouveau message venant d'un client (Serveur), Message : Allume la lumière du salon | OK |
| 31 | Recherche du bon client grâce au message (Serveur) | OK |
| 32 | Recherche tous les Clients au "Salon" (Serveur) | OK |
| 33 | Recherche tous les Clients au "Salon" qui ont "Lumière" (Serveur) | OK |
| 34 | Écriture sur la trame des clients trouvés (Serveur) | OK |
| 35 | Détection de nouveau message sur la trame (Client - PiFace Digital 2), Message : Allume la lumière du salon | OK |
| 36 | Décodage du message (Client - PiFace Digital 2) | OK |
| 37 | Défragmentation du message en plusieurs informations (Client - PiFace Digital 2) | OK |
| 38 | Activation du matériel de lumière (Client - PiFace Digital 2) | OK |
| 39 | Le Client PiFace Digital 2 se met en mode attente d'informations venant du serveur (Client - PiFace Digital 2) | OK |
| 40 | Envoi d'une commande au serveur (Client - Microphone), Message : Quel est la température du salon | OK |
| 41 | Nouveau message venant d'un client (Serveur), Message : Quel est la température du salon | OK |
| 42 | Recherche du bon client grâce au message (Serveur) | OK |
| 43 | Recherche tous les Clients au "Salon" (Serveur) | OK |
| 44 | Recherche tous les Clients au "Salon" qui ont "Température" (Serveur) | OK |
| 45 | Écriture sur la trame des clients trouvés (Serveur) | OK |

| | | |
|----|---|----|
| 46 | Détection de nouveau message sur la trame (Client - Sense HAT) Message : Quel est la température du salon | OK |
| 47 | Mise à jour des valeurs des capteurs (Client - Sense HAT) | OK |
| 48 | Encodage des valeurs en une chaîne de caractères (Client - Sense HAT) | OK |
| 49 | Écriture de la chaîne de caractères dans la trame (Client - Sense HAT) | OK |
| 50 | Nouveau message venant d'un client (Serveur) Message : TEMP=x ;HUMI=y ;PRES=z | OK |
| 51 | Écriture du message sur le client qui à fait la demande (Serveur) | OK |
| 52 | Détection de nouveau message sur la trame (Client - Microphone) Message : TEMP=x ;HUMI=y ;PRES=z | OK |
| 53 | Décodage du message (Client - Microphone) | OK |
| 54 | Synthétisation du message (Client - Microphone) | OK |
| 55 | Déconnexion du client Sense HAT (Client - Sense HAT) | OK |
| 56 | Recherche de client inexistant (Serveur) | OK |
| 57 | Suppression du client déconnecté de la liste des clients connecté (Serveur) | OK |

Il s'agit des tests fonctionnels du déroulement de base du projet, il manque les tests avec l'application, mais les tests sont les mêmes qu'avec le Client - Microphone. Les erreurs qui peuvent survenir sont les suivantes ;

- Connexion d'un client avec un mauvais format (Serveur), le serveur ne l'accepte pas et envoie retourne un message d'erreur sur la console.
- Nouvel connexion d'un client déjà existant (Serveur), le serveur à déjà le client en mémoire
- Envoi de mauvais message venant du client (Client - Microphone), le serveur renvoi un message d'erreur au client
- Envois de message (Client - Microphone), Le serveur ne connaît pas de client à informer (Serveur), le serveur envoie message impossible de trouver.

14 Conclusion

Ce projet est une immense source d'information, de connaissances, d'entraînement ainsi que de recherches. Le travail fourni a été d'une grande envergure, étant donné du nombre de programmes qu'il a fallu développer. Ces programmes m'ont permis d'augmenter mes connaissances en informatique dans le langage C#.

La domotique est un sujet que je n'ai jamais traité auparavant, mais il s'agit d'un thème qui m'a beaucoup intéressé tout au long ma formation d'électronicien et d'informaticien. Du coup, c'était l'occasion rêvée de pouvoir faire un travail de diplôme qui mélange à la fois l'électronique et l'informatique, afin de développer un projet de domotique.

RaspiHome m'a permis de plus me familiariser avec Visual Studio 2015. Ce projet m'a aussi permis de découvrir les applications universelles que l'on peut développer avec Visual Studio 2015. ce type d'application m'a fait découvrir une toute autre façon de travailler étant donné que les bibliothèques à utiliser sont différentes d'une application basique. Et m'a fait découvrir le Raspberry Pi, qui avec lequel, il peut y avoir plusieurs types d'utilisations tels que l'implémentassions de module électronique dessus.

Pour conclure, ce projet ma permis de développer plusieurs compétences à des fins professionnels tels que le développement d'applications dynamique, l'utilisation des différentes technologies que j'ai dû apprendre ainsi que des différents modules qui ont été utilisés tout au long du projet. Grâce à ce projet, je suis maintenant apte à utiliser des Raspberry Pi afin de faire de la domotique, et faire des applications qui communiquent entre elles.

15 Table des figures

| | | |
|----|---|----|
| 1 | Google Home - Image de présentation | 10 |
| 2 | Amazon Echo - Image de présentation | 11 |
| 3 | Amazon Echo - Utilitaire d'Amazon Echo | 12 |
| 4 | Homekit - Image de présentation | 13 |
| 5 | Gatebox - Image de présentation | 14 |
| 6 | Raspberry Pi 3 Modèle B - Plaque électronique | 16 |
| 7 | Sense HAT - Plaque électronique | 17 |
| 8 | PiFace Digital 2 - Plaque électronique | 18 |
| 9 | MAX9814 - Plaque électronique | 19 |
| 10 | Sound sensor - Plaque électronique | 19 |
| 11 | MAX98357A - Plaque électronique | 20 |
| 12 | PAM8302A - Plaque électronique | 20 |
| 13 | Structure réseau du projet | 23 |
| 14 | Interface graphique schématisé de l'application pour tablette . . | 25 |
| 15 | Serveur - Représentation schématisé du réseau | 35 |
| 16 | Circuit imprimé - Premier schéma électronique | 41 |
| 17 | Interface graphique - Gestion maison | 50 |
| 18 | Interface graphique - Gestion salon | 51 |
| 19 | Interface graphique - Barre d'outils de navigation | 51 |
| 20 | Interface graphique - Barre d'outils de choix de localisation . . | 52 |
| 21 | Interface graphique - Bouton de contrôle de la lumière | 52 |
| 22 | Interface graphique - Boutons de contrôle du store | 52 |
| 23 | Interface graphique - Affichage valeurs mesurées | 53 |
| 24 | PiFace Digital 2 - Type de relais utilisé | 66 |
| 25 | PiFace Digital 2 - Fonctionnement du pont en H | 69 |
| 26 | PiFace Digital 2 - Montage transistor NPN et IC Pont en H . . . | 70 |

16 Table des listings

| | | |
|----|---|----|
| 1 | Serveur - Premier état d'écoute de la trame | 26 |
| 2 | Serveur - Écoute constante de la trame | 27 |
| 3 | Serveur - Crédit de clients avec leurs propres informations | 28 |
| 4 | Serveur - Recherche des bons clients | 30 |
| 5 | Serveur - Écriture dans la trame réseau du client | 33 |
| 6 | Serveur - Déconnexion du client | 34 |
| 7 | Constructeur de l'application de contrôle du PiFace Digital 2 | 37 |
| 8 | Constructeur de l'application de contrôle du Sense HAT | 37 |
| 9 | Constructeur de l'application de reconnaissance et de synthèse vocale | 37 |
| 10 | Constructeur de l'application de la tablette Windows | 38 |
| 11 | Client - Lecture de la trame du client | 38 |
| 12 | Client - Écriture sur trame du client | 39 |
| 13 | Reconnaissance vocale - Grammaire personnalisée de la reconnaissance vocale utilisée pour RaspiHome | 43 |
| 14 | Synthèse vocale - Initialisation du synthétiseur vocal et activation de la synthèse | 46 |
| 15 | Interface graphique - Écriture dans la mémoire locale | 53 |
| 16 | Interface graphique - Lecture dans la mémoire locale | 54 |
| 17 | Sense HAT - Ajout de la bibliothèque du Sense HAT | 56 |
| 18 | Sense HAT - Initialisation du Sense HAT et fonctionnement interne | 57 |
| 19 | Sense HAT - Format de l'envoi des valeurs des capteurs | 59 |
| 20 | Piface Digital 2 - Décomposition de la phrase | 60 |
| 21 | Piface Digital 2 - Recherche de la propriété à modifier | 62 |
| 22 | Piface Digital 2 - Traduction et recherche du composant | 63 |
| 23 | Piface Digital 2 - Traduction et recherche du composant | 64 |
| 24 | Piface Digital 2 - Classe de gestion des lumières | 66 |
| 25 | Piface Digital 2 - Classe de gestion des stores | 70 |

17 Références

17.1 Références d'ordre général

Outils Microsoft : <https://msdn.microsoft.com/fr-fr/dn308572.aspx>

Aide StackOverflow : <https://stackoverflow.com/>

17.2 Références STT et TTS

Création d'une grammaire personnel : <https://msdn.microsoft.com/en-us/library/jj127917.aspx>

Projet générale - reconnaissance vocale : <https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/SpeechRecognitionAndSynthesis>

Synthèse vocale projet : <https://www.hackster.io/kvvarma/rpispeechsynthesis-51f269>

Projet générale - synthèse vocale : <https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/SpeechRecognitionAndSynthesis>

17.3 Références Sense HAT

Informations sur le composant : <https://www.raspberrypi.org/products/sense-hat/>

Implémentations de la bibliothèque : <https://www.nuget.org/packages/Emmellsoft.IoT.RPi.SenseHat/>

Utilisation de Sense HAT : <https://github.com/emmellsoft/RPi.SenseHat>

17.4 Références PiFace Digital 2

Caractéristiques technique et achat du PiFace Digital 2 : <http://fr.farnell.com/piface/piface-digital-2/carte-d-exten-i-o-pour-raspberry/dp/2434230>

Utilisation du PiFace Digital 2 : <https://www.element14.com/community/community/raspberry-pi/blog/2015/08/03/piface-digital-ii-on-a-pi2-windows-10-iot-and-a-cool-demo>

17.5 Références Tablette Windows

Stockage local de données : <https://docs.microsoft.com/en-us/windows/uwp/app-settings/store-and-retrieve-app-data>

18 Annexe

- Code source
- Journal de bord
- Manuel utilisateur