DEPARTEMENT INFORMATIQUE - IUT 2 GRENOBLE

Année Universitaire 2023-2024

MEMOIRE DE STAGE

STAGE DANS UN BUREAU D'ÉTUDES EN SYSTÈME EMBARQUÉ

Elyxoft

Présenté par

Mathieu Camus

Jury

IUT : Mme. Dugdale

IUT : M. Chevallet

Société : M. Marion

Déclaration de respect des droits d'auteurs

Par la présente, je déclare être le seul auteur de ce rapport et assure qu'aucune autre ressource que celles indiquées n'ont été utilisées pour la réalisation de ce travail. Tout emprunt (citation ou référence) littéral ou non à des documents publiés ou inédits est référencé comme tel.

Je suis informé qu'en cas de flagrant délit de fraude, les sanctions prévues dans le règlement des études en cas de fraude aux examens par application du décret 92-657 du 13 juillet 1992 peuvent s'appliquer. Elles seront décidées par la commission disciplinaire de l'UGA.

A, Charnècles Le, 07 juin 2024

Signature 

Je souhaite exprimer ma gratitude envers **Sébastien Marion**, mon maître de stage à Elyxoft, pour m'avoir offert l'opportunité d'intégrer l'entreprise et pour sa confiance tout au long de mon stage.

Un grand merci également à **mes parents** pour avoir relu mon rapport et pour m'avoir prêté leur voiture, ce qui m'a permis de me rendre sur le lieu de mon stage.

Je tiens à remercier chaleureusement **Julie Dugdale** pour ses précieux retours sur la première version de mon rapport.

Enfin, je suis reconnaissant envers toute l'**équipe enseignante de l'IUT2 de Grenoble** pour m'avoir transmis les bases essentielles en programmation, administration systèmes et rédaction.

Résumé long

Mon stage de onze semaines s'est déroulé chez Elyxoft, une petite entreprise spécialisée dans la conception logicielle et la programmation embarquée. Fondée en 2020 par Sébastien Marion, Elyxoft se concentre sur les objets communicants et la domotique, en particulier les protocoles de communication radio à faible débit comme Zigbee* et Matter*. La société, une SAS au capital social de 5000€, ne compte qu'un employé : le président de l'entreprise, qui a également été mon maître de stage. J'ai travaillé dans un environnement bienveillant et collaboratif, partagé avec un autre stagiaire, Fernando Cervantes.

Le stage a commencé par un projet annexe visant à me familiariser avec les outils de développement et les technologies que j'utiliserais ensuite. Ce projet d'une semaine portait sur la création d'une touillette connectée affichant la température du café pour éviter les brûlures, utilisant le Bluetooth Low Energy (BLE*) et le framework* Qt* en C++ pour développer l'application associée. Mon travail principal s'est ensuite concentré sur le projet ElyTicBridge, une passerelle protocolaire permettant de récupérer les informations de télé-information client (TIC*) des compteurs électriques Linky et de les transmettre via divers protocoles comme MQTT*, Modbus*, Zigbee, ou Matter.

Pour le projet ElyTicBridge, j'ai utilisé les langages C et C++, le microcontrôleur ESP32-C6 d'Espressif Systems et la bibliothèque ESP-IDF* pour la programmation embarquée. Le framework Qt a été utilisé pour le développement d'applications avec des interfaces utilisateur graphiques.

Pendant mon stage, j'ai commencé par restructurer l'architecture du code afin de la rendre plus modulaire et réutilisable. Cette restructuration a permis de stocker chaque module dans des référentiels Git distincts, facilitant ainsi leur gestion et utilisation ultérieure.

Ensuite, j'ai travaillé sur le module BLE. Celui-ci permet de configurer ElyTicBridge, notamment en configurant les identifiants WiFi afin de connecter le produit au réseau. Mon intervention a consisté à ajouter des paramètres de configuration pour les modules qui n'en possédaient pas encore, comme les modules Ethernet ou MQTT, rendant ainsi le module BLE plus complet et fonctionnel.

Par ailleurs, j'ai développé une application utilisant le framework Qt qui exploite le module BLE afin de configurer ElyTicBridge.

Ce stage m'a permis de renforcer mes compétences en C et C++, et de découvrir de nouvelles technologies comme Qt et la programmation embarquée. J'ai appris à travailler avec des ESP32 et à gérer les contraintes spécifiques de la programmation pour des systèmes embarqués. L'utilisation de divers protocoles de communication (BLE, MQTT, Modbus) m'a permis de mieux comprendre leur fonctionnement dans des systèmes connectés. De plus, j'ai acquis une expérience précieuse en gestion de projet, travail en équipe et communication professionnelle.

* : Voir glossaire

Sommaire

Table des figures

I. Introduction

Dans le cadre de mon cursus universitaire, j'ai eu l'opportunité d'effectuer un stage de onze semaines au sein de l'entreprise Elyxoft. Fondée en 2020, Elyxoft est un bureau d'étude spécialisé dans la conception logicielle, l'expertise métier et dans la programmation embarquée, avec un accent particulier sur les objets communicants et la domotique. Basés sur l'expertise de son président, Sébastien Marion, Elyxoft collabore étroitement avec diverses entreprises du secteur pour offrir des solutions avancées, notamment dans le domaine des protocoles de communication radio faible débit comme Zigbee* ou Matter*. Elyxoft est une petite structure, une société par actions simplifiée (SAS) qui compte un capital social de 5000€ avec seulement 1 employé à ce jour, mon maître de stage et président de l'entreprise, Sébastien Marion. Durant la durée du stage, j'ai aussi été accompagné par un autre stagiaire, Fernando Cervantes.

Le cadre de travail chez Elyxoft est un environnement bienveillant et agréable, propice à la cohésion d'équipe grâce à un bureau unique qui facilite grandement la communication.

Ce stage a duré 11 semaines durant lesquelles j'ai eu l'occasion de prendre part à plusieurs projets. Tout d'abord, j'ai travaillé sur un projet annexe qui m'a permis de me familiariser avec l'environnement de développement et les technologies que j'allais utiliser pendant le reste du stage. Ce projet a durée une semaine. Il s'agissait de développer une touillette connectée capable d'afficher la température du café afin d'éviter les brûlures. L'objectif était aussi de découvrir le Bluetooth Low Energie (BLE*) et Qt*, un framework* multi-plateforme en C++ utilisé pour développer des applications avec des interfaces utilisateur graphiques compilables sur plusieurs systèmes d'exploitation. Après avoir expérimenté et être arrivé à un résultat satisfaisant, mon travail s'est concentré sur le sujet principal de ce stage, le projet ElyTicBridge, qui a occupé le reste de mon temps. C'est une passerelle protocolaire permettant de récupérer les informations de la télé-information client (TIC*) des compteurs électriques Linky et de les transmettre via divers protocoles comme MQTT*, Modbus*, Zigbee ou encore Matter. Pour ce projet, plusieurs tâches m'ont été attribuées. Tout d'abord la restructuration complète de l'architecture du code dans l'objectif de le rendre plus modulaire et réutilisable. Ensuite, mon attention s'est portée sur un module spécifique d'ElyTicBridge, le BLE. Dans le cadre de ce projet, il permet la configuration du bridge (identifiants WiFi, MQTT, etc). Un code était déjà existant, mais il était peu fonctionnel. L'objectif était d'améliorer le code et d'ajouter les paramètres de configuration qui n'étaient pas encore implémentés. Enfin, ma dernière tâche a été de réaliser une application avec Qt permettant de configurer ElyTicBridge grâce au module BLE mis en place auparavant.

Nous débuterons par une analyse du projet de la touillette connectée, détaillant d'abord le matériel nécessaire, puis la phase de développement embarqué et enfin le développement de l'application. Par la suite, nous aborderons le projet ElyTicBridge. Nous commencerons par une présentation du projet suivi d'un focus sur les différentes tâches que j'ai eu à réaliser comme la restructuration du code, le travail sur le module BLE et la conception et réalisation d'une application de configuration.

* : Voir glossaire

II. Projet touillette connectée

Dans le but de découvrir l'environnement de développement et les technologies que j'utiliserai pendant le reste de mon stage, mon maître de stage m'a proposé de travailler sur une touillette connectée. Ce projet, mené par des lycéens dans le cadre de leur cours de Sciences de l'Ingénieur (SI), permet de mesurer la température du liquide dans lequel la touillette est plongée grâce à un thermomètre relié à un microcontrôleur. Une LED RGB informe l'utilisateur lorsque le thé ou le café a atteint la température optimale. Les seuils de température ont été déterminés par les lycéens grâce à des calculs théoriques et des expériences. De plus, la touillette partage la température via BLE, ce qui a nécessité la création d'une application pour récupérer cette valeur.

II.1 Matériels et outils

Les lycéens ont été responsables des choix en termes de matériels et d'outils. Leur priorité était de privilégier la simplicité d'utilisation, compte tenu de leur niveau de compétence, débutant en programmation et en électronique. Ces décisions ont été prises afin de rendre le projet accessible et réalisable, tout en offrant une expérience d'apprentissage enrichissante.

Matériels

Pour ce projet, deux composants étaient nécessaires : la carte de développement et le thermomètre. La carte de développement choisie est une Seeed Studio XIAO nRF52840. Elle a l'avantage de pouvoir être programmée avec Arduino, une plateforme simple et accessible qui facilite la création de projets électroniques. Cette carte intègre aussi un module Bluetooth, une contrainte du projet. Pour le thermomètre, un Dallas 18B20 a été choisi. Il dispose également d'une librairie Arduino*, ce qui simplifie grandement son utilisation. La photo du prototype avec les composants soudés est disponible dans l'Annexe A.

Technologies

Côté développement embarqué, conforme aux spécifications matérielles, Arduino a été choisi pour sa facilité d'utilisation et la disponibilité de bibliothèques compatibles avec le matériel sélectionné. Cette solution est idéale pour les lycéens débutants en programmation.

Pour la partie application mobile, deux outils ont été sélectionnés avec des objectifs distincts. Tout d'abord, pour les lycéens, l'utilisation d'App Inventor a été privilégiés en raison de sa simplicité d'utilisation. App Inventor propose une programmation par bloc, similaire à Scratch, ce qui rend la création d'applications accessible même aux débutants. Le deuxième outil choisi est Qt. L'objectif était ici de se familiariser avec la création d'interfaces graphiques sous Qt et de comprendre les interactions avec le BLE en vue de projets futurs.

II.2 Développement embarqué

Le développement peut être découpé en trois parties : récupération de la température du thermomètre, gestion de la LED embarquée sur la carte de développement et partage de la température grâce au BLE.

Récupération de la température

Pour récupérer la température transmise par le thermomètre Dallas 18B20, une bibliothèque appelée DallasTemperature facilite le processus en offrant des fonctions telles que **requestTemperatures()**, qui demande la température actuelle du thermomètre, et

* : Voir glossaire

getTempCByIndex(), qui renvoie la valeur de la température en degré Celsius. Ainsi, la programmation est grandement simplifiée. Il suffit d'initialiser le thermomètre en spécifiant la broche de la carte à laquelle il est connecté, puis, grâce à ces fonctions, on obtient la température mesurée.

Gestion de la LED

L'objectif de la LED est d'informer l'utilisateur de la température de son café ou de son thé en utilisant trois seuils de température. Si la température est trop élevée, la LED doit s'allumer en rouge ; si la température est optimale, la LED doit s'allumer en vert ; si la température est trop basse, la LED doit s'allumer en bleu.

Pour commencer, on initialise les trois broches auxquelles sont connectées les trois composantes de couleur de la LED, en spécifiant que ces broches doivent fonctionner comme des sorties à l'aide de la fonction suivante : **pinMode(NuméroDeLaBroche, OUTPUT)**. Ensuite, on peut allumer ou éteindre chaque composante de couleur de la LED en envoyant un signal de niveau haut ou de niveau bas sur les broches. Cela peut être réalisé avec la fonction **digitalWrite(NuméroDeLaBroche, [HIGH | LOW])**. Une particularité de la LED embarquée est qu'elle est inversée, ce qui signifie qu'il faut envoyer un signal de niveau bas pour l'allumer et un signal de niveau haut pour l'éteindre. Ainsi, pour obtenir la couleur bleue, on envoie un signal de niveau bas sur la broche correspondant au bleu et un signal de niveau haut sur les broches correspondant au vert et au rouge.

Maintenant que nous comprenons le fonctionnement de la LED, nous pouvons mettre à jour sa couleur en fonction de la température mesurée. Pour cela, la LED doit être constamment mise à jour pour afficher la bonne couleur. Arduino nous permet d'écrire une fonction **loop** qui, comme son nom l'indique, est exécutée en boucle. Nous écrivons donc les instructions permettant de récupérer la température, puis nous mettons à jour la couleur de la LED en fonction de celle-ci.

Partie BLE

Afin de partager la température mesurée par le thermomètre, il a été décidé d'utiliser le BLE. C'est une version dérivée du Bluetooth classique, introduite pour répondre à la demande croissante de communication sans fil avec une faible consommation d'énergie. Sa principale différence est qu'il permet un transfert de données ponctuel, contrairement au Bluetooth classique, qui est conçu pour une communication bidirectionnelle en continu. Cela permet aux appareils utilisant le BLE de se mettre en veille lorsqu'il n'y a pas de communication, économisant ainsi de l'énergie.

Le BLE repose sur quatre concepts clé qui constituent les piliers de son fonctionnement. Tout d'abord, le GAP* (Generic Access Profile) est essentiel pour établir les bases de la communication et de l'interaction entre les dispositifs BLE. Il gère les connexions entre les appareils en leur attribuant des rôles spécifiques : un rôle de périphérique (Peripheral) pour un appareil qui émet et reçoit des données, et un rôle de central (Central) pour un appareil qui initie et gère les connexions. Il existe d'autres rôles, mais ils n'étaient pas pertinents pour ce projet. Ensuite, il y a la notion de GATT* (Generic Attribute Profile), qui est une architecture organisant et gérant les échanges de données entre les appareils BLE. Les deux dernières notions importantes sont les services* et les caractéristiques*. Un service est un ensemble de caractéristiques, correspondant souvent à un besoin spécifique. Une caractéristique est l'unité de base des données dans BLE, représentant une information précise, par exemple une température. Elle inclut des propriétés telles que la lecture ou l'écriture, permettant aux appareils connectés d'interagir avec les services et les caractéristiques proposés. Il existe également la propriété de notification, qui permet d'écrire

* : Voir glossaire

une valeur et de notifier les appareils connectés. Cette propriété a été utile dans le projet de la touillette connectée pour mettre à jour la température sur l'application en temps réel. Les services et caractéristiques sont identifiés par un numéro unique qu'on appelle UUID*.

Voici un schéma qui montre l'emboîtement des concepts du BLE.

Schéma réalisé avec draw.io [1].

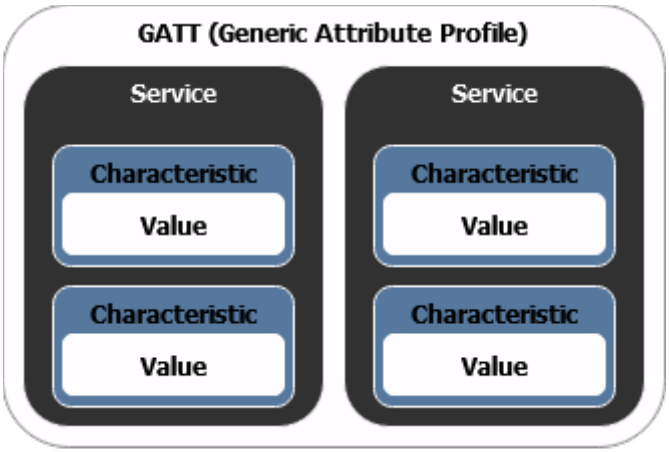

Figure 1 : Représentation d'un profil GATT

Pour mettre en place le BLE, Arduino met à disposition une bibliothèque appelée « **bluefruit** ». Elle permet de gérer le **GAP** et le **GATT** et intègre des classes pour les services et les caractéristiques, rendant la configuration facile. J'ai donc créé un service et une caractéristique grâce aux classes **BLEService** et **BLECharacteristic**. Ensuite, j'ai démarré l'**advertising**, étape où l'appareil met à disposition un service permettant aux autres appareils de le découvrir et d'initier une connexion. Enfin, dans la boucle où la température est lue, j'ai ajouté les instructions nécessaires pour que si une température différente de la précédente est lue, on notifie sur la caractéristique de température la nouvelle valeur.

II.3 Développement de l'application

Pour la partie application mobile, l'objectif était de pouvoir se connecter à la touillette puis d'afficher la température avec un message indiquant si c'est trop chaud, parfait ou trop froid. Pour cela, j'ai d'abord réalisé un prototype avec App Inventor puis j'ai fait une autre version avec Qt.

App Inventor

L'application devait contenir deux vues. La première permet de lancer le scan des appareils BLE et de se connecter à un appareil. La seconde affiche la température.

L'utilisation d'App Inventor étant assez facile, je n'ai pas rencontré de difficulté particulière pour la création du prototype.

La figure ci-dessous correspond aux deux vues de ce prototype. La première vue permet de scanner les appareils BLE puis d'en sélectionner un afin de s'y connecter. La deuxième vue affiche la température avec un message indiquant si c'est trop froid, trop chaud, ou à la bonne température.

* : Voir glossaire

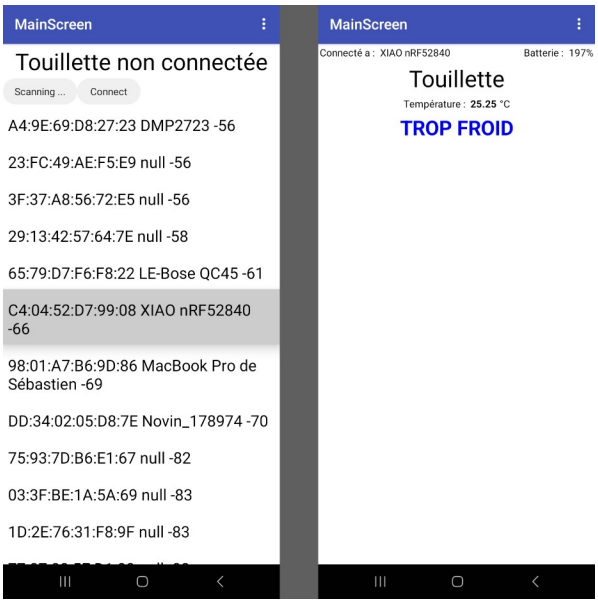

Figure 2 : Prototype de l'application Touillette connectée avec App Inventor

Qt

L'objectif de la réalisation de cette version de l'application avec Qt était que je découvre le framework et l'environnement de développement pour que je sois opérationnel pour les futurs projets.

Qt possède plusieurs bibliothèques graphiques pour afficher des boutons, des conteneurs, etc. Ce framework offre aussi la possibilité de gérer des « **SIGNALS** » et des « **SLOTS** ». Leur objectif est de pouvoir déclarer des signaux (**SIGNALS**) qui déclencheront les fonctions des **SLOTS** connectés à ces signaux. Cela est très pratique pour gérer les interactions utilisateur telles que les clics de souris ou les touches de clavier. Chaque élément des bibliothèques graphiques de Qt intègre déjà des **SIGNALS** et des **SLOTS**. Par exemple, les boutons ont un slot **onClicked()** qui s'active lorsqu'on clique sur le bouton. Un autre avantage de Qt est qu'il permet de compiler le code pour plusieurs systèmes d'exploitation, tels que Windows, MacOS, Android et iOS.

Pour réaliser l'application, je me suis basé sur un exemple proposé par Qt qui gère l'interaction avec le BLE. Cet exemple est un scanner permettant de découvrir les appareils à proximité, de s'y connecter et de lire les valeurs des caractéristiques. J'ai donc modifié cet exemple pour qu'il réponde à nos besoins.

Tout d'abord, j'ai modifié les vues pour en avoir uniquement deux, comme dans le prototype sur App Inventor. Ensuite, j'ai modifié le code de l'application pour qu'elle récupère automatiquement la valeur de la caractéristique sans demander à l'utilisateur de choisir le service auquel il veut se connecter, comme c'était le cas dans le scanner. J'ai également limité les appareils affichés à l'utilisateur lors du scan, pour n'afficher que ceux dont le nom commence par « XIAO nRF52840 », qui est le nom actuel de la touillette connectée (ce nom peut être changé). Ainsi, au démarrage de l'application, l'utilisateur peut se connecter à un appareil et voir apparaître la vue qui lui donne la température.

La figure ci-dessous illustre les vues de cette application. On y voit la vue dédiée au scan et à la connexion à un appareil BLE, ainsi que celle affichant la température.

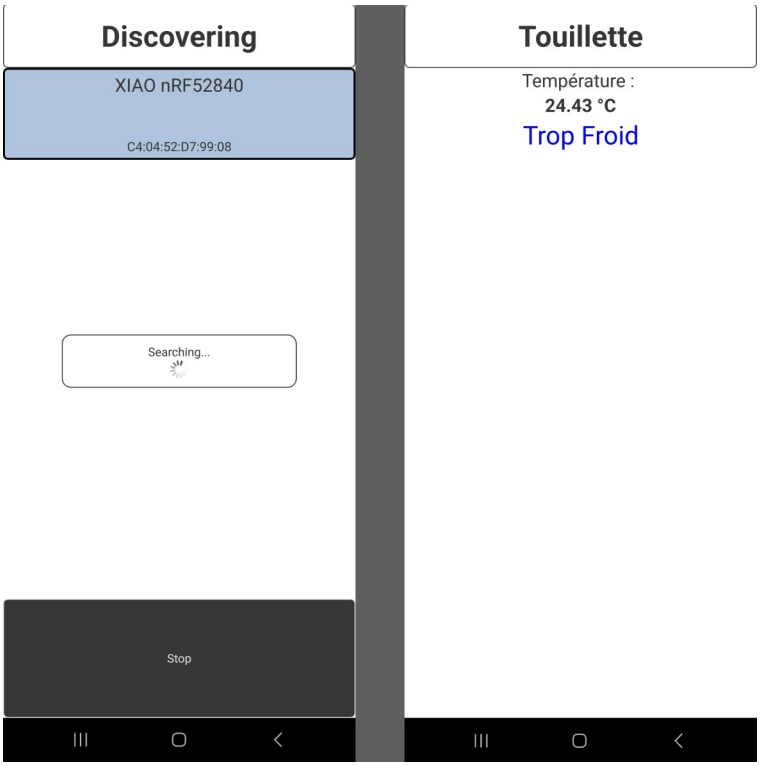

Figure 3 : Application Touillette connectée avec Qt

II.4 Conclusion

Ce projet a été une très bonne entrée en matière. Il a duré une semaine et m'a permis de découvrir et de prendre en main les technologies que j'ai utilisées pour le projet ElyTicBridge. La découverte du BLE et de son fonctionnement m'a notamment fait gagner beaucoup de temps par la suite. De plus, cette première application sous Qt a été enrichissante, car j'ai pu m'appuyer sur son code pour réaliser celle d'ElyTicBridge.

III. Projet ElyTicBridge

III.1 Présentation et cas d'utilisation

Présentation

Le projet ElyTicBridge s'intègre dans un contexte où la gestion de l'énergie constitue un enjeu crucial pour notre société. Son objectif est de faciliter la transition énergétique en déployant une solution de collecte de données sur la consommation électrique, permettant ainsi un meilleur contrôle de celle-ci. Pour cela, il exploite une technologie couramment présente dans les foyers français : la télé information client des compteurs Linky. Cette technologie permet de recueillir les informations de consommation environ toutes les secondes. L'objectif de ce produit est de récupérer ces information et de les retransmettre via plusieurs moyens de communication. En effet, l'un des défis d'ElyTicBridge est de pouvoir s'adapter à une grande diversité d'environnements et de contextes, tenant compte de la position du compteur, de l'accès au réseau internet ou non et des besoins spécifiques de l'utilisateur. C'est pourquoi il intègre plusieurs protocoles de communication tels que Zigbee ou Matter, largement utilisés dans le domaine de la domotique, permettant ainsi une intégration dans un environnement résidentiel. De plus, il prend en charge Modbus TCP, un protocole couramment utilisé dans le milieu industriel, ainsi que MQTT, principalement adopté par des utilisateurs ayant des compétences techniques avancées.

Cas d'utilisation

Le projet ElyTicBridge propose une variété de cas d'utilisation. Dans un contexte résidentiel, il permet de surveiller en temps réel la consommation énergétique d'un foyer, facilitant ainsi la gestion des appareils domestiques pour réduire la consommation et les coûts énergétiques. Par exemple, un propriétaire peut recevoir des alertes lorsque la consommation d'un appareil dépasse un certain seuil, permettant une intervention rapide pour éviter une surconsommation inutile. Prenons l'exemple d'une machine à café. ElyTicBridge peut détecter un pic de consommation à intervalle régulier. Le propriétaire est alors alerté et, après une petite investigation, peut se rendre compte que c'est la machine à café qui se met en préchauffage. Il peut alors choisir de l'éteindre la nuit pour éviter des coûts inutiles.

Dans un contexte industriel, ElyTicBridge permet aux entreprises de suivre l'utilisation de l'énergie de leurs équipements, optimisant ainsi les processus et prévenant les pannes coûteuses grâce à une maintenance prédictive. Par exemple, une usine peut surveiller la consommation énergétique de ses machines en temps réel et détecter des chutes de courant signalant une défaillance du matériel. Ainsi, l'entreprise peut programmer des maintenances avant que la machine ne tombe en panne et n'entraîne des arrêts de production coûteux.

Par ailleurs, pour les utilisateurs ayant des compétences techniques avancées, ElyTicBridge ouvre des possibilités d'intégration avec des systèmes de gestion d'énergie personnalisés et des plateformes IoT sophistiquées, permettant des analyses avancées et des automatisations complexes. Par exemple, un développeur peut créer un logiciel qui reçoit des données de consommation en temps réel et ajuste automatiquement les paramètres de divers appareils pour optimiser l'efficacité énergétique.

Langage utilisé

ElyTicBridge est une carte électronique équipée d'un microcontrôleur ESP32-C6 (photo disponible en annexe B), une puce électronique conçue par Espressif Systems, idéale pour créer des objets connectés. Ce microcontrôleur intègre le WiFi, le Bluetooth, Zigbee et d'autres protocoles de communication. Pour programmer sur cette plateforme, Espressif

fournit une bibliothèque appelée **ESP-IDF***, écrite en langage C. Ainsi, l'intégralité du projet ElyTicBridge est développé en C.

III.2 Restructuration

La première tâche qui m'a été confié a été de restructurer entièrement le projet pour qu'il soit plus modulaire. Une volonté de mon maître de stage était de séparer chaque module pour pouvoir les stocker dans des référentiels Git différents. Cela permet d'attribuer des licences différentes aux modules et de faciliter leur réutilisation. Aussi, en structurant le projet de cette manière, on peut utiliser le « **Component Manager** » mis à disposition avec **ESP-IDF**. Il s'agit d'un outil qui facilite la gestion et l'intégration des composants. Il permet de rechercher, installer, mettre à jour et gérer les dépendances des composants, ce qui simplifie grandement le processus de développement et assure une meilleure organisation et modularité du code.

Analyse de la structure existante

Le projet est découpé en modules, chacun dédié à une fonctionnalité spécifique. Ainsi, la structure était la suivante : un répertoire par module contenant les fichiers d'en-tête et de code C nécessaires au fonctionnement du module, ainsi qu'un fichier main.c pour l'initialisation et le démarrage des modules. Tout cela était stocké dans un seul référentiel Git.

La figure ci-dessous permet de voir que chaque module a son propre répertoire dans le répertoire **main**.

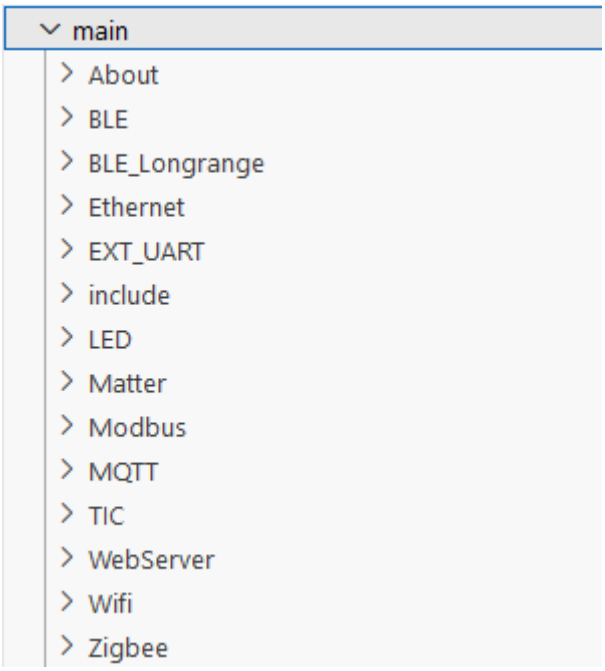

Figure 4 : Structure initiale du projet

Pour pouvoir séparer les modules, j'ai dû identifier ce qui les liait et les empêchait d'être séparés dans leur état actuel.

ElyTicBridge devant être adaptable, chaque module peut être configuré (par exemple pour les identifiants WiFi), et ils peuvent être configurés de deux manières différentes : via un serveur Web qui fournit une page HTML ou via le BLE (qui n'était pas fonctionnel à ce moment-là). Il y a plusieurs étapes clé lors de la configuration :

* : Voir glossaire

1. L'interface de configuration (WebServer ou BLE) reçoit la configuration envoyée par l'utilisateur et envoie un événement contenant la nouvelle configuration.
2. Tous les modules reçoivent la configuration et exécutent une fonction pour mettre à jour leur propre configuration.
3. L'interface de configuration stocke la nouvelle configuration en mémoire non-volatile*. Un type de mémoire qui conserve les données même en cas de coupure d'alimentation. C'est utile lors d'un redémarrage pour que les modules gardent la configuration précédente.

Dans cette version, j'ai remarqué plusieurs problèmes. Tout d'abord, il n'y avait qu'une seule structure de données contenant la configuration de tous les modules et un seul espace de stockage en mémoire non-volatile partagé par tous les modules. Cela rendait les modules interdépendants, de sorte que lorsqu'un module recevait une nouvelle configuration, tous les modules la recevaient également. De plus, le stockage en mémoire était géré par les modules WebServer et BLE, ce qui empêchait les autres modules de gérer leur propre configuration. Cela rendait difficile la réutilisation d'un module dans un autre projet, car il faudrait réécrire le processus de stockage en mémoire.

Le diagramme de séquence ci-dessous résume le processus de configuration pour un module A. On voit qu'un module B reçoit l'événement de configuration alors qu'aucune modification le concernant a été faite.

Diagramme de séquence réalisé avec Visual Paradigm [2].

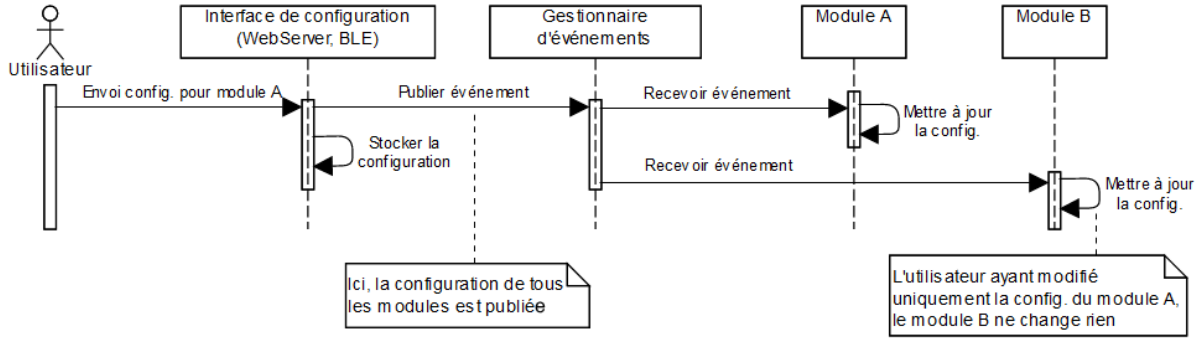

Figure 5 : Processus existant de configuration d'un module

Aussi, certains modules doivent obligatoirement être liés entre eux. C'est le cas du module TIC, qui permet de lire les informations envoyées par le compteur. Il est lié aux modules MQTT et Modbus TCP, qui publient les données extraites par le module TIC.

Conception et réalisation

Pour palier à ces problèmes et rendre les modules indépendants, j'ai fait des changements dans le processus de configuration.

Tout d'abord, j'ai ajouté un événement spécifique pour la configuration de chaque module. Ainsi, quand l'interface de configuration reçoit une configuration pour un module spécifique, elle peut la publier en ciblant ce module, évitant ainsi des mises à jour inutiles des modules non concernés. J'ai aussi déplacé la charge de stockage dans les modules eux-mêmes. Chaque module gère désormais son espace de stockage indépendamment des autres, rendant le code plus réutilisable. Enfin, j'ai ajouté un module « Configuration » qui fait l'interface entre les interfaces de configuration et les modules. Il met à disposition des

* : Voir glossaire

fonctions permettant de récupérer et de publier des configurations pour chaque module, améliorant ainsi la modularité.

Le diagramme de séquence ci-dessous montre le nouveau processus de configuration. On peut y observer l'ajout du module Configuration. Aussi, lorsque le module A reçoit une nouvelle configuration, le module B ne reçoit plus d'événements superflus. De plus, le module A est maintenant responsable du stockage de sa configuration en mémoire.

Diagramme de séquence réalisé avec Visual Paradigm [2].

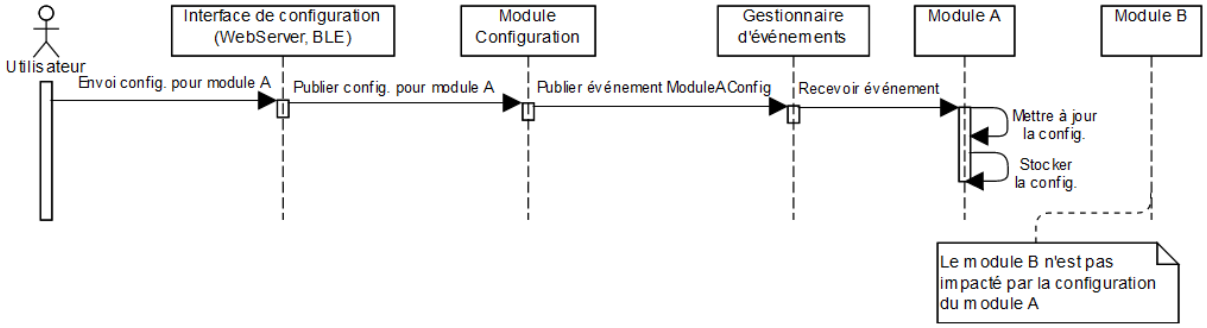

Figure 6: Nouveau processus de configuration d'un module

Pour les modules nécessitant des liaisons entre eux, il a fallu les diviser. Le code essentiel et réutilisable a été placé dans le composant associé, tandis que le code spécifique à ElyTicBridge est resté dans le module.

Le module TIC émet un événement chaque fois qu'il reçoit des données. Cet événement est utilisé pour faire communiquer les modules MQTT et Modbus TCP. Ces deux modules possèdent une fonction handler* qui est appelée à la réception d'un événement TIC. Cette fonction se charge de publier les données reçues via son protocole respectif. J'ai décidé de laisser cette fonction dans le code d'ElyTicBridge et de ne pas la transférer dans un composant, car elle effectue un travail spécifique à ElyTicBridge.

Pour lier la partie du composant avec celle restée dans ElyTicBridge, j'ai ajouté une fonction **register_event**. Comme son nom l'indique, cette fonction permet d'enregistrer un événement afin de faire la liaison entre la réception d'un événement et l'appel d'une fonction handler.

III.3 Module BLE

Le module BLE permet de configurer les différents modules d'ElyTicBridge, à la manière du module WebServer. Il permet par exemple de configurer le module WiFi afin qu'il se connecte à un réseau en entrant les identifiants.

Analyse de l'existant

À mon arrivée sur le projet, le module BLE existait déjà. Toute la partie concernant le GAP et le GATT était écrite. Il y avait également un service avec des caractéristiques pour la configuration du WiFi. Cependant, certaines fonctionnalités étaient manquantes.

Premièrement, il manquait des services pour que chaque module fonctionnel puisse être configuré. Il fallait donc ajouter les services pour les modules Ethernet, TIC, MQTT, Modbus TCP et BLE (qui doit pouvoir être désactivé).

* : Voir glossaire

De plus, les modifications apportées lors de la restructuration, notamment celles sur la partie configuration, rendaient ce module obsolète. Enfin, les caractéristiques contenant les valeurs de configuration ne se mettaient pas à jour et contenaient des informations par défaut, qui ne reflétaient pas la configuration réelle.

Réalisation

Ma première tâche a été de mettre à jour la manière de récupérer et de publier les configurations. J'ai adapté le code existant pour qu'il utilise le module Configuration, implémenté lors de la restructuration. Cette étape était plutôt facile, car ce module simplifie grandement les étapes de configuration en gérant lui-même les événements.

Ensuite, j'ai ajouté des services et des caractéristiques pour les modules manquants. Pour chaque module, il faut ajouter une base de données BLE, qui est une structure de données fournie par ESP-IDF. Cela permet de déclarer un service et ses caractéristiques, avec leur UUID, une valeur par défaut, la taille maximale de la valeur (en bits), des propriétés comme la lecture ou l'écriture, etc.

Il est également nécessaire de créer une fonction handler qui est appelée par le GATT à chaque fois qu'un événement BLE est reçu. Cette fonction permet notamment d'initialiser le service et les caractéristiques lors de l'initialisation du BLE. Elle permet aussi de publier les nouvelles configurations lorsqu'un événement d'écriture est reçu.

Enfin, pour mettre à jour les caractéristiques lors de changements de configuration, notamment, si c'est la partie WebServer qui effectue des modifications, j'ai utilisé les événements de configuration déclarés par chacun des modules. En effet, comme expliqué dans la partie restructuration, pour configurer un module, on envoie un événement qui est reçu par le module afin qu'il puisse mettre à jour sa configuration interne. Mon idée était d'utiliser cet événement pour mettre à jour les caractéristiques BLE. Ainsi, si l'utilisateur utilise le WebServer pour mettre à jour les identifiants de son serveur MQTT, le module Configuration enverra un événement **MQTT_CONFIG_EVENT**. Le module BLE peut alors écouter cet événement et récupérer les informations de configuration pour mettre à jour les caractéristiques.

Après avoir effectué ces modifications, le module BLE est fonctionnel et permet de configurer ElyTicBridge de la même manière que par le WebServer.

Optimisation

En testant la configuration par BLE, j'ai remarqué quelques instabilités. En envoyant la configuration complète d'un module, certaines valeurs étaient écrasées ou non prises en compte de manière aléatoire. Après une investigation, j'ai compris d'où venait le problème.

En fait, pour envoyer une configuration, on envoie les valeurs de chaque caractéristique une par une, créant ainsi dans le module BLE autant d'événements d'écriture qu'il y a de caractéristiques. À chaque événement d'écriture, une nouvelle configuration est publiée par le module BLE. Le module concerné par cette nouvelle configuration reçoit les paramètres et les écrit alors en mémoire. Cela crée de nombreux accès et écritures en mémoire très rapidement, rendant le système instable.

Pour pallier ce problème, j'ai décidé d'ajouter une caractéristique pour chaque module, qui, lorsqu'elle est écrite, publie la configuration. Cette caractéristique permet de valider que toute la configuration a été envoyée par l'appareil connecté et qu'on peut maintenant l'appliquer. Cela réduit grandement les écritures en mémoire et a rendu le système plus stable. J'ai décidé de lui attribuer un UUID terminant par « ff » pour pouvoir facilement la reconnaître.

III.4 Application de configuration via BLE

L'objectif de cette application est de se connecter au module BLE d'ElyTicBridge afin de visualiser et de modifier les valeurs contenues dans les caractéristiques pour configurer l'appareil. L'application doit être portable, ce qui a conduit à privilégier Qt comme framework pour son développement.

Réalisation de la logique applicative

Pour réaliser cette application, je me suis basé sur le code que j'avais précédemment développé lors du projet de la touillette connectée. J'avais donc déjà une base permettant de scanner les appareils BLE ayant un nom spécifique (que j'ai modifié pour qu'il soit « ElyTicBridge »). Ensuite, j'ai réutilisé l'exemple fourni par Qt, nommé « BluetoothLowEnergy Scanner », car il permet, une fois connecté à un appareil, de visualiser les services puis les caractéristiques. Ainsi, chaque service peut être vu comme un sous-menu correspondant à un module d'ElyTicBridge.

J'ai ensuite développé la fonctionnalité permettant de modifier la valeur d'une caractéristique. Aussi, lors de la découverte des caractéristiques, j'ai ajouté un code permettant d'enregistrer celle qui permet d'appliquer les modifications. Dans le module BLE, j'ai fait en sorte qu'elle ait un UUID finissant par « ff ». Ainsi, grâce à une expression régulière, l'application recherche si une caractéristique a un UUID de la forme « 0xYYff », où Y est une valeur hexadécimale quelconque. Si c'est le cas, on enregistre cette caractéristique dans une variable « **apply_characteristic** ». Cela m'a permis d'écrire une fonction « **applyChanges** » permettant d'appliquer les modifications faites à la configuration en modifiant la valeur de la caractéristique précédemment enregistrée.

Enfin, j'ai ajouté un fichier contenant deux HashMap* constante : **BLE_SRV_UUID** et **BLE_CHAR_UUID**. Une HashMap est une structure de données permettant d'associer une clé avec une valeur. Elles permettent de stocker les UUID des services et des caractéristiques et de leur attribuer des valeurs telles qu'un nom, une couleur ou, pour les caractéristiques, un type. Ces valeurs sont utiles pour la partie interface utilisateur.

Réalisation de l'interface utilisateur

Pour l'interface utilisateur, j'ai utilisé QML, un langage de description déclaratif conçu pour créer des interfaces utilisateur avec Qt.

L'interface devait comporter trois vues :

1. **Vue de scan des appareils et de connexion** : pour trouver et se connecter aux appareils BLE.
2. **Vue de sélection du module à configurer** : pour choisir le service à configurer.
3. **Vue des caractéristiques** : pour afficher et modifier les caractéristiques du module sélectionné.

Je suis parti de l'exemple « BluetoothLowEnergy Scanner » qui possédait déjà ces trois vues. Mon travail a consisté à les adapter pour qu'elles intègrent la possibilité de modifier les caractéristiques, tout en revoyant le style de l'application pour correspondre aux codes de l'interface Web mise à disposition par le WebServer, comme demandé par mon maître de stage.

La figure ci-dessous montre les différentes vues de l'application BluetoothLowEnergy Scanner. La première vue correspond au scan et à la sélection d'un appareil BLE afin de s'y

* : Voir glossaire

connecter. La seconde vue affiche les services disponibles sur l'appareil, que l'on peut sélectionner pour accéder à la troisième vue. Cette dernière affiche les caractéristiques avec leur valeur et leurs propriétés.

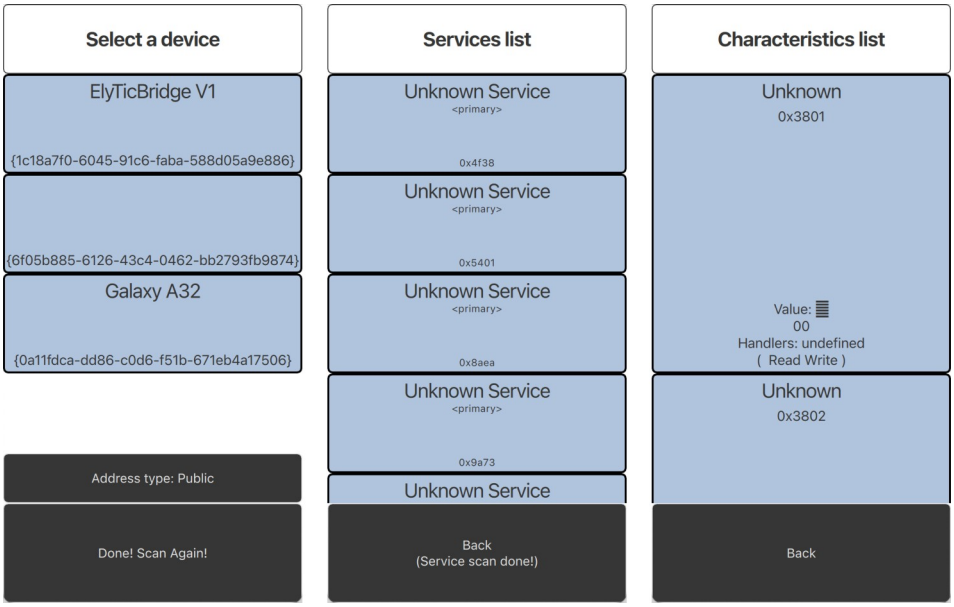

Figure 7 : Vues de l'application exemple « BluetoothLowEnergy Scanner »

En QML, il existe des briques graphiques de base comme des rectangles, du texte ou des boutons. On peut aussi créer nos propres composants graphiques en partant d'une brique existante et en ajoutant des propriétés ou en les combinant. Pour permettre la modification des caractéristiques, j'ai créé un élément **TextInput** composé d'un label et d'une entrée texte. Pour la première version de l'application, ce **TextInput** a été utilisé pour permettre à l'utilisateur de visualiser et de modifier toutes les caractéristiques. Cette version a permis de valider la partie logique de l'application.

Ensuite, je me suis concentré sur le design. L'interface Web avait un code couleur pour les services (voir annexe C), j'ai donc intégré ces couleurs. Pour cela, j'ai utilisé la HashMap **BLE_SRV_UUID** dont j'ai parlé dans la partie précédente. Il s'agit d'une HashMap ayant pour clé une chaîne de caractères (le UUID du service) et pour valeur une autre HashMap contenant des propriétés comme le nom du service à afficher et la couleur. J'ai donc ajouté le code permettant de récupérer la couleur de chaque service et de l'appliquer lors de la construction de la vue correspondant aux services. J'ai aussi récupéré cette couleur pour la vue des caractéristiques pour plus d'homogénéité.

Pour la vue des caractéristiques, j'ai ajouté différents types d'entrées, tels qu'un bouton à bascule, une entrée pour les mots de passe et un bouton spécifique pour la partie TIC, permettant de choisir le mode de transfert de données du compteur. Une fois ces composants graphiques créés, j'ai utilisé la propriété **type** de la HashMap **BLE_CHAR_UUID** pour choisir quel type d'entrée utiliser pour chaque caractéristique.

Finalement, j'ai ajouté un bouton « **send** » et un bouton « **refresh** » pour mettre à jour les caractéristiques et rafraîchir les informations affichées.

La figure ci-dessous présente le résultat de l'application. La première vue correspond au scan et à la sélection d'un appareil BLE. J'ai utilisé la couleur violette pour le bouton afin de rappeler le logo d'Elyxoft. La seconde vue affiche les services mis à disposition par ElyTicBridge. Les couleurs utilisées ici sont celles de l'interface Web (voir annexe C). Enfin,

les deux dernières vues montrent la configuration du module TIC et du module WiFi. On peut remarquer que les boutons à bascule reprennent le style de l'interface Web.

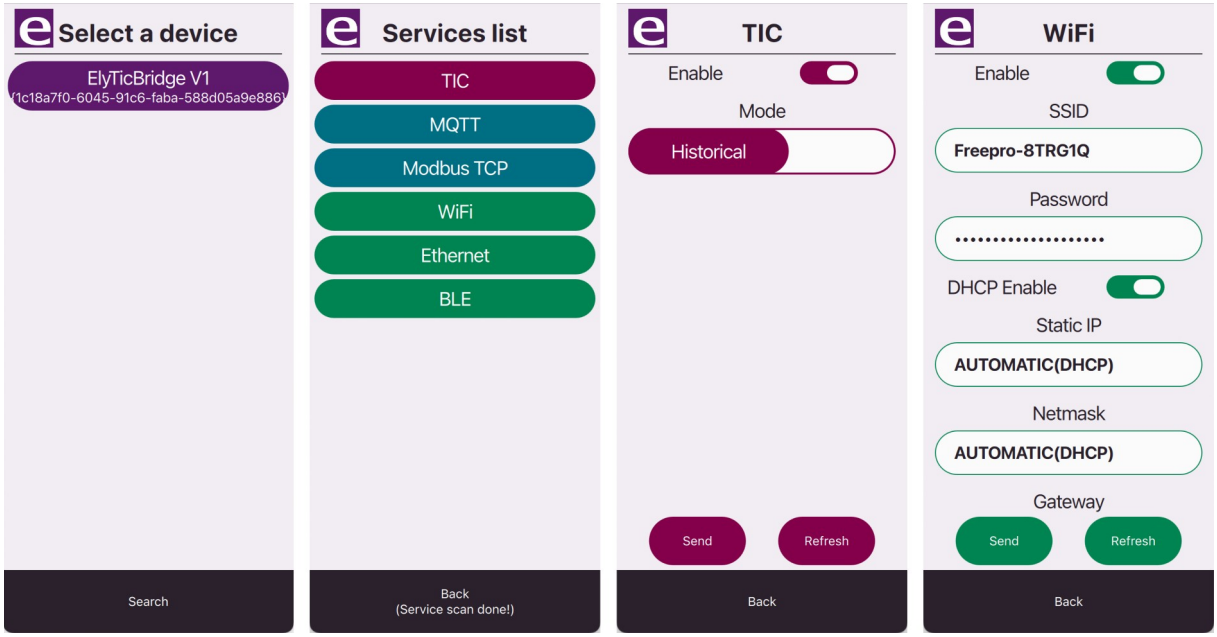

Figure 8 : Résultat de l'application de configuration

Il y a encore des améliorations à faire sur cette application. Notamment ajouter du retour pour l'utilisateur pour connaître l'état des différents modules, tel que l'état et l'adresse IP du WiFi ou de l'Ethernet. Une autre volonté de mon maître de stage est d'ajouter une vue permettant de visualiser les informations remontées par la TIC directement depuis l'application.

III.5 Conclusion

ElyTicBridge a été un projet très complet qui m'a permis d'apprendre et de mettre en application les concepts vus en cours. J'ai pu améliorer mes compétences dans les langages C et C++ et j'ai découvert le framework Qt. Aussi, j'ai découvert la programmation embarquée et notamment les contraintes liées à ce type de programmation.

IV. Conclusion

Ce stage chez Elyxoft a été une expérience enrichissante. Il m'a permis de mettre en pratique les compétences développées en cours tout en découvrant de nouvelles technologies. J'ai pu appliquer et approfondir mes compétences dans les langages C et C++, ce qui m'a offert une meilleure maîtrise de la programmation bas niveau.

J'ai aussi découvert la programmation embarquée. En travaillant avec des ESP32, j'ai compris les particularités et les défis de la programmation pour des systèmes embarqués. Cette expérience m'a familiarisé avec les contraintes spécifiques de ce domaine, comme la gestion des ressources limitées et la nécessité d'optimiser les performances.

Un autre aspect de ce stage a été l'utilisation de Qt pour développer des interfaces utilisateur. J'ai beaucoup apprécié ce framework, car il permet de créer des applications multi-plateformes de manière efficace. La possibilité de compiler le code pour plusieurs systèmes d'exploitation, comme Windows, MacOS, Android et iOS, m'a montré la polyvalence et la puissance de Qt.

Enfin, j'ai eu l'opportunité d'expérimenter avec divers protocoles de communication, notamment BLE, MQTT et Modbus TCP. Cela m'a permis de comprendre leur fonctionnement et leurs applications dans des systèmes connectés.

Au-delà des compétences techniques, ce stage m'a également permis de découvrir le milieu professionnel. J'ai appris à travailler en équipe, à gérer des projets et à communiquer de manière professionnelle. C'est une expérience que j'ai beaucoup appréciée et qui me motive à continuer à travailler dans ce domaine.

Glossaire

— **Arduino** : Une plateforme open-source utilisée pour le prototypage de projets électroniques. Elle comprend à la fois du matériel (cartes électroniques) et un environnement de développement logiciel.

— **BLE** (Bluetooth Low Energy) : Une technologie de communication sans fil à courte portée, conçue pour réduire la consommation d'énergie par rapport au Bluetooth classique.

— **Caractéristique** : Dans le contexte du BLE, une unité de données identifiable représentant une propriété spécifique d'un service.

— **ESP-IDF** (Espressif IoT Development Framework) : Un ensemble de bibliothèques, de pilotes et d'outils de développement logiciel pour les microcontrôleurs Espressif, notamment ceux utilisés dans les projets IoT.

— **Framework** : Une infrastructure logicielle qui fournit des fonctionnalités génériques et des outils pour aider au développement d'applications.

— **GAP** (Generic Access Profile) : Un profil Bluetooth standard qui définit les procédures pour l'établissement de connexion et la découverte de dispositifs.

— **GATT** (Generic Attribute Profile) : Un profil Bluetooth standard qui définit la structure des données échangées entre les dispositifs BLE.

— **Fonction handler** : Une fonction exécutée à la réception d'un d'événements spécifique.

— **HashMap** : Une structure de données qui associe des clés à des valeurs, permettant un accès rapide et efficace aux éléments en fonction de leur clé.

— **Matter** : Un protocole de connectivité standardisé destiné à faciliter l'interopérabilité entre les appareils intelligents dans l'écosystème de l'IoT.

— **MQTT** (Message Queuing Telemetry Transport) : Un protocole de messagerie légère qui fournit une méthode efficace pour la transmission de données entre des appareils connectés à Internet.

— **Modbus** : Un protocole de communication série utilisé principalement dans les systèmes de contrôle industriel pour l'échange de données entre des dispositifs électroniques.

— **Mémoire Non-Volatile** : Un type de mémoire qui conserve les données même lorsque l'alimentation électrique est coupée, par opposition à la mémoire volatile qui perd les données dans de telles situations.

— **Qt** : Un framework multi-plateforme pour le développement d'applications logicielles, offrant des outils et des bibliothèques pour créer des interfaces utilisateur graphiques et d'autres fonctionnalités.

— **QML** (Qt Modeling Language) : Un langage de balisage déclaratif utilisé pour la conception d'interfaces utilisateur dans le framework Qt.

— **TIC** (Télé-Information Client) : Un système de communication disponible sur les compteurs électrique Linky, permettant de transmettre les informations sur la consommation électrique d'un foyer.

— **Service** : Dans le contexte du BLE, une collection de caractéristiques représentant une fonctionnalité sur un périphérique.

— **UUID** (Universally Unique Identifier) : Dans le contexte du BLE, un identifiant unique utilisé pour identifier de manière unique les services, les caractéristiques et d'autres entités dans la communication entre les périphériques BLE.

— **Zigbee** : Un protocole de connectivité standardisé destiné à faciliter l'interopérabilité entre les appareils intelligents dans l'écosystème de l'IoT.

Sitographie

[1] draw.io : https://app.diagrams.net/

[2] Visual paradigm : https://www.visual-paradigm.com/

[3] Site de l'entreprise Elyxoft : https://www.elyxoft.fr/

Annexes

IV.1 Annexe A : Touillette connectée

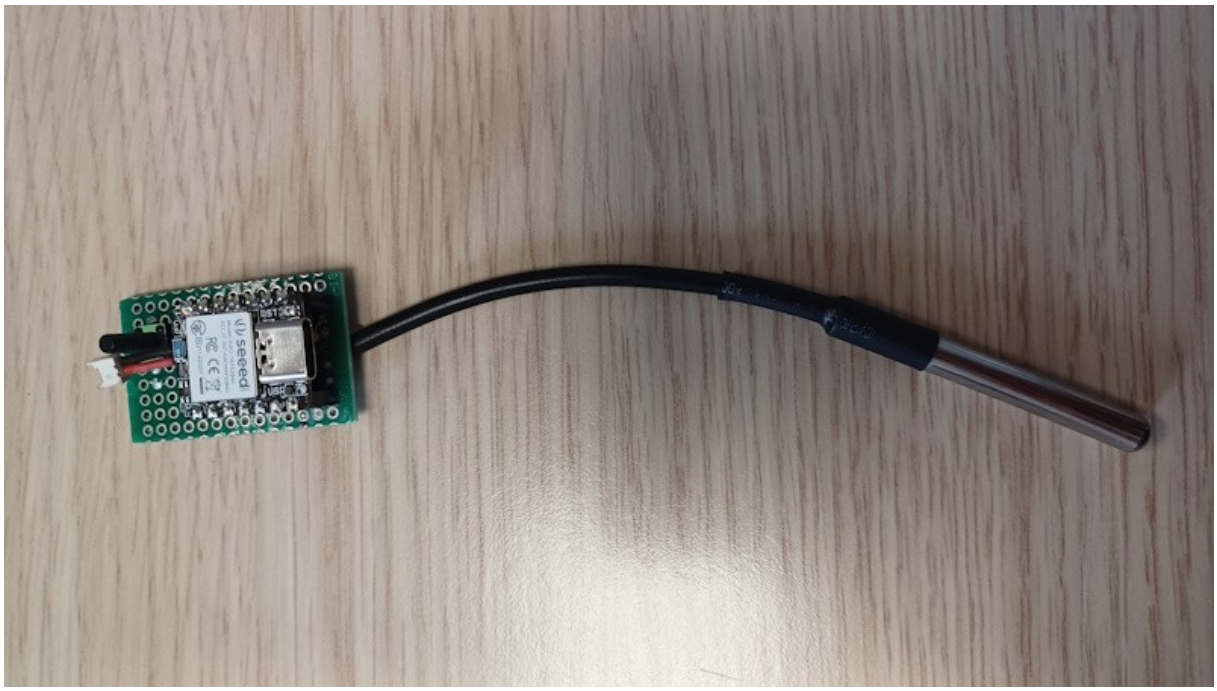

Figure 9 : Prototype de la touillette connectée

IV.2 Annexe B : ElyTicBridge

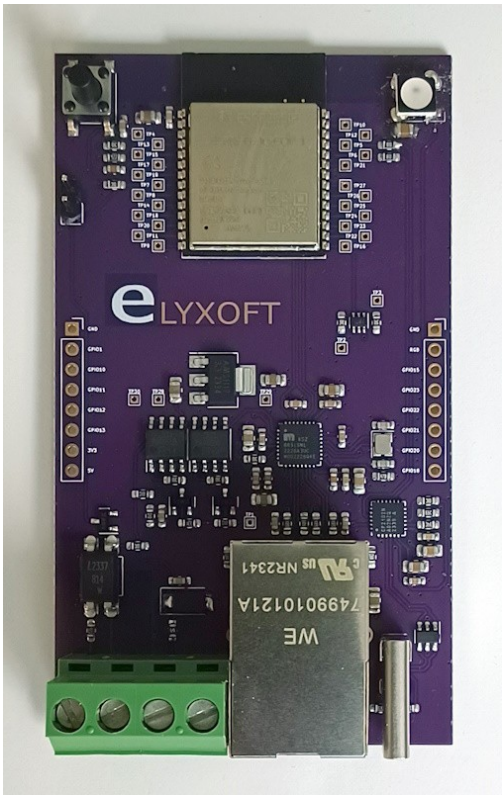

Figure 9 : Carte électronique du projet ElyTicBridge

IV.3 Annexe C : Interface Web d'ElyTicBridge

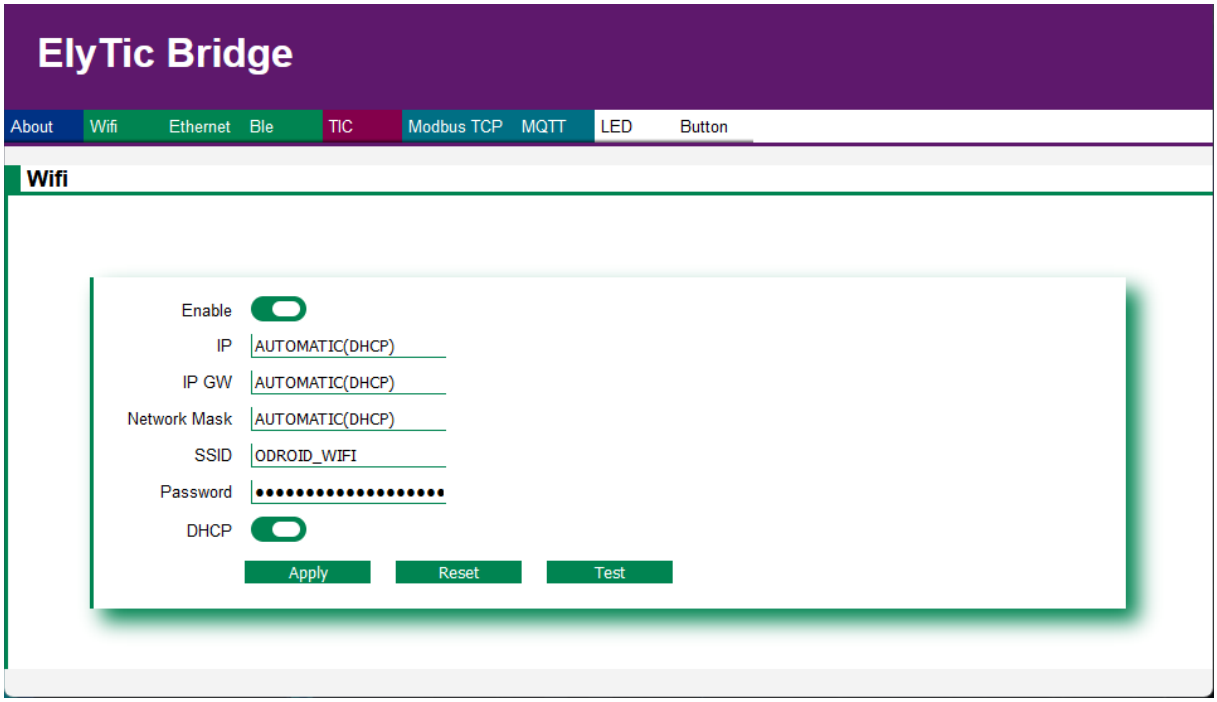

Figure 10 : Interface web pour la configuration d'ElyTicBridge (menu WiFi)

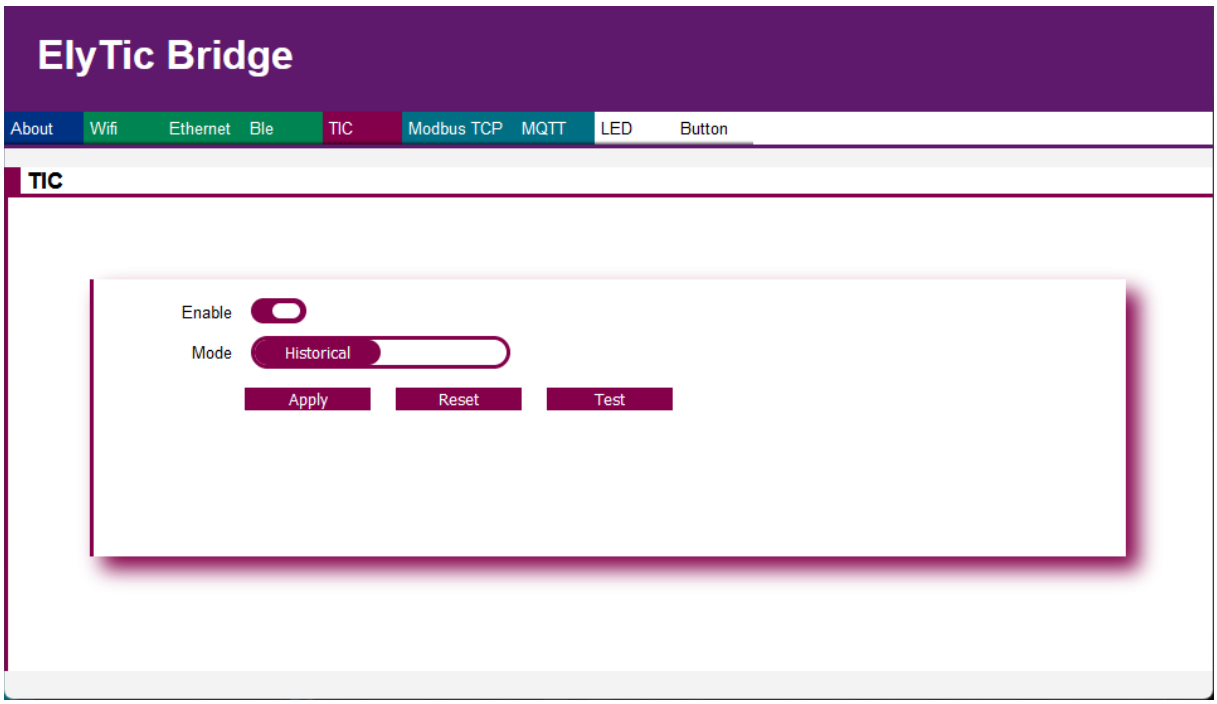

Figure 11 : Interface web pour la configuration d'ElyTicBridge (menu TIC)

Résumé :

Mon stage de onze semaines chez Elyxoft, une entreprise spécialisée dans la conception logicielle et la programmation embarquée, avait pour objectif de participer au développement d'ElyTicBridge. Cette passerelle protocolaire permet de récupérer et de transmettre les informations des compteurs électriques Linky via divers protocoles (MQTT, Modbus, Zigbee, Matter).

J'ai d'abord restructuré l'architecture du code pour la rendre plus modulaire, facilitant ainsi la gestion des modules via des référentiels Git distincts. Ensuite, j'ai travaillé sur le module BLE, qui permet de configurer les différents modules du produit, notamment en modifiant les identifiants WiFi pour connecter ElyTicBridge à un réseau. Enfin, j'ai développé une application Qt permettant d'utiliser le module BLE pour configurer ElyTicBridge.

Ce stage m'a permis de découvrir le milieu professionnel, de renforcer mes compétences techniques et de découvrir de nouvelles technologies.

Mots clés :

Programmation embarquée, objets communicants, application mobile, Bluetooth Low Energy (BLE), Qt, ESP32, Télé Information Client (TIC), Linky, gestion de l'énergie.

Abstract :

Enhancing ElyTicBridge: Modular Architecture and BLE Configuration for Efficient Electricity Consumption Data Transmission

Mathieu Camus

abstract: The ElyTicBridge project aims to create a protocol gateway that collects electricity consumption data from Linky meters and transmits it using various communication protocols such as MQTT, Modbus, Zigbee, and Matter. The objective of this internship was to enhance the architecture and functionality of ElyTicBridge, making it more modular and efficient while incorporating new features. The approach involved restructuring the codebase to improve modularity, refining the BLE module to enable full configuration of the product via this protocol, and developing a Qt-based application for configuration via BLE. The results demonstrated a more modular architecture, improved BLE configuration processes, and a user-friendly application for device configuration. These advancements have enhanced the performance and usability of ElyTicBridge. The internship provided practical experience in embedded programming and software development, emphasizing the importance of modular design and efficient communication protocols in engineering projects.

Keywords: Embedded programming, communicating objects, mobile applications, Bluetooth Low Energy (BLE), Qt, ESP32, TIC, Linky, energy management.