DEPARTEMENT INFORMATIQUE - IUT 2 GRENOBLE

Année Universitaire 2024-2025

MEMOIRE D'ALTERNANCE

DÉVELOPPEMENT D'UN BANC DE TESTS AUTOMATISÉ POUR DES PROTOCOLES RADIO DOMOTIQUES CHEZ SCHNEIDER ELECTRIC

Elyxoft

Présenté par

Mathieu Camus

3ème Année Parcours A

Jury

IUT : Mme. Dupuy-Chessa

IUT : Mme. Hamon

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 et tout usage à un outil doté d'IA a été mentionné et sera de ma responsabilité.

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, Rives Le, 14 juin 2025

Signature

Je tiens à remercier tout d'abord **Sébastien Marion**, mon maître d'apprentissage chez Elyxoft, pour m'avoir permis de travailler sur un projet d'une telle envergure et à fort enjeu stratégique pour l'entreprise. Sa confiance, son accompagnement ponctuel et son expertise sur les aspects techniques ont été essentiels à la réussite de cette expérience.

Je remercie également **Élodie Chargy**, cheffe de projet au sein de la Wireless Connectivity Team de Schneider Electric, pour m'avoir intégré pleinement à l'équipe, pour sa disponibilité et pour les nombreux échanges constructifs tout au long du projet. Son encadrement m'a permis de progresser rapidement dans un environnement exigeant.

Mes remerciements vont également à **Sophie Dupuy-Chessa**, ma tutrice universitaire, pour son suivi attentif durant toute l'année, ses conseils avisés, ainsi que sa relecture attentive de ce rapport.

Je souhaite aussi remercier l'ensemble des collaborateurs de **Schneider Electric** avec qui j'ai eu le plaisir de travailler au quotidien. Leur accueil et leur collaboration ont grandement enrichi cette alternance, tant sur le plan technique qu'humain.

Enfin, je tiens à exprimer ma reconnaissance à **l'ensemble de l'équipe pédagogique de l'IUT2 de Grenoble**, pour la qualité de l'enseignement dispensé. Les bases solides en programmation, en administration des systèmes et en rédaction technique que j'y ai acquises m'ont été précieuses tout au long de cette mission.

Enfin, je remercie **mes parents**, pour leur soutien constant et leur aide précieuse au quotidien — notamment en me prêtant leur voiture, sans laquelle mes trajets professionnels n'auraient pas été possibles.

Et un mot tout particulier pour **ma compagne**, pour sa patience, ses encouragements et sa présence rassurante tout au long de cette année d'alternance.

Sommaire

Table des figures

I. Introduction

Dans le cadre de ma troisième année de BUT Informatique, j'ai effectué mon alternance au sein d'Elyxoft, une entreprise spécialisée dans la conception de solutions logicielles pour les objets communicants. Cette alternance constitue une immersion longue dans un environnement professionnel, alliant développement logiciel, systèmes embarqués et protocoles de communication*.

Ma mission principale a été réalisée pour le compte de Schneider Electric, un acteur majeur du secteur de l'énergie et de la domotique. En collaboration avec leur équipe Wireless Connectivity Team (WCT), j'ai participé à la conception et au développement d'un banc de tests automatisé destiné à évaluer les performances des protocoles Zigbee* et Thread*, deux standards de communication sans fil utilisés dans les objets connectés.

Le projet portait sur la mise en place d'une infrastructure reposant sur des micro-ordinateurs (Raspberry Pi*) et des modules radio, capable de simuler des scénarios complexes en réseau maillé*. L'objectif était de fournir un environnement reproductible et flexible pour tester les comportements réels des protocoles Zigbee et Thread dans des conditions variées, dans le but de guider les choix technologiques de Schneider pour ses futurs produits.

Ce rapport présente le contexte professionnel et technique de cette mission, les choix d'architecture mis en œuvre ainsi que les développements réalisés.

Dans un premier temps, nous détaillerons l'environnement professionnel dans lequel le projet s'inscrit, puis nous présenterons les objectifs du banc de test et l'architecture mise en place. La suite du rapport exposera le fonctionnement de l'image système* développée (multitool), les choix techniques associés, ainsi que les travaux d'intégration et de tests. Enfin, les dernières parties du rapport reviendront sur la collaboration avec l'équipe WCT et les évolutions apportées au projet au fil des échanges, avant d'aborder les impacts techniques et sociétaux de cette mission, puis de conclure par un bilan personnel et professionnel de cette expérience.

II. Contexte professionnel et technique

II.1. Présentation d'Elyxoft

Elyxoft est le bureau d'études dans lequel je réalise mon alternance depuis septembre. Il s'agit d'une structure à taille humaine, composée de quatre personnes : Sébastien Marion, fondateur de l'entreprise et mon maître d'apprentissage, Pascal Guedon, ingénieur orienté intelligence artificielle recruté en décembre, Lynn Hayot, stagiaire en deuxième année de BUT Informatique, et moi-même. Cette petite taille donne à l'entreprise une grande réactivité et une proximité entre ses membres, tout en imposant un haut niveau d'autonomie.

L'activité d'Elyxoft se concentre sur la conception de solutions logicielles autour des objets communicants. L'entreprise intervient principalement en tant que prestataire technique, avec des missions orientées prototypage, développement embarqué et accompagnement client. Ses domaines d'expertise couvrent un ensemble de technologies liées aux systèmes embarqués et aux protocoles de communication, dans des contextes variés allant du confort thermique aux applications industrielles.

Elyxoft entretient depuis plusieurs années une relation de confiance avec Schneider Electric. L'entreprise agit comme renfort technique pour différents projets stratégiques. C'est dans ce cadre que s'inscrit ma mission actuelle. Bien que rattaché à Elyxoft, j'interagis quotidiennement avec une équipe technique de Schneider, de manière assez indépendante. Mon maître d'apprentissage intervient surtout en appui ponctuel, notamment sur les aspects liés au développement bas niveau ou aux problématiques embarquées spécifiques.

II.2. Schneider Electric et la Wireless Connectivity Team

Schneider Electric est une entreprise française multinationale spécialisée dans la gestion de l'énergie et l'automatisation, avec une présence dans le domaine de la domotique. Dans le cadre de mon alternance chez Elyxoft, j'ai été amené à collaborer avec l'une de leurs équipes internes : la Wireless Connectivity Team (WCT).

Cette équipe, composée d'un peu plus d'une vingtaine de collaborateurs, joue un rôle stratégique dans le groupe. Elle ne conçoit pas de produits finis, mais agit comme une équipe d'expertise transversale chargée d'anticiper l'évolution des technologies de communication radio, d'en étudier les performances, et d'accompagner les équipes produits dans leurs choix technologiques. Elle est positionnée très en amont dans la chaîne de développement de Schneider, et doit ainsi être capable d'évaluer les technologies émergentes pour orienter les futurs projets du groupe. L'arrivée du standard Matter, par exemple, a amené la WCT à se pencher de près sur le protocole Thread, qui en constitue la couche réseau. L'un des objectifs du projet sur lequel nous travaillons est de mettre au point un banc de tests afin de permettre à l'équipe d'évaluer concrètement cette technologie (capacité de passage à l'échelle, robustesse, performances, etc.) pour pouvoir formuler des recommandations fondées auprès des équipes produits.

Au sein de la WCT, nous sommes actuellement quatre à travailler sur le projet de banc de tests :

- Élodie Chargy, ingénieure en charge du pilotage du projet, qui assure également la coordination technique et organisationnelle,

- Lucien Dinh, ingénieur développeur, qui contribue aux spécifications, à la conception et au développement du banc,
- Joris Derewiany, stagiaire arrivé récemment, qui est chargé de réaliser des tests utilisateurs sur les premières versions du système,
- et moi-même, alternant chez Elyxoft, intégré à part entière à cette équipe projet.

Bien que je ne sois pas physiquement présent dans les locaux de Schneider (je travaille à distance depuis les bureaux d'Elyxoft), je suis pleinement intégré à l'équipe. Nous échangeons quotidiennement via Microsoft Teams, et tenons un point hebdomadaire par visioconférence pour discuter de l'avancée du projet, des retours fonctionnels, des évolutions à envisager ou encore des nouvelles fonctionnalités à intégrer. Ces échanges réguliers nourrissent la conception et l'infrastructure globale, en tenant compte des usages concrets pressentis par l'équipe, mais aussi des retours sur les premiers tests.

Notre travail collaboratif s'appuie également sur GitHub, utilisé comme plateforme centrale de gestion de projet. Les tâches sont discutées lors du point hebdomadaire et formalisées sous forme d'issues intégrées à un tableau Kanban. Chaque tâche fait l'objet d'une branche dédiée, suivie d'une pull request soumise à relecture par au moins un membre de l'équipe. Cette approche permet de garantir la qualité du code et de favoriser un développement itératif, guidé par les retours réguliers.

II.3. Notions clés du développement embarqué

Les objets connectés, comme les capteurs ou les dispositifs domotiques, reposent souvent sur des composants électroniques programmables. Pour mieux comprendre les développements décrits dans ce rapport, il est utile de clarifier quelques notions fondamentales propres à l'univers des systèmes embarqués.

Microcontrôleur : un mini-ordinateur autonome

Un microcontrôleur est une puce électronique conçue pour exécuter un programme simple, sans système d'exploitation complet. Il regroupe dans un même composant un processeur, de la mémoire et des interfaces de communication. Contrairement à un ordinateur classique, il est optimisé pour réaliser des tâches précises, souvent en temps réel, avec une très faible consommation d'énergie.

Ce type de composant est utilisé dans la plupart des objets connectés actuels.

Micrologiciel (firmware) : le programme embarqué

Le micrologiciel* (ou *firmware**) est le programme que le microcontrôleur exécute. Il détermine le comportement du dispositif : par exemple, rejoindre un réseau sans fil, communiquer avec d'autres appareils ou gérer des capteurs.

Le micrologiciel est stocké directement dans la mémoire du microcontrôleur et s'exécute automatiquement dès sa mise sous tension.

Téléversement d'un micrologiciel (flashage)

On appelle téléversement (ou *flashage*) l'opération qui consiste à installer un micrologiciel dans un microcontrôleur. Cette étape est indispensable pour initialiser ou modifier le comportement d'un appareil. Elle se fait généralement via un port de

communication (USB, série, etc.) et nécessite que l'appareil soit dans un état particulier : le mode bootloader*.

Le bootloader : passerelle de programmation

Le bootloader est un petit programme intégré au microcontrôleur, dont le rôle est de permettre l'installation d'un micrologiciel. Lorsqu'un appareil démarre en mode bootloader, il est prêt à recevoir un nouveau programme. Ce mode est souvent utilisé lors de la fabrication, de la mise à jour ou de la reconfiguration d'un objet.

Une fois le micrologiciel téléversé avec succès, l'appareil peut redémarrer en mode applicatif, c'est-à-dire exécuter normalement le programme installé.

Deux états principaux : bootloader et applicatif

Un microcontrôleur passe typiquement par deux états :

- Le mode bootloader : il attend un micrologiciel à installer. L'appareil ne fonctionne pas encore normalement.
- Le mode applicatif : le micrologiciel est en place, et l'appareil exécute sa fonction prévue.

Ce mécanisme de basculement entre mode bootloader et mode applicatif est central dans les projets embarqués. Il sera largement utilisé dans le reste de ce rapport, notamment pour expliquer comment les dispositifs de test sont préparés et reconfigurés automatiquement avant chaque expérience.

II.4. Présentation des technologies radio

Les protocoles Zigbee et Thread reposent sur des communications radio : ce sont des échanges d'informations sans fil entre appareils, via des ondes électromagnétiques. Pour mieux comprendre, on peut les comparer au Wi-Fi, que tout le monde utilise pour connecter un ordinateur ou un smartphone à une box Internet. Comme le Wi-Fi, Zigbee et Thread permettent de faire circuler des données sans câble, mais ils sont conçus pour d'autres usages : consommer très peu d'énergie et faire fonctionner des objets connectés en réseau, même dans des environnements contraints comme une maison ou un bâtiment.

La norme IEEE 802.15.4 : la base commune

Zigbee et Thread utilisent tous les deux une même technologie radio de base, appelée IEEE 802.15.4*. Cette norme définit les règles fondamentales pour permettre aux appareils de communiquer entre eux sans fil. Elle précise par exemple :

- quelles fréquences radio utiliser (comme le Wi-Fi, elle utilise la bande des 2,4 GHz),
- à quelle vitesse les données peuvent être envoyées (jusqu'à 250 kilobits par seconde),
- comment éviter que deux appareils parlent en même temps et se gênent.

On peut dire que c'est le "langage de base" que les appareils utilisent pour s'entendre sur comment envoyer des messages radio. Ensuite, des protocoles comme Zigbee ou Thread viennent se construire au-dessus de cette base pour ajouter des fonctions plus avancées, comme la création de réseaux maillés ou la sécurisation des échanges.

Zigbee : un réseau maillé pour la domotique

Zigbee est un protocole de communication sans fil spécifiquement conçu pour les objets connectés à faible consommation d'énergie, comme les ampoules intelligentes, les capteurs de température ou les volets roulants. Contrairement au Wi-Fi, qui consomme davantage et repose souvent sur un point d'accès central (comme une box), Zigbee s'appuie sur un réseau maillé. Cela signifie que chaque appareil peut relayer les messages des autres, un peu comme si les objets formaient une chaîne coopérative pour faire circuler l'information. Ce fonctionnement rend le réseau plus robuste : si un chemin est bloqué, les données peuvent passer par un autre itinéraire.

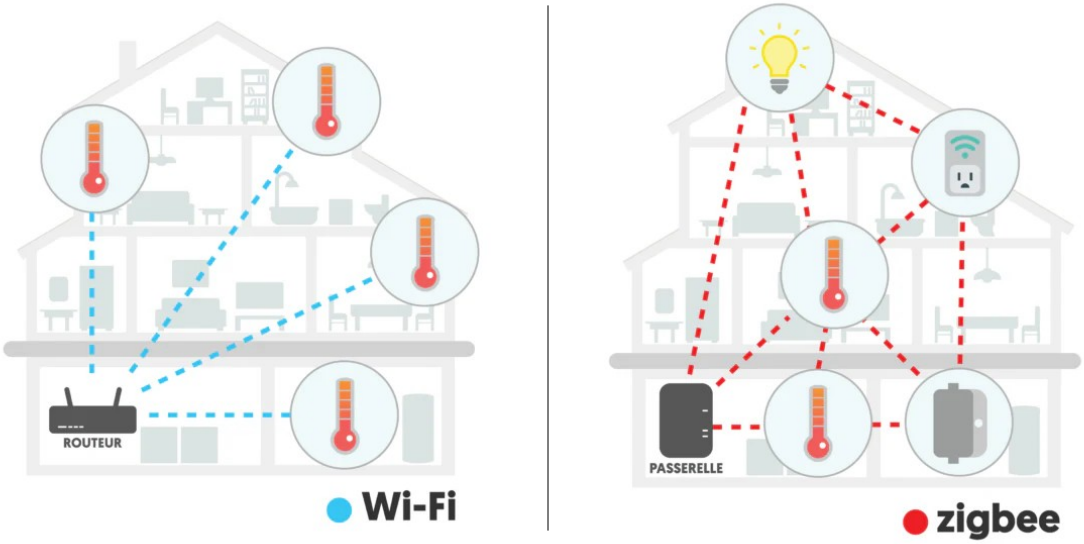

Figure 1: Comparaison entre une architecture Wi-Fi et une architecture Zigbee dans une maison connectée [1]

La figure 1 illustre la différence fondamentale entre une architecture Wi-Fi classique et un réseau maillé Zigbee. À gauche, dans le cas du Wi-Fi, tous les capteurs de température communiquent directement avec un routeur central. Cette configuration dépend fortement de ce point d'accès unique : si la connexion au routeur est interrompue, les communications s'arrêtent. À droite, le réseau Zigbee montre un fonctionnement distribué : les différents objets connectés (ampoule, capteurs, prise, etc.) échangent des données entre eux grâce à une passerelle, mais aussi via des connexions entre appareils. Cette topologie maillée permet aux messages de rebondir d'un nœud à l'autre, assurant une plus grande fiabilité et une meilleure portée du réseau.

Thread : un protocole moderne et interopérable

Thread repose sur la même base radio que Zigbee (la norme IEEE 802.15.4), mais il a été conçu plus récemment pour répondre à de nouveaux besoins. Il se distingue notamment par l'utilisation du protocole IPv6*, le même que celui utilisé sur Internet. Grâce à cela, chaque appareil Thread peut être identifié de manière unique, ce qui facilite l'intégration avec d'autres systèmes et l'interopérabilité entre fabricants. Thread utilise lui aussi un réseau maillé, mais se veut plus standardisé et sécurisé, en s'appuyant sur des protocoles IP bien connus.

Des protocoles puissants mais complexes à tester

Bien qu'ils soient conçus pour simplifier les communications entre objets, Zigbee et Thread sont des protocoles techniquement complexes. Ils intègrent des mécanismes de maillage, de routage, de sécurité, de gestion d'énergie, etc. Leur comportement peut varier selon la topologie du réseau, la qualité du signal ou le nombre d'appareils connectés. Sans outils adaptés, il devient difficile de comprendre comment le réseau réagit réellement dans certaines situations (déconnexions, délais de transmission, etc.). C'est pourquoi des infrastructures de test spécialisées sont nécessaires pour les analyser en conditions réalistes.

II.5. Objectifs du projet de banc de test

Le projet de banc de test a pour objectif de fournir un environnement de test automatisé et reproductible pour évaluer les protocoles radio basés sur IEEE 802.15.4, en particulier Zigbee et Thread. Ces protocoles sont au cœur des communications des produits domotiques de Schneider Electric. Afin de garantir leur robustesse, leur efficacité et leur pertinence pour les futurs produits, il est essentiel de pouvoir les tester dans des conditions variées et contrôlées.

Un besoin né d'expériences passées

Avant le développement du banc de test, certaines expérimentations ont été menées manuellement par l'équipe WCT. L'une d'entre elles consistait à comparer l'utilisation de deux puces radio distinctes (une pour Zigbee, une pour Thread) avec une solution de type CMP (Concurrent MultiProtocol) proposée par Silicon Labs.

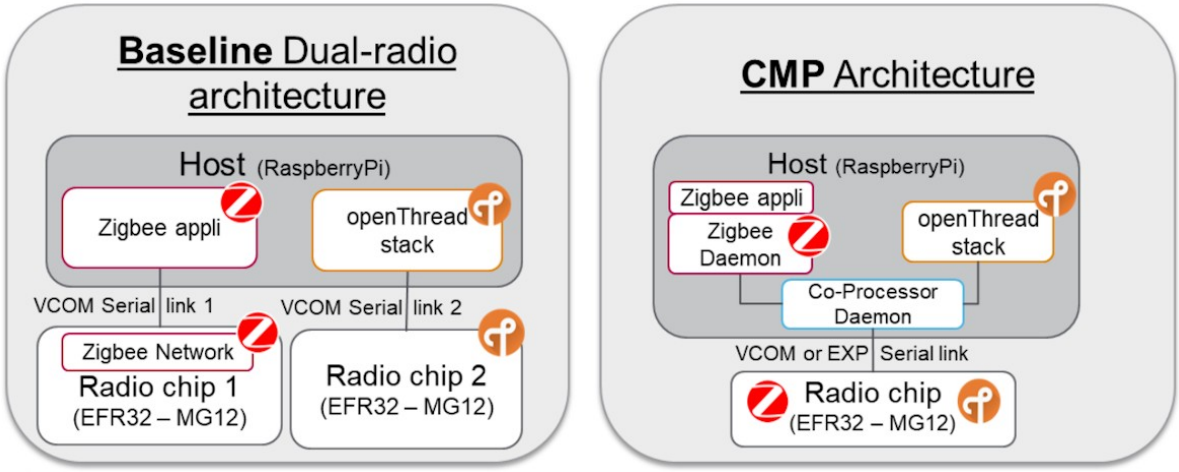

Figure 2: Comparaison entre l'architecture « baseline » à double radio et l'architecture « CMP » à radio unique [2].

La figure 2 illustre deux approches matérielles pour faire coexister les protocoles Zigbee et Thread dans un système embarqué :

- À gauche l'utilisation classique :
    - Le Raspberry Pi hôte exécute séparément l'application Zigbee et OpenThread*, chacune communiquant avec sa propre puce radio.
    - Deux émetteurs distincts sont donc nécessaires : un dédié à Zigbee et un autre à Thread.

- À droite, l'utilisation que l'équipe voulait tester :
    - Une seule puce radio est utilisée pour gérer à la fois Zigbee et OpenThread. Sur le Raspberry Pi, un petit programme spécial (appelé démon CMP) organise les échanges entre les deux protocoles et la puce radio, en passant par un simple câble de communication.
    - La topologie réduit le nombre de composants radio et simplifie la configuration matérielle tout en assurant la compatibilité multi-protocoles.

Le test consistait à faire circuler des données entre plusieurs appareils Thread, sous forme de messages UDP* de différentes tailles et fréquences, simulant par exemple des communications entre objets connectés ou des mises à jour logicielles. En parallèle, des appareils Zigbee échangeaient aussi des données, telles que des relevés périodiques ou des demandes d'état, représentatifs d'un usage domotique classique. L'équipe augmentait progressivement la charge réseau, en intensifiant les échanges sur les deux protocoles, afin d'observer leur impact sur les performances globales. Ces tests ont mis en évidence des différences marquées de comportement selon l'architecture matérielle utilisée. Toutefois, leur exécution manuelle limitait leur reproductibilité, ce qui a mis en lumière le besoin d'un outil structuré, scriptable et réutilisable : le banc de test.

Objectifs techniques

Le banc de test vise à permettre l'exécution de scénarios réseau automatisés à l'aide de scripts Python*. Ces scénarios peuvent inclure des actions telles que :

• rejoindre un réseau,

• échanger des paquets de données,

• injecter du trafic,

• simuler des déconnexions/reconnexions,

• mesurer la performance sous contrainte de charge, etc.

Les tests permettent d'évaluer différents critères de qualité des protocoles domotiques, selon les objectifs spécifiques de chaque scénario. À titre d'exemples, on peut citer :

• l'efficacité, via la mesure de la latence d'appairage* ou du débit de communication

• le passage à l'échelle, en augmentant progressivement le nombre d'appareils connectés

• la résilience, en analysant la robustesse du réseau face aux pertes de paquets ou aux congestions

• la stabilité, par l'observation du comportement sur de longues durées

• l'interopérabilité, en testant différents matériels et piles logicielles (Silicon Labs, Espressif, Nordic, etc.)

Cette liste n'est pas exhaustive : la conception modulaire et flexible du banc permet d'adapter les tests à une grande variété de cas d'usage. Il est ainsi possible de combiner

plusieurs critères au sein d'un même scénario, ou d'ajouter de nouveaux types de mesures à mesure que les besoins évoluent.

Les scripts sont stockés localement sur la machine du testeur et peuvent être relancés à tout moment, garantissant la reproductibilité des expériences. Cela facilite notamment la comparaison de différentes versions d'un même protocole ou de différents matériels dans des conditions identiques.

Un outil mutualisable

Au-delà de son usage par l'équipe WCT, le banc de test a été conçu pour être réutilisable par d'autres équipes. Il peut par exemple être utile pour les équipes qui conçoivent des produits. Un cas concret pourrait être de tester une passerelle Zigbee en simulant l'appairage d'un grand nombre d'appareils à celle-ci. Ce type de test, fastidieux et difficile à mettre en œuvre manuellement, devient accessible en quelques lignes de script.

Enfin, un des objectifs majeurs du projet est de gagner du temps lors des phases de test et de standardiser les scénarios d'évaluation.

II.6. Vue d'ensemble du banc de test

Infrastructure matérielle

Le banc s'appuie sur un serveur central et un parc de micro-ordinateurs Raspberry Pi. Ces petits ordinateurs à bas coût sont couramment utilisés dans des contextes éducatifs, industriels ou domotiques. Dans notre cas, ils jouent le rôle de nœuds de test : chaque Raspberry Pi peut accueillir jusqu'à quatre dongles radio* branchés en USB.

Un dongle radio est un petit périphérique permettant d'émettre et de recevoir des communications sans fil selon un protocole donné. Deux types de dongles sont utilisés dans le banc :

- **MG13** : compatibles avec le protocole Zigbee
- **MG24** : compatibles avec le protocole Thread

Les Raspberry Pi et le serveur sont interconnectés via un ensemble de commutateurs réseau (switchs Ethernet). Cette infrastructure filaire garantit une communication stable entre les différents éléments du banc tout en isolant les communications radio testées, afin d'éviter les interférences ou perturbations non maîtrisées.

Aujourd'hui, deux bancs de test de ce type sont déployés chez Schneider Electric :

- Un petit banc de test comprenant environ 20 Raspberry Pi.
- Un grand banc de test comprenant environ 100 Raspberry Pi.

Des photos de ces bancs sont disponibles en annexe A.

Infrastructure logicielle

Le banc de test repose sur une plateforme logicielle appelée WalT*, installée sur le serveur central. WalT est un outil open-source conçu pour piloter à distance un ensemble de machines physiques ou virtuelles dans le cadre de tests réseau ou système. Il permet de gérer efficacement un parc de Raspberry Pi tout en assurant une cohérence dans les tests effectués.

Parmi ses principales fonctionnalités, on retrouve :

- le déploiement d’images système personnalisées sur les Raspberry Pi.
- la prise de contrôle à distance des Raspberry Pi (accès terminal, redémarrage, configuration réseau),
- la collecte des journaux d’exécution, utiles pour analyser le comportement des tests,
- et l’automatisation des expériences via des scripts, pour permettre une exécution centralisée, répétable et fiable.

Cette architecture permet ainsi de gérer à grande échelle l'ensemble du banc de test, du déploiement initial aux expérimentations finales.

L’image système embarquée sur chaque Raspberry Pi, ainsi que le fonctionnement détaillé des tests, sont abordés dans la section suivante.

III. Architecture générale du projet

III.1. Architecture globale du banc de test

Le banc de test conçu par l'équipe repose sur une architecture modulaire et automatisée, permettant à des utilisateurs d'écrire des scripts pour émuler des réseaux domotiques complexes, en s'appuyant sur des Raspberry Pi équipés de dongles radio.

L'architecture peut être divisée en deux grandes briques logicielles :

- l'image multitool, installé sur chaque Raspberry Pi, qui gère la détection, la configuration et l'utilisation des dongles,
- le framework de test, utilisé côté client, qui interagit avec multitool pour piloter les tests via une interface Python.

Fonctionnement global

La figure 3 ci-dessous illustre le fonctionnement global :

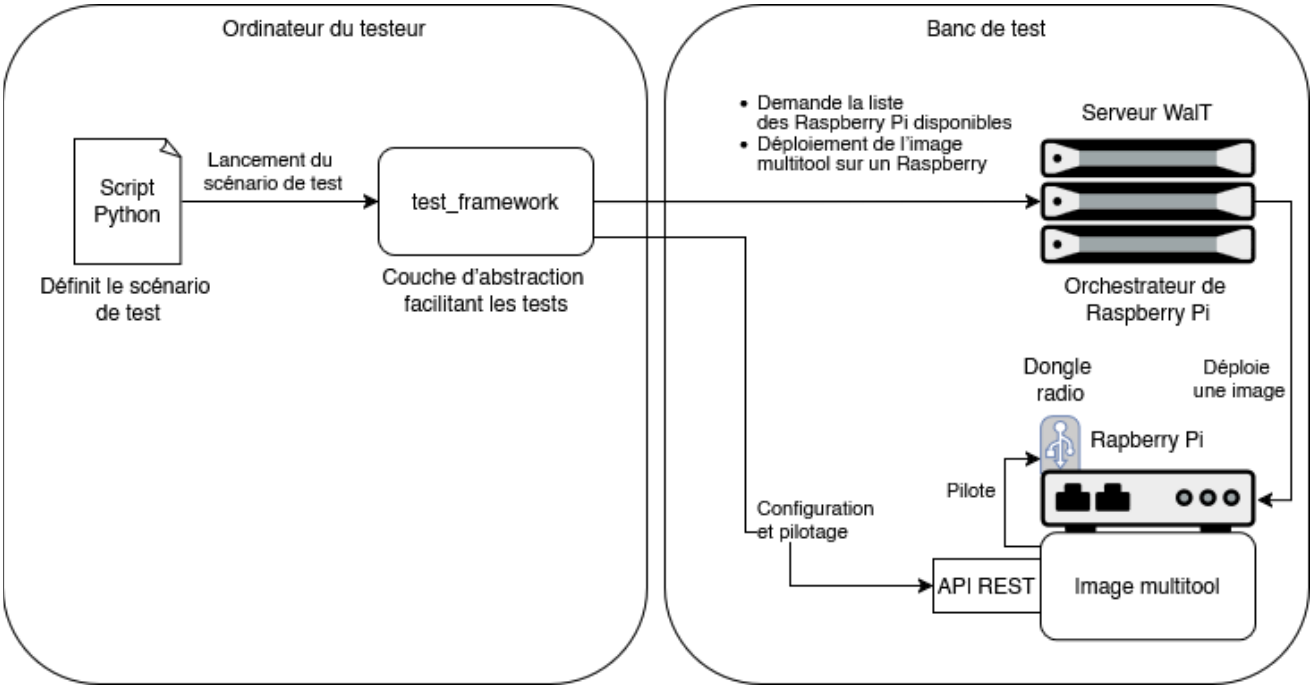

Figure 3: Schéma des interactions entre les composants logiciels et matériels

Tout commence côté utilisateur. La personne qui souhaite lancer un test écrit un script en Python, en utilisant le framework de test (test_framework), qui agit comme une couche d'abstraction. Ce framework dialogue avec la plateforme WalT pour obtenir la liste des Raspberry Pi disponibles, puis déploie l'image multitool sur un Raspberry.

Une fois le Raspberry démarré, test_framework interagit avec lui à travers l'API* REST* exposée par multitool. Cette API permet de configurer les dongles branchés, de lancer les outils adaptés, et de simuler des comportements domotiques.

Du point de vue logiciel, chaque Raspberry multitool est donc une sorte de "boîte noire" spécialisée, capable d'exécuter des commandes sur ses dongles, de téléverser un

micrologiciel, de les redémarrer, ou de lancer un utilitaire adapté selon le protocole utilisé (Zigbee ou Thread).

Détail des deux briques principales

1. L'image multitool

Multitool est une image système personnalisée, conçue pour être déployée sur n'importe quel Raspberry Pi via WalT. Elle automatise toutes les tâches liées à la gestion des dongles :

- détection automatique,
- identification du type de puce (MG13 ou MG24),
- détection du micrologiciel,
- passage en mode bootloader si nécessaire,
- téléversement automatique du micrologiciel,
- démarrage des utilitaires (Thread ou Zigbee).

Ce fonctionnement est encapsulé derrière une API REST développée avec FastAPI*, qui expose différents endpoints pour interagir avec les dongles. Aucun appel système complexe n'est nécessaire côté utilisateur : l'ensemble est encapsulé dans des appels Python simples, faits par le framework de test.

2. Le framework de test

Le framework de test a été développé par un membre de l'équipe en parallèle de multitool. Il s'agit d'une bibliothèque Python fournie aux utilisateurs finaux, leur permettant d'écrire des scripts de test sans avoir à se soucier des appels HTTP, de la configuration des dongles ou de la gestion des Raspberry.

Ce framework prend en charge :

- la découverte et le déploiement des Raspberry via WalT,
- les appels à l'API de multitool (configuration des dongles, démarrage des utilitaires),
- la coordination des tests (création de réseaux, envoi de messages, simulation de capteurs, etc.),
- la récupération des données pour analyse.

Cette séparation claire entre les responsabilités de multitool (exécution côté Raspberry) et du framework de test (pilotage côté utilisateur) permet de garder une architecture souple, modulaire, et facilement maintenable.

III.2. Contraintes et exigences de conception

Le projet devait répondre à un ensemble de contraintes, mais aussi à plusieurs exigences de conception destinées à garantir la flexibilité, la robustesse et l'évolutivité du système.

Contraintes

- Langage Python : le développement a été réalisé en Python, conformément aux choix technologiques de l'équipe. Cela a permis une intégration rapide avec les bibliothèques existantes et une bonne lisibilité du code.
- Plateforme WalT : bien que choisie pour sa pertinence (déploiement d'images, gestion centralisée), WalT était encore en développement actif. Certaines fonctionnalités utilisées étaient en version bêta, ce qui a parfois impliqué des contournements liés à des bugs.
- Automatisation nécessaire : le système devait permettre de lancer des tests sans intervention manuelle, si ce n'est l'exécution d'un script Python. Il était donc impératif que tous les composants puissent être initialisés, contrôlés et surveillés automatiquement.
- Contraintes réseau et infrastructure Schneider : travaillant sur le réseau interne Schneider et depuis des postes PC Schneider, nous avons dû composer avec les règles strictes de la DSI (Direction des Systèmes d'Information, responsable de la gestion et de la sécurité des infrastructures informatiques internes). Ces règles imposent notamment des proxys et des filtres qui peuvent perturber les communications réseau, nécessitant souvent des recherches approfondies et des contournements pour identifier l'origine de comportements inattendus. Cette contrainte a complexifié la mise au point des outils réseau et l'interaction avec les API.

Exigences non fonctionnelles

Modularité

Tous les Raspberry Pi sont configurés de manière identique dans WalT et peuvent être utilisés de manière interchangeable. Seuls les dongles radio varient selon les besoins. L'image système déployée, appelée multitool, détecte automatiquement les dongles connectés et s'y adapte dynamiquement.

Robustesse

Le système intègre plusieurs mécanismes pour gérer les erreurs matérielles :

- Signalement des échecs de détection dans les logs.
- Réponses d'erreur détaillées via l'API.
- Redémarrage logiciel des ports USB en cas de dysfonctionnement.

Simplicité d'usage

La bibliothèque Python *test_framework* permet aux testeurs de lancer des scripts sans avoir à gérer les détails du matériel ou du réseau. Elle prend en charge la détection des Raspberry Pi, le déploiement des images via WalT et la communication avec les API exposées.

Détection automatique et adaptation dynamique

Lorsqu'un test requiert un certain rôle ou protocole, le système téléverse automatiquement le micrologiciel adapté sur le dongle sélectionné, sans intervention humaine.

Interopérabilité Zigbee / Thread

L'image multitool embarque les outils nécessaires aux deux protocoles, évitant ainsi la multiplication des environnements et facilitant la maintenance.

Évolutivité

L'ensemble du système a été conçu pour être facilement extensible. De nouveaux types de tests, de matériels ou de protocoles pourront être intégrés au fil des évolutions du projet.

IV. Conception et développement de l'image multitool

IV.1. Objectifs fonctionnels de l'image multitool

L'image multitool est le cœur du projet. Elle est née du besoin de simplifier et d'automatiser la gestion des tests à grande échelle, tout en réduisant au maximum les interventions manuelles. L'objectif principal est de proposer une infrastructure reproductible et robuste, capable de détecter, configurer et piloter des dongles de test de manière autonome et transparente pour l'utilisateur.

Finalité et usage de l'image

L'image est construite à partir d'une base Debian, choisie pour sa stabilité, sa compatibilité avec l'écosystème Raspberry Pi et la richesse de ses paquets disponibles. Elle est pensée pour être directement opérationnelle : tous les outils nécessaires sont préinstallés et accessibles localement, et un service *systemd* lance automatiquement le script principal au démarrage. Un utilisateur distant (via le framework *test_framework*, par exemple) peut ainsi demander de simuler un appareil, sans se soucier de l'état matériel réel du dongle.

À partir de la requête /sys/start, multitool :

- choisit automatiquement le micrologiciel adapté,
- téléverse ce micrologiciel dans le dongle,
- lance l'utilitaire permettant de dialoguer avec le micrologiciel,
- et expose les commandes nécessaires via de nouveaux endpoints API.

L'approche adoptée permet de garantir simplicité, reproductibilité et compatibilité, tout en laissant la logique métier (choix des outils, orchestration des tests) au niveau supérieur, dans le framework Python.

IV.2. Démarrage et détection automatique

Au démarrage de la Raspberry Pi, un service *systemd* est automatiquement lancé. Ce service exécute un script Python principal, responsable d'initialiser la détection des dongles connectés, de préparer leur utilisation et de démarrer le serveur d'API. Ce script s'appuie sur un package développé spécifiquement pour ce projet : *dongle_manager*.

Ce module a été conçu pour effectuer la détection de manière robuste, même lorsque les dongles sont dans des états différents (applicatif ou bootloader). Le diagramme de séquence de la figure 4 illustre cette séquence de démarrage :

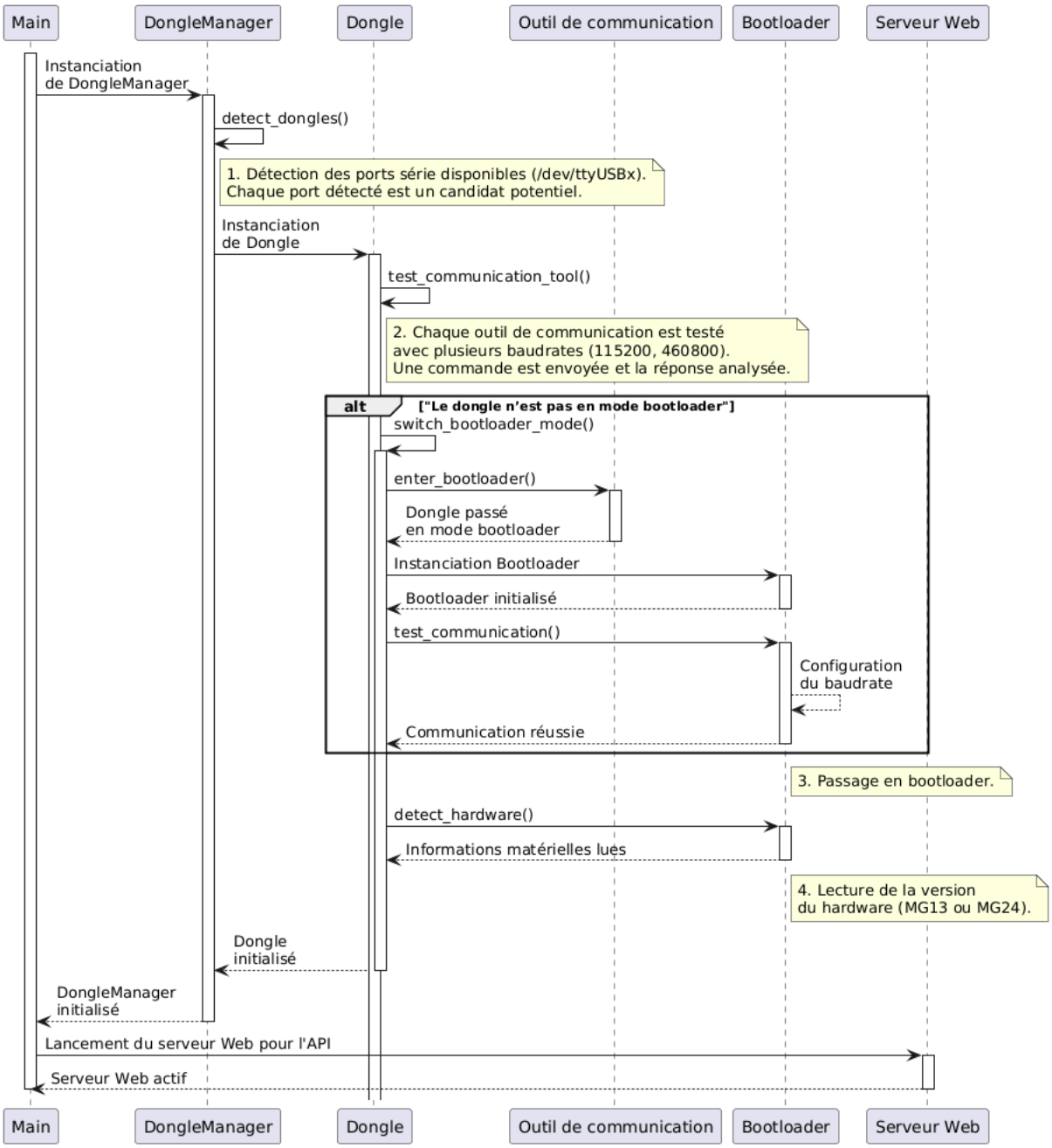

Figure 4: Diagramme de séquence du démarrage de multitool

1. Détection des ports série disponibles

Le script commence par scanner les périphériques série disponibles, typiquement exposés sous la forme de fichiers /dev/ttyUSBx sur les systèmes Linux. Chaque port détecté est alors considéré comme un candidat potentiel, c'est-à-dire un dongle qui pourrait être utilisé pour les tests.

2. Essais successifs des utilitaires

Pour chaque port, dongle_manager tente d'utiliser successivement les différents utilitaires de communication embarqués dans l'image multitool. Ces utilitaires sont conçus pour communiquer avec les dongles soit en mode applicatif, soit en mode bootloader.

Chaque utilitaire est testé avec plusieurs valeurs de débit (baudrate*), principalement 115200 et 460800, car les micrologiciels utilisés dans le projet se basent sur l'un de ces deux réglages. Une commande caractéristique est envoyée, et la réponse est analysée pour déterminer si la communication est réussie ou non.

3. Passage en bootloader

Si un utilitaire parvient à communiquer avec un dongle, celui-ci est alors mis en mode bootloader (sauf s'il l'est déjà). Le passage en bootloader est nécessaire pour garantir que le dongle puisse être reprogrammé ou identifié de manière fiable. Ce passage utilise des commandes spécifiques à chaque utilitaire.

4. Lecture des informations matérielles

Une fois en bootloader, le script interroge le dongle pour obtenir sa version. Le dernier chiffre de cette version encode le type de matériel (par exemple, 04 pour MG13, 09 pour MG24). Cette information est essentielle pour déterminer quel firmware devra être utilisé par la suite.

À l'issue de cette détection, tous les dongles connus et valides sont dans un état initialisé, prêts à être utilisés via l'API. Les dongles non reconnus ou mal configurés sont ignorés, mais leur statut est consigné dans les logs pour information.

Après cette phase, le système est en attente d'une commande utilisateur (par exemple via l'API) pour téléverser un micrologiciel et lancer un outil sur un dongle donné.

IV.3. Gestion de l'API

L'interface entre le framework de test et les Raspberry Pi est assurée par une API REST, développée avec FastAPI et servie en local sur le port 8000 de chaque Raspberry Pi. Cette API joue un rôle fondamental dans la simplification de l'utilisation des dongles depuis les scripts de test, en masquant toute la complexité liée à leur détection, leur configuration ou encore la reprogrammation des dongles.

L'accès à cette API ne se fait pas directement depuis le réseau Schneider : les Raspberry Pi sont isolés dans un réseau dédié à la plateforme WalT. Pour établir la communication, un mécanisme d'exposition de ports est mis en place sur le serveur WalT, qui possède une double interface réseau. Ce dernier expose par convention un port unique pour chaque Raspberry Pi.

Par exemple, pour la Raspberry Pi dont le nom est *FRB-01*, l'API FastAPI locale sur le port 8000 est rendue accessible via le port 8001 du serveur WalT.

IV.4. Intégration des utilitaires

Le système multitool permet d'intégrer différents utilitaires communicant avec un dongle afin de simuler des comportements spécifiques de périphériques domotiques, en particulier ceux basés sur les protocoles Thread et Zigbee. Ces utilitaires sont utilisés tels quels, sans être modifiés, afin de respecter les standards internes de leurs auteurs. Multitool ajoute une surcouche légère qui permet de les encapsuler proprement et de les exposer via une API HTTP unifiée.

Cas général – Intégration d'utilitaires Python

Le cas le plus courant est celui des outils écrits en Python, comme OTCI (OpenThread CLI), fourni par la fondation OpenThread. Ces outils sont directement importés dans le code multitool. Pour chaque utilitaire, un fichier Python dédié est créé. Ce fichier définit les endpoints API nécessaires à l'interaction avec l'outil. Concrètement, il expose un objet router (ou une fonction retournant un router) qui regroupe les routes/endpoints FastAPI associées à cet utilitaire.

Multitool utilise ces routeurs pour intégrer dynamiquement les fonctionnalités offertes par chaque outil, sous la forme d'endpoints HTTP.

Ces endpoints permettent d'exposer de façon simple les appels internes aux méthodes Python de l'outil. La surcouche ne modifie pas l'outil lui-même mais en rend les fonctionnalités accessibles via l'API multitool.

Cas particulier – Simulateurs Zigbee externes

Pour la partie Zigbee, l'intégration est plus complexe. Les simulateurs utilisés (Zigbee GW Simulator, Zigbee Pro Simulator, Zigbee GP Simulator) sont des exécutables compilés en C++ fournis par une autre équipe de Schneider (TestOps). Ils permettent de simuler différents types de périphériques Zigbee.

Pour les intégrer, comme ces simulateurs ne sont pas des bibliothèques Python, mais des programmes autonomes, on utilise la bibliothèque Python *subprocess* pour les exécuter. Une fois lancés, ces simulateurs démarrent leur propre serveur HTTP, avec une API spécifique permettant de les piloter.

Au moment de leur lancement, multitool leur passe des arguments spécifiques pour :

- Spécifier le port série du dongle à utiliser,
- Définir le port HTTP à exposer pour leur propre API.

Par défaut, ces simulateurs gèrent tous les dongles connectés et utilisent des ports fixes. Pour éviter les conflits (plusieurs outils partageant un même port ou un même dongle), multitool configure chaque instance avec un port dédié et un dongle unique.

Voici la convention actuelle des plages de ports utilisées :

- Zigbee GW Simulator : à partir de 4000
- Zigbee Pro Simulator : à partir de 5000
- Zigbee GP Simulator : à partir de 6000

Chaque lancement d'un simulateur alloue dynamiquement le prochain port libre dans la plage correspondante.

Un endpoint spécifique est prévu pour connaître le port utilisé par un simulateur :

```
/hw/{dongle_name}/{tool_name}/port
```

Cette architecture permet au framework de test client de communiquer directement avec l'API du simulateur via ce port, sans repasser par l'API exposé par multitool. Cela réduit la complexité des appels réseau et améliore les performances.

IV.5. Architecture interne du code multitool

L'infrastructure logicielle de multitool est conçue de manière modulaire afin d'assurer la maintenabilité du projet et de faciliter l'ajout de nouveaux outils ou matériels. Cette section présente l'organisation générale du code, les relations entre les composants majeurs et le rôle de chaque module, en s'appuyant sur le diagramme de classes fourni en annexe B.

Présentation modulaire

Le code est structuré autour d'un module principal appelé *dongle_manager*, qui centralise la gestion de l'ensemble des dongles connectés et la logique associée aux outils. Les autres composants sont répartis dans différents fichiers organisés en répertoires, sans pour autant former des packages Python à part entière. On y trouve notamment les routes FastAPI, les composants de sniffing (expliqués par la suite), la gestion du proxy et des utilitaires communs.

Diagramme de classes

Le diagramme en annexe B illustre les principales classes présentes dans le projet ainsi que leurs relations :

Description des composants principaux

- **DongleManager**

  Il s'agit de la classe centrale qui gère l'ensemble des dongles détectés. Elle stocke une liste de tous les objets Dongle actifs, permet leur détection, leur mise à jour (y compris téléversement de micrologiciel), et orchestre l'utilisation des outils. Toutes les opérations critiques (lancement, arrêt, vérification d'état) passent par cette classe.

- **Dongle**

  Chaque dongle physique est représenté par une instance de la classe Dongle. Elle contient son port, son nom, son état actuel (BOOTLOADER ou FIRMWARE), ainsi qu'une référence vers l'outil actuellement actif (ex. ToolOtci). C'est cette classe qui encapsule la logique de changement d'état du dongle.

- **CommunicationTool**

  Classe abstraite dont héritent tous les outils compatibles avec multitool. Elle définit l'interface minimale à respecter (enter_communication, enter_bootloader, etc.) et fournit des méthodes utilitaires communes comme la gestion du port série. Les outils concrets comme ToolOtci, ToolZabConsoleTest ou ToolZigbeeProSim héritent de cette classe.

- **Outils spécifiques**

  Chaque outil implémente la logique nécessaire à son bon fonctionnement. Par exemple, ToolOtci permet une communication bidirectionnelle avec un firmware

Thread. Ces outils sont instanciés dynamiquement dans les objets Dongle, selon la demande de l'utilisateur.

- **Sniffer et Proxy**

  Ces deux composants sont complémentaires :

  - Le Sniffer* permet la capture des trames réseau via un dongle dédié.
  - Le proxy, écoute les trames sniffées envoyées par les dongles sniffer via UDP. Il les reçoit en temps réel, les regroupe, puis les enregistre dans un fichier PCAP* pour permettre une analyse a posteriori.

- **Routers (FastAPI)**

  Les fichiers otci.py, zigbee_gp.py, etc., exposent des routes REST via FastAPI. Ces routes accèdent directement aux méthodes de DongleManager, notamment pour démarrer un outil ou en exposer un port.

Relations entre les modules

Lorsqu'un utilisateur fait une requête pour démarrer un outil (ex. via /sys/start), l'API transmet la demande à DongleManager, qui identifie le dongle ciblé, vérifie son état, et instancie l'outil demandé si le dongle est disponible. L'outil est ensuite lancé, ce qui peut inclure la communication série, le démarrage d'un exécutable, ou d'autres opérations spécifiques.

À tout moment, l'utilisateur peut arrêter l'outil via une nouvelle requête, ce qui remet le dongle dans un état neutre (BOOTLOADER) et le rend disponible pour une autre utilisation.

Le système permet donc à un même dongle de changer dynamiquement de rôle, à condition qu'il soit correctement libéré au préalable.

Les différents modules interagissent de manière fluide :

- DongleManager centralise les états et décisions,
- Les routers agissent comme interface REST vers ces décisions,
- Les outils encapsulent la logique spécifique à chaque protocole.

V. Intégration et tests

V.1. Intégration dans le framework de test

Le framework de test utilisé dans ce projet a été développé en interne par l'équipe. Il permet d'automatiser des scénarios sur le banc de test en s'appuyant sur l'API HTTP exposée par les images multitool déployées sur les Raspberry Pi. Les échanges se font via des requêtes HTTP envoyées avec la bibliothèque Python *requests*.

Ce framework repose sur une abstraction assez poussée des équipements domotiques, ce qui permet de manipuler des concepts comme un leader Thread, un sniffer, ou une passerelle Zigbee à travers des commandes simples dans les scripts de test. Lorsqu'un test demande, par exemple, la création d'un nœud Thread en tant que leader, le framework orchestre automatiquement toute la séquence : démarrage d'un multitool, allocation d'un dongle, lancement de l'utilitaire (comme OTCI), configuration, etc.

Je n'ai pas intégré multitool au framework moi-même, mais j'ai accompagné cette intégration en aidant Lucien, l'auteur principal du framework, à comprendre le fonctionnement de multitool. Je l'ai notamment aidé à identifier les bons endpoints à appeler, à comprendre les séquences de démarrage, et à déboguer certains comportements pendant l'intégration.

Par ailleurs, j'ai contribué à faire évoluer l'API multitool pour faciliter son utilisation côté framework. J'ai par exemple :

- ajouté un endpoint pour forcer la détection des dongles,
- mis en place une route de réinitialisation globale,
- et harmonisé les réponses des endpoints pour simplifier leur traitement côté framework.

V.2. Tests réalisés et retours

La phase de tests automatisés a commencé récemment. Nous sommes encore dans une phase de mise en place, et ces premiers scripts de tests facilitent l'identification et la correction des dysfonctionnements au sein du projet.

J'ai rédigé un premier scénario de test autour de Thread, avec la simulation de deux appareils. Le script configure les rôles réseau, démarre un sniffer, lance une capture, puis envoie des requêtes *ping* entre les deux nœuds pour vérifier la connectivité. Les trames sont ensuite récupérées dans un fichier PCAP pour analyse.

Un second test est en cours d'écriture pour simuler l'appairage d'une centaine de dispositifs Zigbee à une passerelle. D'autres scénarios suivront, au fur et à mesure de la stabilisation du système.

Cette première phase a déjà permis d'identifier plusieurs problèmes :

- Un comportement étrange sur Thread, avec plus de paquets reçus que de paquets envoyés lors d'un ping. Cela vient du mécanisme de retransmission automatique de

Thread au niveau MAC* : en cas d'absence d'ACK* (confirme que le paquet est bien arrivé) , un paquet peut être renvoyé, même s'il a déjà été bien reçu.

- Un bug dans test_framework provoquait des erreurs 404 quand des requêtes étaient envoyées juste après le lancement d'un utilitaire. Les endpoints associés n'étaient pas encore disponibles. J'ai corrigé ce bug en ajoutant un mécanisme de réessaie avec temporisation, ce qui évite les crashs liés à un démarrage trop rapide.

VI. Collaboration et évolutions

VI.1. Collaboration avec l'équipe WCT

Les échanges avec l'équipe WCT se sont déroulés principalement à distance, via Microsoft Teams, mais j'ai également eu l'occasion de les rencontrer en personne et de visiter leurs locaux à trois reprises. Les communications étaient régulières, avec des messages presque quotidiens et une réunion hebdomadaire en visioconférence pour faire le point sur l'avancement.

Le recueil des besoins s'est fait de façon continue, sans phase de spécifications figée. De nouvelles attentes et contraintes techniques sont apparues au fil du projet, souvent discutées et ajustées lors des réunions hebdomadaires. L'architecture a donc évolué par itérations successives.

Lors des points d'équipe, des revues informelles ont permis d'intégrer des retours et d'adapter la structure du code. J'ai pu aussi proposer des solutions techniques au fur et à mesure de ma compréhension du projet. Par exemple, j'ai suggéré une organisation plus claire de la communication avec l'API multitool dans l'environnement test_framework.

Certaines décisions techniques ont été motivées par des imprévus. Par exemple, le firmware Thread initialement prévu et son outil associé ont été abandonnés rapidement, l'outil ayant été archivé sur GitHub et n'étant plus à jour. J'ai donc dû réorienter mon travail vers une autre solution, impliquant une reprise importante du développement.

J'ai également reçu des retours sur des aspects plus précis, comme le nommage des endpoints API, pour améliorer leur lisibilité. La complexité de l'architecture m'a conduit à produire de la documentation complémentaire (diagrammes de classes et de séquence) pour faciliter la compréhension du code par l'équipe.

Enfin, lors de tests à plus grande échelle, des retours fonctionnels et techniques sont apparus. Par exemple, un bug survenu lors de la reprogrammation simultané d'une centaine de dongles a été détecté. Certains dongles restaient bloqués en mode bootloader après flash. L'analyse a montré que le problème venait du dongle lui-même qui était instable de manière aléatoire. J'ai mis en place un correctif qui réinitialise l'alimentation USB du Raspberry Pi pour redémarrer proprement les dongles défaillants.

VI.2. Travail d'équipe et gestion de projet

Le projet a suivi un cycle de développement itératif et incrémental, adapté aux besoins de l'équipe et au contexte non critique du banc de test. Chaque incrément apportait une évolution fonctionnelle : ajout d'un nouveau module, amélioration de la robustesse, ou intégration d'un outil de test. Le découpage du travail n'était pas strictement formalisé, mais il suivait une logique progressive, avec des jalons techniques identifiables.

Organisation du projet

Un tableau Kanban GitHub permettait de suivre l'ensemble des tâches et des tickets (bugs, évolutions, idées) pour tous les sous-projets liés au banc de test. Chaque développeur

gérait ses propres cartes, et les tâches passaient par les colonnes classiques : To do, In progress, Review, Done.

Les développements étaient organisés autour de branches Git par fonctionnalité ou correction. À la fin d'une tâche, une pull request était ouverte et soumise à relecture (par un ou deux collègues), permettant d'échanger avant la fusion dans la branche principale. En moyenne, deux itérations suffisaient à valider une contribution.

Une réunion hebdomadaire permettait de faire le point sur les avancées de chacun, de réévaluer les priorités et de lever les éventuels blocages. Le reste des échanges techniques se faisait de manière asynchrone via GitHub ou en discussion directe.

Planning personnel

Même s'il n'y avait pas de planning global figé, j'ai pu organiser mes propres tâches selon une progression logique tout au long de l'année :

- Septembre à novembre
  → Découverte de l'environnement de travail (GitHub, WalT, outils internes)
  → Création d'une première image Raspberry Pi pour l'utilisation d'un sniffer Zigbee

- Décembre à février
  → Début du développement de l'image multitool
  → Mise en place de l'architecture logicielle : gestion des outils, API, configuration système

- Février à mars
  → Développement du package dongle_manager, d'abord pour gérer proprement les dongles dans multitool
  → Pendant les réunions, on identifie que ce module est aussi utile pour d'autres cas, notamment pour un collègue partant installer un banc de tests en Inde
  → Le développement est donc accéléré et finalisé en priorité pour ce besoin

- Mars à mai
  → Finalisation de multitool avec détection automatique des dongles, gestion du flash, API fonctionnelle

- Juin
  → Début de l'utilisation réelle du banc de test : rédaction de scripts, identification et correction de bugs

Autonomie et échanges

J'ai eu beaucoup d'autonomie tout au long du projet, notamment sur la partie multitool que j'ai entièrement conçue et développée. J'ai aussi pris l'initiative de découper certaines fonctions en modules réutilisables, comme le dongle_manager. Cette autonomie n'empêchait pas les échanges réguliers avec l'équipe, surtout lors des relectures ou des réunions hebdo. Les retours m'ont permis d'ajuster certaines parties et d'améliorer la qualité globale du travail.

VII. Impact du projet

VII.1. Impact technique

Le projet a grandement facilité la mise en place de tests liés aux protocoles radio Zigbee et Thread. L'uniformisation des Raspberry Pi à l'aide d'images système standardisées, notamment par l'image multitool, qui simplifie les interventions techniques et réduit les risques d'erreurs liés à des configurations variables.

L'automatisation de nombreuses tâches (détection des dongles, téléversement des micrologiciels, lancement des utilitaires) permet de gagner un temps précieux lors de la préparation des campagnes de test. Par exemple, un test d'appairage de 100 nœuds Zigbee, qui demandait auparavant plusieurs heures de configuration manuelle, peut désormais être lancé en quelques secondes via un script Python. Cette automatisation améliore également la reproductibilité des tests, ce qui est essentiel pour obtenir des résultats comparables et fiables.

VII.2. Impact stratégique pour Schneider

Le banc de test s'inscrit dans la démarche d'innovation de Schneider Electric en fournissant un outil performant pour évaluer les protocoles Zigbee et Thread dans des conditions proches de l'usage réel. Il permet d'identifier en amont les forces et limites de chaque protocole, facilitant ainsi les choix d'intégration dans les futurs produits.

Grâce à l'automatisation, le recours aux tests manuels diminue, ce qui réduit la charge de travail, les coûts de développement, et les risques d'erreurs non détectées. Cela se traduit concrètement par une meilleure qualité produit et moins de retours clients.

VII.3. Impact environnemental et sociétal

En facilitant l'évaluation de protocoles radio plus efficaces, le projet contribue indirectement à améliorer la performance énergétique des objets communicants. À grande échelle, cela permet de limiter leur consommation électrique et donc leur impact environnemental.

D'un point de vue sociétal, la réduction des retours en service après-vente permet de limiter le gaspillage de matériel, les transports liés aux remplacements, et d'allonger la durée de vie des produits.

VIII. Conclusion

Dans le cadre de cette alternance, j'ai été amené à contribuer à un projet stratégique pour Schneider Electric, consistant à concevoir un banc de test automatisé pour l'évaluation des protocoles de communication Zigbee et Thread. Ce projet, mené en collaboration avec la Wireless Connectivity Team, avait pour objectif de fournir une infrastructure flexible, reproductible et capable de simuler des scénarios réalistes de communication entre objets connectés.

Sur le plan technique, mon travail s'est principalement concentré sur le développement de l'image système « multitool » utilisée sur les Raspberry Pi du banc de test. Cette image intègre notamment un mécanisme de détection automatique des dongles radio, la gestion du téléversement des micrologiciels, ainsi qu'une API permettant de piloter les tests à distance. J'ai également accompagné l'intégration de cet outil dans le framework de test existant, en contribuant à l'adaptation de l'interface et en facilitant l'utilisation des fonctionnalités depuis les scripts d'expérimentation. Ces développements ont été concrètement exploités lors des premières campagnes de tests.

Ce projet a représenté une expérience formatrice, tant sur le plan technique que professionnel. J'ai pu évoluer en autonomie sur des tâches complexes, tout en collaborant étroitement avec une équipe externe via des échanges réguliers. Cette collaboration à distance m'a permis de développer mes compétences en communication et en organisation, et j'ai également pu constater l'importance des rencontres sur site pour renforcer la cohésion et l'efficacité du travail en équipe. Progressivement, j'ai pris confiance dans ma capacité à intervenir dans un environnement professionnel exigeant, à participer activement aux réunions et à proposer des solutions.

Le projet entre désormais dans une phase d'amélioration continue, avec des perspectives d'évolution pour en renforcer la stabilité, la facilité d'utilisation et les fonctionnalités disponibles. La passation est en cours avec un collègue d'Elyxoft afin d'assurer la continuité du développement à mon départ. Cette mission a renforcé mon intérêt pour les systèmes embarqués et les objets communicants. Elle m'a donné envie de poursuivre dans cette voie en approfondissant mes compétences techniques, notamment dans des environnements encore plus proches du matériel, ce que je pourrai explorer à travers ma formation à l'Esisar et ma future alternance chez STMicroelectronics.

Glossaire

ACK (Acknowledgment) : Message envoyé par un récepteur pour confirmer la bonne réception d'un paquet de données.

API (Application Programming Interface) : Interface de programmation permettant à deux logiciels de communiquer entre eux. Dans le projet, une API REST est utilisée pour piloter les Raspberry Pi à distance via des requêtes HTTP.

Appairage : Processus d'association entre deux appareils sans fil (par ex. une ampoule Zigbee et une passerelle), leur permettant de communiquer.

Baudrate : Vitesse de transmission des données sur une liaison série, mesurée en bauds. Des valeurs typiques comme 115200 ou 460800 sont utilisées pour configurer les communications avec les dongles.

Bootloader : Petit programme présent sur un microcontrôleur, chargé de permettre l'installation ou la mise à jour du micrologiciel (firmware). Le bootloader est une étape nécessaire avant de faire fonctionner l'appareil en mode "applicatif".

Dongle radio : Petit périphérique USB permettant d'émettre et de recevoir des communications sans fil selon un protocole donné (Zigbee ou Thread).

FastAPI : Framework web Python moderne utilisé pour créer l'API REST côté Raspberry Pi.

Firmware (Micrologiciel) : Programme informatique embarqué dans un microcontrôleur, responsable du fonctionnement de l'appareil. Il est stocké en mémoire et s'exécute automatiquement au démarrage.

IEEE 802.15.4 : Norme de communication radio utilisée comme couche physique par les protocoles Zigbee et Thread. Elle définit notamment la fréquence (2,4 GHz) et le débit maximal (250 kbps).

Image système : Version préconfigurée d'un système d'exploitation, adaptée à un usage spécifique (ici, multitool sur Raspberry Pi).

IPv6 : Version récente du protocole IP permettant d'identifier les appareils sur un réseau. Utilisé par Thread pour garantir une meilleure connectivité et standardisation.

MAC (Medium Access Control) : Sous-couche de la couche liaison du modèle OSI. Elle gère l'accès au média de communication, la détection de collisions, l'adressage des trames et la retransmission en cas d'erreur.

Micrologiciel : *Voir Firmware.*

OpenThread : Implémentation open-source du protocole Thread développée par Google. Elle inclut un outil en ligne de commande (OTCI) utilisé dans les tests.

PCAP : Fichier de capture réseau contenant l'ensemble des trames échangées pendant un test. Utilisé pour l'analyse après coup (post-mortem).

Protocole de communication : Ensemble de règles définissant comment deux appareils doivent échanger des informations (Zigbee, Thread, HTTP, etc.).

Python : Langage de programmation utilisé pour le développement du framework de test, des scripts et de l'API multitool.

REST (Representational State Transfer) : Style d'architecture pour les services web. L'API REST exposée permet d'interagir avec les Raspberry Pi via des URL et des méthodes HTTP standard (GET, POST, etc.).

Raspberry Pi : Micro-ordinateur à bas coût, utilisé ici comme nœud de test. Il pilote les dongles et exécute les scénarios réseau via l'image multitool.

Réseau maillé (mesh) : Type de réseau où chaque appareil peut relayer les messages pour les autres. Ce type d'architecture améliore la portée et la robustesse des communications (utilisé dans Zigbee et Thread).

Sniffer : Dispositif ou logiciel permettant de "sniffer" (capter) les trames d'un réseau sans les modifier. Utilisé pour observer les échanges radio lors des tests.

Thread : Protocole de communication sans fil basé sur IEEE 802.15.4, utilisant IPv6. Il est conçu pour les objets connectés et les réseaux domotiques.

UDP (User Datagram Protocol) : Protocole de communication rapide mais non fiable (sans vérification d'arrivée), souvent utilisé pour les tests de performance ou les échanges de données entre objets.

WalT : Plateforme open-source permettant de piloter des machines physiques (ici des Raspberry Pi) dans un environnement de test réseau.

Zigbee : Protocole sans fil à faible consommation d'énergie, utilisé pour la domotique (ampoules, capteurs, volets…). Basé sur IEEE 802.15.4 et organisé en réseau maillé.

Sitographie

[1] Sinopé Technologies. Disponible sur : https://www.sinopetech.com/. Consulté en juin 2025.

[2] Équipe WCT. *When processing time exceeds applicative timeouts: Concurrent Zigbee/Thread stack evaluation*. Disponible sur : https://ieeexplore.ieee.org/document/10817417. Consulté en juin 2025.

[3] Site de l'entreprise Elyxoft. Disponible sur : https://www.elyxoft.fr/ Consulté en juin 2025.

[4] OpenAI, ChatGPT**.** Assistance rédactionnelle et suggestions pour l'amélioration des formulations et la relecture. Contenus adaptés et intégrés dans le rapport. Disponible sur : https://chat.openai.com. Consulté en juin 2025.

Annexe A : Photos des bancs de test

Figure 5: Photo du petit banc de test (20 Raspberry Pi)

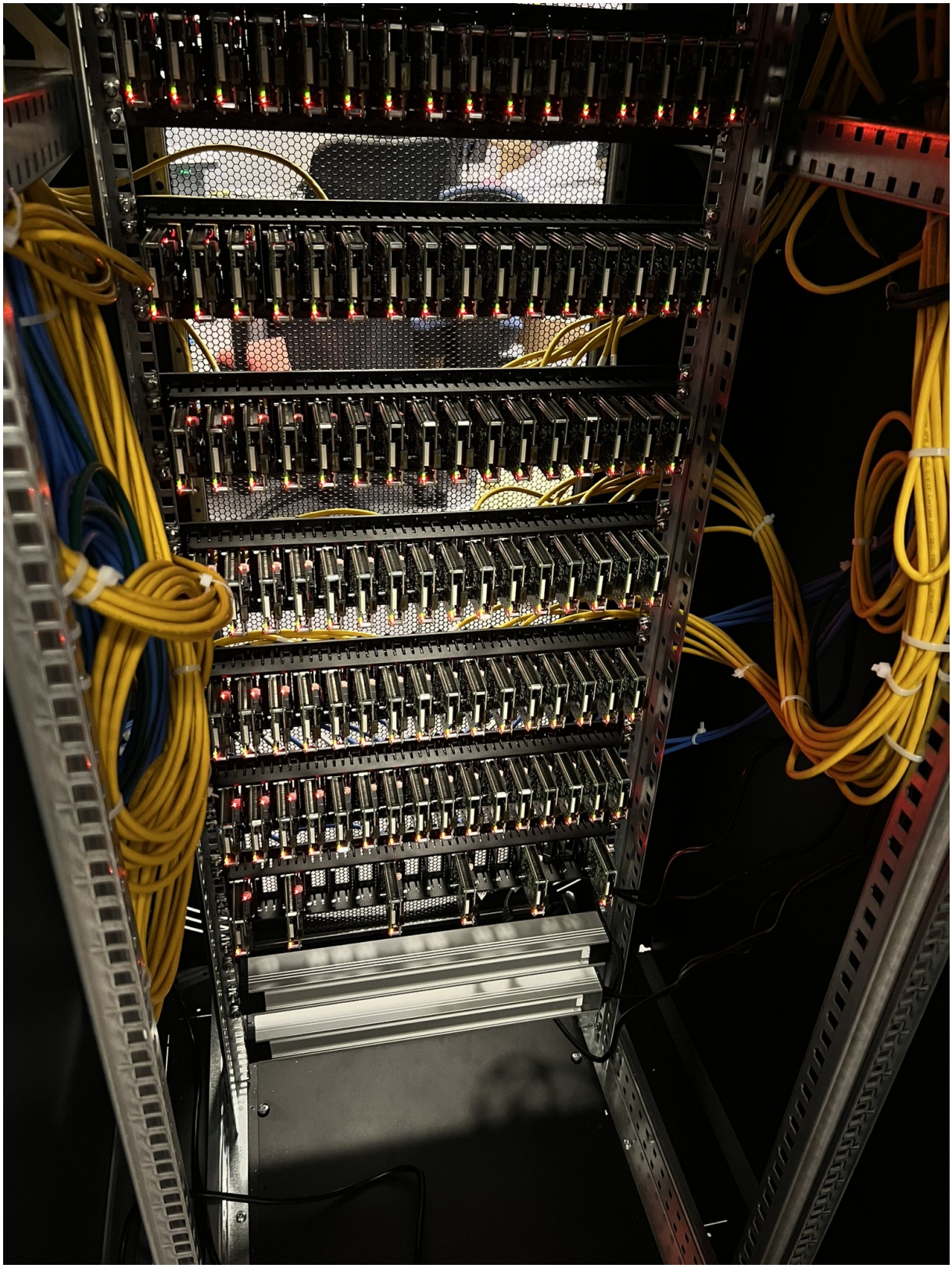

Figure 6: Photo du grand banc de test (100 Raspberry Pi)

Annexe B : Diagramme de classes de multitool

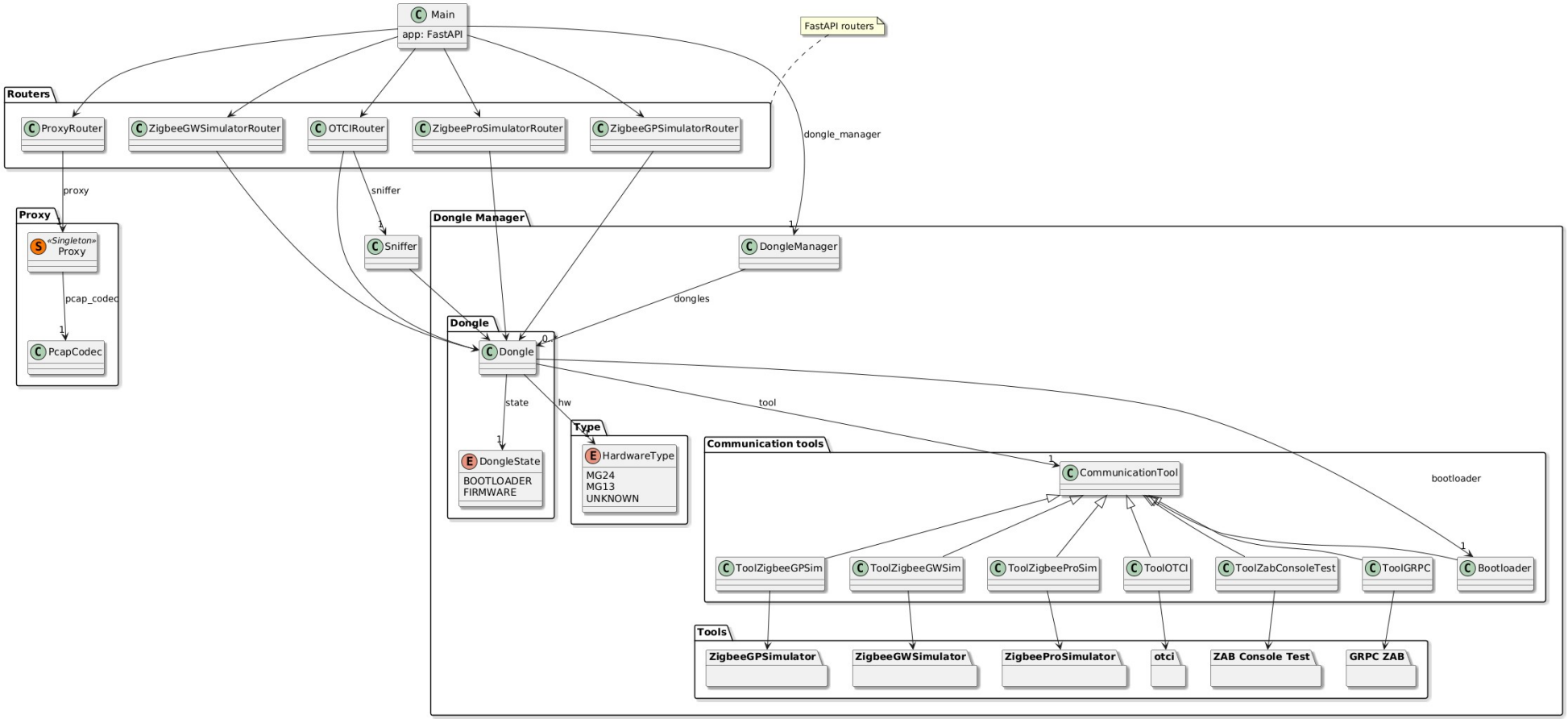

Figure 7: Diagramme de classes de multitool

Résumé :

Dans le cadre de mon alternance chez Elyxoft, j'ai été impliqué dans un projet stratégique pour Schneider Electric, centré sur l'évaluation de protocoles radio pour objets connectés, dans un contexte domotique et industriel. J'ai conçu une image système nommée *multitool*, destinée à équiper des Raspberry Pi pour automatiser la configuration de modules radio Zigbee et Thread. Cette solution s'intègre dans une architecture plus large de banc de test, accessible via une API REST et orchestrée à l'aide de scripts Python. Le développement s'est fait de manière itérative, en interaction continue avec l'équipe Wireless Connectivity Team de Schneider. L'image multitool est aujourd'hui capable de détecter automatiquement les dongles connectés, de téléverser les firmwares adaptés, et de piloter des tests à distance. Sa souplesse permet de simuler rapidement des scénarios complexes, comme l'appairage massif de nœuds Zigbee ou des communications maillées en Thread. Cette expérience m'a permis de monter en compétences sur les systèmes embarqués, les réseaux radio, et le développement logiciel en environnement industriel. Le projet a également mis en lumière les bénéfices d'une approche modulaire et automatisée dans les phases de test produit. Au-delà de l'aspect technique, j'ai aussi beaucoup appris en termes de gestion de projet, de collaboration à distance et d'autonomie dans le travail.

Mots clés :

objets connectés, banc de test, Raspberry Pi, Zigbee, Thread, systèmes embarqués, automatisation.

Abstract:

As connected devices increasingly rely on wireless protocols like Zigbee and Thread, there is a growing need for tools to test their behavior under realistic and repeatable conditions. This work was conducted during my apprenticeship at Elyxoft, in collaboration with Schneider Electric. I developed a modular and automated test infrastructure based on Raspberry Pi boards and USB radio dongles. At the heart of this system is *multitool*, a custom Linux image capable of detecting, configuring, and flashing radio modules automatically. This image exposes a REST API used by a Python test framework to execute scripted test scenarios across the networked Raspberry Pis. The resulting testbed supports large-scale, reproducible experiments simulating mesh networks under different loads and configurations. It enables seamless testing of connectivity, performance, and interoperability for Zigbee and Thread protocols. Early tests, such as Thread ping communication and massive Zigbee pairing, were successfully deployed and analyzed using this platform. This project demonstrates the feasibility and value of an automated, flexible test environment in the evaluation of low-power wireless technologies. It helps reduce manual effort, improves reproducibility, and contributes to better-informed decisions in protocol adoption.

Keywords:

wireless protocols, Zigbee, Thread, IoT testbed, embedded systems, automation, network simulation.