Thingz code documentation

Paul Bivic, Pierre Boulc'h, Alexis Busseneau,
Lucie Labadie, Mathieu Lochet, Adrien Thébault

9 janvier 2018

Table des matières

Serveur

1.1 Fonctionnement

Le serveur permet de récupérer les données nécessaires à la configuration de l'application et du Thingz. Il envoie une requête au serveur contenant sa configuration en JSON et reçoit un message au format JSON contenant les données désirées. Ce message contient les informations concernant l'emploi du temps, le trajet et l'heure actuelle. Un exemple de message est présent dans la partie 1.2.

1.2 Messages échangés avec l'application

Exemple de message échangé :

{"agenda":
{"beginning":1513584000,"end":1513587600,"duration":3600,"summary":"DS Elaboration et Caractéris. Nanostruct. 0026 Dispositifs","location":"125 (V)","groups":["S9-SGM"],"details":"LE CORRE Alain","teacher":"Elaboration et Caracterisation de Nanostructures e","current":false,"ended":false},
"weather":
{"weather":"Clouds","temp":9},
"travel_time":9,"timestamp":1513515889}

Application

2.1 Fonctionnement

L'application permet à l'utilisateur de configurer le Thingz en entrant ses préférences : temps de préparation, caractéristique d'emploi du temps et adresse. Un messager est ensuite envoyé au serveur pour récupérer le message montré dans la section 1.2. Le résultat est ensuite transformé comme montré dans la section 2.2 pour envoyer la configuration au Thingz.

2.2 Messages échangés avec le Thingz

Exemple de message échangé entre l'application et le Thingz :

```
 AGENDA data;data;data
WEATHER data;data
TRAVEL_TIME data
TIMESTAMP data
```

Thingz

3.1 Fonctionnement

Le Thingz commence par se connecter à l'application via Bluetooth. Une fois connecté, il est prêt à recevoir la configuration utilisateur qui suit le format montré à la section 2.2. Les données sont ensuite parsées pour initialiser les varibales du Thingz. L'appuie sur un bouton provoque l'attente de réception d'une nouvelle configuration. L'heure, la météoet le premier cours de la journée sont affichés sur l'écran. La prise commandée reliée à la cafetière se déclenche suffisamment à l'avance pour que le café soit près au réveil. Lors que le réveil sonne une fonction snooze est disponible en touchant la brique tactile. Lorsque la lumière est allumée, le réveil s'arrête tout seul.

Liste des briques utilisées :
— 3 boutons ;
— Brique Bluetooth ;
— Brique tactile ;
— Ecran LCD ;
— Buzzer ;
— Capteur de lumière ;
— 3 LED ;
— Potentiomètre ;
— Brique relai (prise commandée) ;
— Brique météo ;
— Détecteur de mouvement.

3.2 Utilisation des briques

3.2.1 LED

Ajouter `#include "Led.h"`
Pour créer un objet `Led` : `Led led;`

`led.allume()`
Allume la LED. La LED reste allumée tant que la fonction pour éteindre n'est pas appelée.

`led.eteint()`
Allume la LED. La LED reste éteinte tant que la fonction pour allumer n'est pas appelée.

`led.inverse()`
Change l'état de la LED vers son inverse. Si la LED est éteinte alors elle s'allume, l'inverse sinon.

`led.estAllume()`
Renvoie `true` si la LED est allumée, `false` sinon.

3.2.2 Écran

Ajouter `#include "Screen.h"`
Pour créer un objet `Screen` : `Screen screen;`

`screen.printMsg(string msg, int lineNumber)`
Affiche le texte `msg` à la ligne `lineNumber` % 6 sur l'écran. Une ligne peut contenir jusqu'à 13 caractères. Seulement 6 lignes sont disponibles à l'écran.

`screen.clear()`
Efface tout ce qui est affiché sur l'écran.

`screen.clearLine(int lineNumber)`
Efface ce qui est affiché à la ligne `lineNumber` % 6 sur l'écran.

`screen.switchOn()`
Allume le rétro-éclairage de l'écran.

`screen.switchOff()`
Éteint le rétro-éclairage de l'écran.

`screen.setContrast(int val, bool mem)`
Modifie le contraste de l'écran à la valeur `val` et mémorise cette valeur si `mem` est à `true`, ne mémorise pas sinon. La mémorisation reste lorsqu'un nouveau programme est téléversé.

3.2.3 Buzzer

Ajouter `#include "Son.h"`
Pour créer un objet `Son` : `Son buzzer;`

`buzzer.sonne(int freq)`
Fait sonner le buzzer à la fréquence `freq` passée en paramètre (en Hz).

`buzzer.arreteDeSonner()`
Stop le buzzer.

`buzzer.tone(int freq, int duree)`
Fait sonner le buzzer à la fréquence `freq` pendant la durée `duree`.

3.2.4 Bluetooth

Ajouter `#include "Bluetooth.h"`
Pour créer un objet `Bluetooth` : `Bluetooth bluetooth;`

`bluetooth.connect(string name, string pwd, bool)`
Se connecte au bluetooth de nom `name` avec le mot de passe `pwd`. Le boolean sert à indiquer si le Thingz se connecte à un autre Thingz.

`bluetooth.disconnect()`
Se déconnecte du bluetooth.

`bluetooth.send(string data)`
Envoie `data` via la connection bluetooth établie.

`bluetooth.acceptConnection(string name)`
Accepte la connexion du bluetooth `name`. Attente active jusqu'à connexion.

`bluetooth.receive()`
Récupère les messages en attente de réception.

`bluetooth.dataAvailable()`
Renvoie `true` si le module a des données non-reçues, `false` sinon.

`bluetooth.localName()`
Renvoie le nom de la brique bluetooth.

`bluetooth.localPassword()`
Renvoie le mot de passe de la brique bluetooth.

`bluetooth.getModuleName()`
Renvoie le nom du module.

`bluetooth.setPassword(string newpwd)`
Change le mot de passe du bluetooth à `newpwd`.

`bluetooth.changeModuleId(string newID)`
Change l'identifiant du module bluetooth.

`bluetooth.changeModuleName(string newname)`
Remplace le nom du bluetooth à `newname`.

3.2.5 Prise commandée

Ajouter `#include "Relai.h"`
Pour créer un objet `Relai` : `Relai relai;`

`relai.allumer(int prise, int groupe)`
Allume la prise `prise` dans le groupe `groupe` (1 par défaut).

`relai.eteindre(int prise, int groupe)`
Eteint la prise `prise` dans le groupe `groupe` (1 par défaut).

3.2.6 Potentiomètre

Ajouter `#include "Potentiometer.h"`
Pour créer un objet `Potentiomètre` : `Potentiometer potentiometer;`

`potentiometer.etat()`
Renvoie la valeur du potentiomètre (valeur entre 0 et 100).

3.2.7 Brique tactile : MakeyMakey

Ajouter `#include "MakeyMakey.h"`
Pour créer un objet `Makey` : `MakeyMakey makey;`

`makey.touched()`
Renvoie `true` si le contact avec l'objet est maintenu pendant l'appel de cette fonction, `false` sinon.

`makey.clicked()`
Renvoie `true` si l'objet a été cliqué, `false` sinon.

3.2.8 Bouton

Ajouter `#include "Bouton.h"`
Pour créer un objet `Bouton` : `Bouton button;`

`button.estTenuAppuye()`
Renvoie `true` tant que le bouton est maintenu appuyé, `false` sinon.

`button.aEteAppuye()`
Renvoie `true` si le bouton a été cliqué ou maintenu appuyé, `false` sinon.

3.2.9 Capteur de luminosité

Ajouter `#include "Luminosite.h"`
Pour créer un objet `Luminosite` : `Luminosite luminosite;`

`luminosite.etat()`
Renvoie la valeur de la luminosité captée (valeur entre 0 et 100).

3.2.10 Récepteur infrarouge

Ajouter `#include "Infrarouge.h"`
Pour créer un objet `Infrarouge` : `Infrarouge infrarouge;`

`infrarouge.detecteUnSignal()`
Renvoie le signal détecté si un signal est reçu (permettant d'identifier différents bouton de la télécommande), `false` sinon.

3.2.11 Brique météo

Ajouter `#include "Temperature.h"`
Pour créer un objet `Temperature` : `Temperature meteo;`

`meteo.temperature()`
Renvoie la température mesurée (valeur entre 0 et 50°C).

`meteo.humidite()`
Renvoie le taux d'humidité mesuré (valeur entre 20 et 95%).

3.2.12 Détecteur de mouvement

Ajouter `#include "MotionSensor.h"`
Pour créer un objet `MotionSensor` : `MotionSensor motionSensor;`

`motionSensor.detectsMotion()`
Renvoie `true` si une suite de mouvement est détectée, `false` sinon.

`motionSensor.detectsStartOfMotion()`
Renvoie `true` si un mouvement est détecté, `false` sinon/ensuite.

3.3 Wifi

Premièrement le module wifi nécessite de recevoir 3,3V en tension et en utilisation maximale 500mA d'intensité du courant. Une carte Arduino classique fournit 5V de tension et 50mA d'intensité. Pour relier le module et la carte, la tension doit être adapté et l'alimentation doit pouvoir fournir jusqu'à 500mA d'intensité. Pour commencer nous avons adapté les branches Rx et Tx (qui servent à envoyer les commandes au micro-contrôleur du module). Comme illustré sur le schéma 3.1, l'adaptation a été réalisée à partir d'un pont diviseur de tension avec des résistances en parallèles. Pour Vcc (l'alimentation du module) nous avons utilisé un simple variateur.

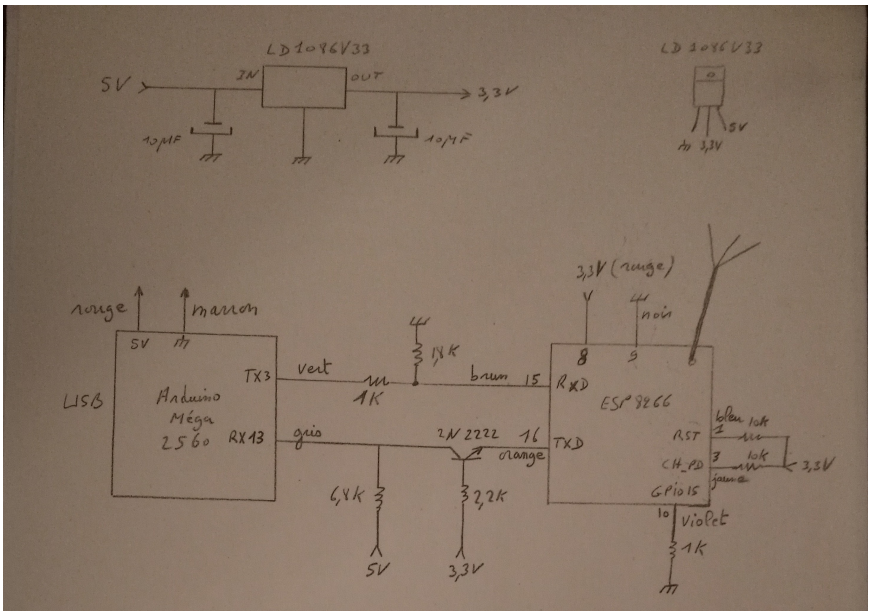

FIGURE 3.1 – Schéma électrique, liaison carte Arduino-module

Obtenir une intensité du courant de 500mA est plus compliqué. Pour commencer l'alimentation du Thingz correspond au courant délivré par le port USB de l'ordinateur. Un port USB peut délivrer un maximum de 500mA, ce qui nous convient. Cependant depuis une pin hacker branchée sur le Thingz il nous est impossible d'obtenir plus de 50mA. En effet le fonctionnement même du Thingz est de rendre chacun des pins de la carte universel dans le sens identique à tous les autres. Cet équilibre est fait par un nivellement par le bas, donc le pin ayant le moins de capacités représente le pin universel que deviennent tous les autres. Pour fournir 500mA à notre module il faut donc aller chercher l'intensité du courant à sa source. Pour cela 3 solutions s'offrent à nous :

— Ouvrir le boîtier du Thingz et aller chercher le courant à l'entrée de l'alimentation du boîtier ;
— Dénuder le câble USB pour récupérer le courant directement à la sortie du port USB ;
— Utiliser une alimentation externe chargée d'alimenter le module wifi.

Ouvrir le boîtier était une solution peu envisageable pour nous. L'alimentation externe du module demande des moyens matériels mais cette solution reste envisageable. Dénuder le fil du câble USB est aussi une solution possible.

Pour l'aspect un peu plus technique de programmation et utilisation du module wifi, nous nous heurtons une nouvelle fois au nivellement par le bas du Thingz. Tous les pins n'étant pas capable d'assurer une communication

Rx et Tx, aucun n'en est capable. Pour contourner ce problème nous avons contacté l'équipe responsable du développement du Thingz. Ils nous ont conseillé d'émuler un port série (capable de communiquer en Rx et Tx) afin d'envoyer nos commandes au module. La principe est de créer un port série virtuel sur un pin du Thingz et de le forcer à communiquer en Tx et Rx. Cependant il faut tester cette technique sur plusieurs pins pour émuler sur un pin ayant à l'origine les capacités de communiquer en Rx et Tx. Les membres de l'équipe du Thingz, pour nous aider dans notre tâche, nous ont fourni des sources implémentant déjà un émulateur de port série. La technologie et le développement derrière le Thingz étant la propriété de l'entreprise Thingz, nous ne pouvons pas transmettre les sources fournies par l'équipe.

Debogage

4.1 Installation des outils

Deux outils sont recommandés pour programmer sur Arduino : `Arduino IDE` ou `Slober` (plugin Chrome).

Vous pouvez les utiliser pour afficher les messages console du Thingz ET/OU pour téléverser du code. Pour téléverser du code, il faut importer la librairie Arduino du Thingz (fourni dans le git) dans un projet Arduino Mega 2560, afin d'avoir l'ensemble des sources lors du téléversement.

4.2 Affichage dans la console

Il faut ajouter dans la fonction `setup()`, `Serial.begin(9600);` afin d'autoriser les affichages dans la console. Pour afficher un message, il suffit d'utiliser : `Serial.print();` ou `Serial.println();`.

Si le Thingz a démarré et que vous ouvrez la console, vous devriez voir les messages s'afficher.