

POLYTECH ANGERS
RAPPORT DE PROJET 4A

Automatisme avec Unipi et Raspberry Pi



AYA RHOUMDANE
BAPTISTE MARCHAND

Tuteur : M. DELANOUÉ
Année 2022/2023

Abstract

As engineering students, we undertook a project to understand and control the Unipi card using the Mervis IDE development environment. Our primary objective was to find new solutions for controlling the Unipi card without Mervis IDE, exploring open-source software tools available on Raspberry Pi. The project also involved designing a gas-filling system for a fleet of vehicles using Unipi and Raspberry Pi, implementing appropriate security measures, and saving data in a database.

Throughout the project, we gained hands-on experience in exploring the hardware and software architecture of the Unipi solution with Mervis IDE, implementing a proof of concept of new solutions without Mervis IDE, and testing the results. We also explored and used various new technologies such as Flask and Redis to develop a web interface for the gas filling system.

In conclusion, this project provided us with a great opportunity to learn new technologies, apply our engineering skills, and collaborate with each other to develop an innovative solution to a real-world problem. We believe that our project has the potential to improve the efficiency and safety of gas-filling systems for vehicles, and we are excited about the possibilities for further development and application of our solution.

Using Unipi is the future of automated systems, and we are grateful for the opportunity to contribute to its development and learn new technologies in the process.

Contents

Introduction	6
1 Présentation de la problématique et généralités sur l’Unipi	7
1.1 Présentation du projet	7
1.1.1 Unipi	7
1.1.2 Objectifs à atteindre	7
1.2 Description de l’existant	8
1.2.1 Analyse de l’existant	8
1.2.2 Proposition de la solution	9
1.2.3 Comparaison	10
2 Mise en place de la solution Unipi avec Mervis IDE :	11
2.1 Architecture matérielle :	11
2.1.1 Raspberry Pi 4 :	11
2.1.2 Unipi 1.1 :	13
2.2 Architecture logicielle :	14
2.2.1 Mervis IDE :	15
2.2.2 Interface Homme Machine :	16
2.2.3 SCADA :	17
2.2.4 Mervis DB :	18
2.2.5 Mervis RT	18
2.3 Mise en pratique de la solution :	19
2.3.1 Description du schéma synoptique du projet :	19
2.3.2 Système d’exploitation Mervis	19
2.3.3 Liaison avec l’automate	21
2.3.4 configuration matérielle :	21
2.3.5 Programme FBD de notre solution	22
2.3.6 Conception de l’interface homme-machine	23
2.4 Tests et résultats :	26
3 Mise en place de la solution Unipi sans API (EVOK) et sans Mervis:	29
3.1 Description de la solution	29
3.2 Implémentation de la solution	29
3.2.1 Schémas de branchement des broches GPIO de la Raspberry Pi	29
3.2.2 Mise en œuvre du contrôle des broches GPIO de la Raspberry Pi en C	31

3.2.3	Mise en œuvre du contrôle des broches du MCP23008 en C	31
3.3	Limitations de la solution Unipi sans l'utilisation de l'API EVOK ni de Mervis	34
4	Utilisation des solutions Unipi API	34
4.1	Configuration de notre réseau	36
4.2	Installation d'EVOK	39
4.3	Utilisation d'EVOK	40
5	La solution proposé	41
5.1	Json RPC	42
5.2	La base de données	43
5.3	Flask	45
5.4	Communication entre les scipts python à l'aide de Redis	46
5.4.1	Description de la solution	46
5.4.2	Configuration de Redis	47
5.4.3	Connexion à un serveur Redis distant	48
	Conclusion et points d'amélioration	50
	Code final	51
	Code webserver : Flask.py	52
	Code principal : Gestion.py	53
	Page Html principale	56
	Page Html de validation	59

List of Figures

1	Exemple d'installation pilotée via un contrôleur Unipi	7
2	Structure interne d'un automate programmables industriels (API)	9
3	Architecture matérielle	10
4	Raspberry Pi 4	12
5	Unipi 1.1	13
6	Langage FBD	16
7	Language ST	16
8	Interface Homme Machine	17
9	SCADA	18
10	Mervis DB	19
11	Schéma synoptique du projet	20
12	Insertion de la carte SD	20
13	Adresse IP attribuée(we need to get our ip address)	21
14	UDP broadcast	21
15	Détection du Raspberry Pi.png	22
16	Configuration matérielle	22
17	Programme FBD	23
18	propriétés du terminal	24
19	propriétés du terminal Channel	24
20	Paramètres SSCP	25
21	Paramètres WB	25
22	Configuration de l'élément indicateur de valeur (value indicator)	26
23	Interface Homme Machine	26
24	Page Web	27
25	Branchemet de l'entrée analogique	27
26	Branchemet du relai	28
27	Montage de la solution	29
28	Raspberry Pi P1 header map	30
29	Raspberry Pi P1 header map	30
30	MCP23008 pin map	30
31	-MCP23008 pinout	32
32	Schéma de câblage du MCP23008 sur le Raspberry Pi en I2C .	32
33	APIs proposées par Unipi	35
34	Architecture EVOK	36
35	Configuration réseau	37
36	Configuration réseau du raspberrypi	38
37	Partage réseau windows	38

38	Activation de VNC	38
39	Connexion avec VNC	39
40	Résultat de l'installation d'EVOK	40
41	Architecture logicielle	42
42	Conneion à la BD à partir du code Python	44
43	Test de la Base de données	44
44	Changement de fuseau horaire	45
45	Page web de renseingement des données	45
46	Lancement de Flask dans une interface en ligne de commande	46

Introduction

Les avancées de l'Unipi et de l'association Raspberry Pi transforment le paysage de l'automatisation industrielle en intégrant des technologies de pointe pour améliorer l'organisation globale . Les cartes de prototypage comme Arduino, Raspberry Pi, UDOO et Spark Core ont révolutionné le fonctionnement des industries, et les chercheurs se concentrent sur le développement de méthodes et d'outils pour améliorer ces technologies pour une meilleure efficacité . Des descriptions détaillées de la programmation PISA suggèrent qu'elle peut être utilisée pour l'analyse sur Raspberry Pi ou d'autres cartes à usage général, ce qui peut changer la donne dans l'industrie de l'automatisation industrielle .

Ce rapport présente le projet dans lequel les capacités des systèmes Unipi et Raspberry Pi ont été explorées pour l'automatisation industrielle. Avec la disponibilité d'ordinateurs abordables et de petite taille, ces systèmes offrent une alternative aux contrôleurs industriels traditionnels pour la surveillance, le contrôle et la régulation des systèmes automatisés. La carte d'extension Unipi pour Raspberry Pi offre une architecture E/S universelle avec des entrées numériques et analogiques, des relais et des options de connectivité pour l'analyse, le contrôle et la régulation des systèmes automatisés.

Dans ce contexte, notre projet 4A SAGI consiste en la réalisation d'un système d'automatisation industrielle à l'aide de Raspberry Pi et Unipi, sans l'utilisation de l'interface de développement Mervis. Nous avons cherché à trouver de nouvelles solutions pour utiliser ces deux outils de manière plus efficace et innovante, en proposant une preuve de concept pour une nouvelle méthode de contrôle de la partie opérative du remplissage de gaz dans une flotte de véhicules. Ce rapport de projet détaille notre approche et nos résultats, ainsi que les défis que nous avons rencontrés et les opportunités que nous avons identifiées pour l'avenir de l'automatisation industrielle.

1 Présentation de la problématique et généralités sur l'Unipi

1.1 Présentation du projet

1.1.1 Unipi

Depuis quelques années, de nouveaux ordinateurs de tailles réduites et à petit prix sont disponibles. La star de cette catégorie est le Raspberry Pi. Dans une logique de hacker (comprendre débrouillard), c'est souvent un système d'exploitation GNU/Linux qui est installé.

Via ce système matériel et logiciel, il est possible de programmer un système industriel comme peut le faire un automate programmable industriel (API). La société Unipi met à disposition des cartes d'extension pour l'ordinateur Raspberry Pi qui lui permet de fonctionner comme un contrôleur logique programmable pour la surveillance, le contrôle et la régulation des systèmes d'automatisation (voir figure 1). La carte présente une architecture d'E/S universelle avec des entrées numériques, un ensemble de relais, des E/S analogiques et une large connectivité grâce au bus 1-Wire, aux ports I2C et UART. Après quelques paramétrages et branchements, il est possible de programmer une installation industrielle tout en profitant d'un système d'exploitation basé sur Linux, pour par exemple historiser les données.

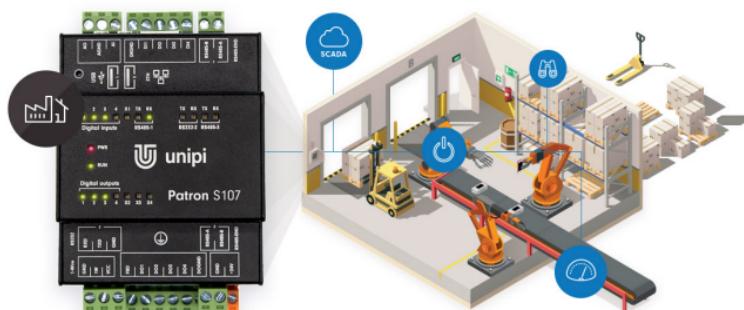


Figure 1: Exemple d'installation pilotée via un contrôleur Unipi

1.1.2 Objectifs à atteindre

Dans le cadre de ce projet 4A, notre objectif initial était de comprendre le fonctionnement et le pilotage de notre carte Unipi en utilisant l'environnement

de développement Mervis IDE. Mervis est une plateforme logicielle offrant une solution professionnelle adaptée à une large gamme d'applications, de la gestion des bâtiments (BMS) à l'automatisation industrielle.

Cependant, notre objectif ultime était de trouver une nouvelle solution pour piloter notre carte Unipi sans Mervis IDE. Nous avons donc exploré de nouvelles méthodes pour contrôler les entrées/sorties de notre carte Unipi en utilisant d'autres outils logiciels open-source disponibles sur Raspberry Pi. Nous avons ainsi réalisé une preuve de concept de nouvelles solutions pour l'utilisation de ces systèmes sans avoir recours à Mervis IDE.

Notre projet avait également pour objectif de concevoir un système de remplissage de gaz pour une flotte de véhicules en utilisant la carte Unipi et Raspberry Pi. Nous avons ainsi élaboré une interface homme-machine pour le système, mis en place des mesures de sécurité appropriées, et sauvegardé les données en base de ce système de remplissage de gaz.

1.2 Description de l'existant

Dans le cadre du projet précédent, l'IDE Mervis a été utilisé pour contrôler la carte Unipi et créer une interface utilisateur efficace. Cependant, cette solution présentait également des faiblesses telles que la nécessité d'acheter des licences pour l'utilisation de l'IDE et des problèmes de sécurité pour la connexion des utilisateurs. Bien que l'utilisation de l'IDE Mervis ait permis de développer rapidement un système fonctionnel, certains utilisateurs peuvent trouver qu'elle présente des limites en termes de flexibilité et de personnalisation.

De plus, certains utilisateurs peuvent préférer développer leur propre logiciel et programmes de contrôle plutôt que de se fier à des solutions logicielles préconçues comme Mervis IDE. Cela peut offrir plus de flexibilité et de personnalisation, mais cela nécessite également un niveau de compétences en programmation plus élevé.

1.2.1 Analyse de l'existant

Un contrôleur logique programmable (de l'anglais : Programmable Logic Controller (PLC)) est un composant qui est programmé et utilisé pour la commande ou la régulation d'une machine ou d'une installation. Ces commandes sont mises en application dans des secteurs très différents, par exemple pour commander des installations de production, des presses à injection ou justement des machines d'essais de dureté entièrement automatisées. Un PLC est pourvu d'entrées, de sorties, d'un système d'exploitation (firmware) et d'une interface par laquelle peut être chargé un programme d'application (logiciel

d'exploitation). Le programme d'application définit la façon dont les sorties doivent être commutées en fonction des entrées. Le système d'exploitation (firmware) fait en sorte que le programme d'application ait toujours à sa disposition l'état actuel des capteurs. Sur la base de ces informations, le programme d'application peut commuter les sorties de sorte que la machine ou l'installation fonctionne comme prévu. La liaison entre le PLC et la machine est réalisée par des capteurs et des actionneurs. Les capteurs sont branchés aux entrées du PLC et communiquent à ce dernier ce qui se passe dans la machine d'essais de dureté. Les capteurs sont par exemple des palpeurs, des interrupteurs de fin de course ou des capteurs de force (cellule de charge). Les actionneurs sont raccordés aux sorties du PLC et permettent de commander la machine d'essais de dureté. Les actionneurs sont par exemple différents servomoteurs destinés à l'application de la force ou au pilotage des axes.

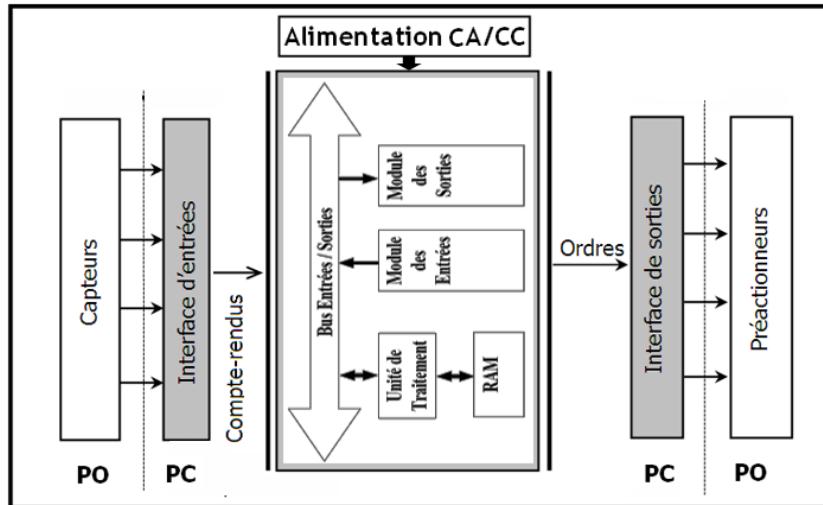


Figure 2: Structure interne d'un automate programmable industriel (API)

1.2.2 Proposition de la solution

Notre objectif principal est de proposer une solution alternative au PLC traditionnel en utilisant la carte d'extension Unipi pour Raspberry Pi. Cette solution permet d'avoir un contrôleur logique programmable moins coûteux tout en conservant une puissance de traitement suffisante.

La solution que nous proposons est basée sur l'utilisation d'un Raspberry Pi en tant qu'unité centrale de commande, connecté à une carte d'extension Unipi qui agit en tant qu'unité de puissance. La combinaison de ces deux

éléments permet de créer un système de contrôle flexible et évolutif, capable de répondre à diverses exigences de l'industrie.

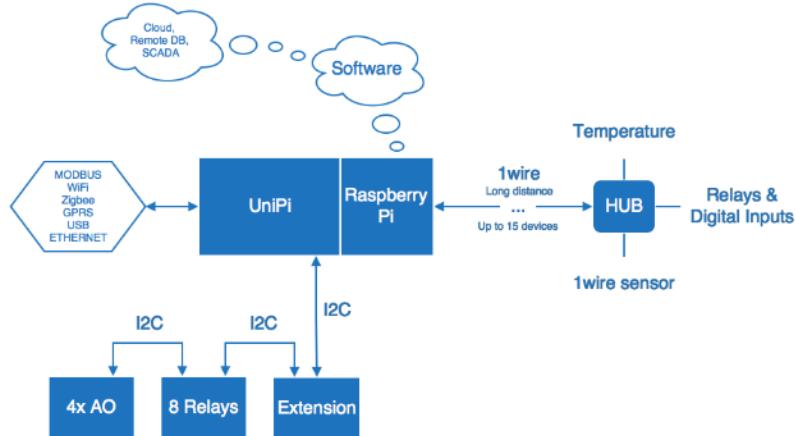


Figure 3: Architecture matérielle

1.2.3 Comparaison

Le choix entre Unipi et un PLC peut dépendre de plusieurs facteurs, notamment les exigences du projet, les performances requises et le budget disponible. Voici une comparaison entre les deux solutions en termes de coût et de performances :

Coût :

Les cartes d'extension Unipi pour Raspberry Pi sont généralement moins chères que les PLC traditionnels. En effet, les Unipi peuvent être acquises pour une fraction du coût d'un PLC, ce qui en fait une solution rentable pour les petites et moyennes installations industrielles.

Performances :

Les Unipi sont équipés d'un processeur ARM puissant qui peut gérer efficacement les tâches de contrôle et de surveillance, tout en offrant une grande flexibilité grâce à son architecture d'E/S universelle. Cependant, les PLC traditionnels ont été développés spécifiquement pour les applications industrielles et ont des performances prouvées en termes de fiabilité et de robustesse.

Type d'Automate	PLC	Raspberry + Unipi
Performances du CPU	Moyen à élevé	Moyen
Taille de la mémoire	Moyen à élevé	Moyen à élevé
Alimentation électrique	Elevé	Bas à moyen
Interface d'entrées	Modulaire	Moyen
Interface de sorties	Modulaire	Modulaire
Interface de communication	Moyen à élevé	Moyen
Encombrement	Moyen à élevé	Petit
Personnalisable	Modulaire	Modulaire
Coût	Elevé	Bas
Compatibilité Électromagnétique	Oui	Non

Tableau 1 – Comparaison PLC-Unipi

2 Mise en place de la solution Unipi avec Mervis IDE :

2.1 Architecture matérielle :

2.1.1 Raspberry Pi 4 :

Le Raspberry Pi 4 B est un nano-ordinateur pouvant se connecter à un moniteur, à un ensemble clavier/souris et disposant d'interfaces Wi-Fi, Bluetooth et Ethernet.

Il fonctionne depuis une carte micro-SD et fonctionne avec un système d'exploitation basé sur Linux ou Windows 10 IoT. Il est fourni sans boîtier, alimentation, clavier, écran et souris dans le but de diminuer le coût et de favoriser l'utilisation de matériel de récupération.

Cette nouvelle version de la carte Raspberry Pi 4 B est basée sur:

- 1 x processeur ARM Cortex-A72 64 bits quatre coeurs à 1,5 GHz

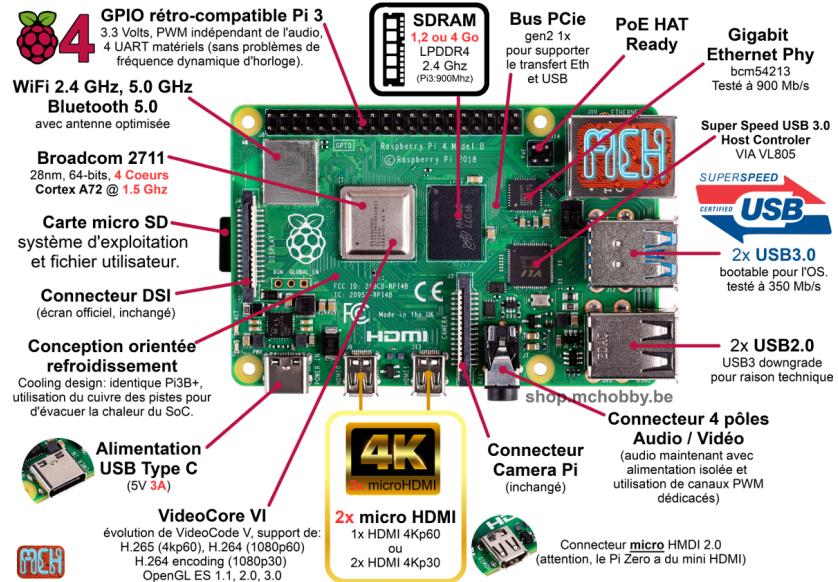


Figure 4: Raspberry Pi 4

- 4 GB de mémoire RAM (existe également en version 1 et 2 GB de RAM)
- 1 x interface Wi-Fi
- 1 x interface Bluetooth
- 2 x ports USB 2.0
- 2 x ports USB 3.0
- 1 x port Ethernet Gigabit
- 2 x ports micro-HDMI pour deux écrans jusqu'à 4k (3840 x 2160 pixels) à 60 FPS
- 1 x port micro-SD
- 1 x connecteur GPIO avec 40 broches d'E/S
-

Les interfaces Ethernet et Bluetooth ont été améliorées par rapport à la version Pi 3 B+ et supportent maintenant l'Ethernet Gigabit ainsi que le Bluetooth 5.0.

Cette carte est basée sur un processeur ARM et permet l'exécution du système d'exploitation GNU/Linux/Windows 10 IoT et des logiciels compatibles.

2.1.2 Unipi 1.1 :

Unipi 1.1 est une carte d'extension pour le mini-ordinateur Raspberry Pi qui peut être utilisée comme contrôleur logique programmable (PLC). Il s'agit d'un contrôleur simple et peu coûteux pour les maisons intelligentes, les systèmes BMS (Technical Building Management) et les développements innovants dans le domaine de l'IoT. Les exemples incluent l'automatisation du chauffage des bâtiments, l'allumage automatique des lumières, les systèmes de gicleurs, le contrôle de garage et de portail, et d'autres applications. Unipi est une unité de contrôle abordable qui vous aide à améliorer le confort et à réduire les coûts d'exploitation.

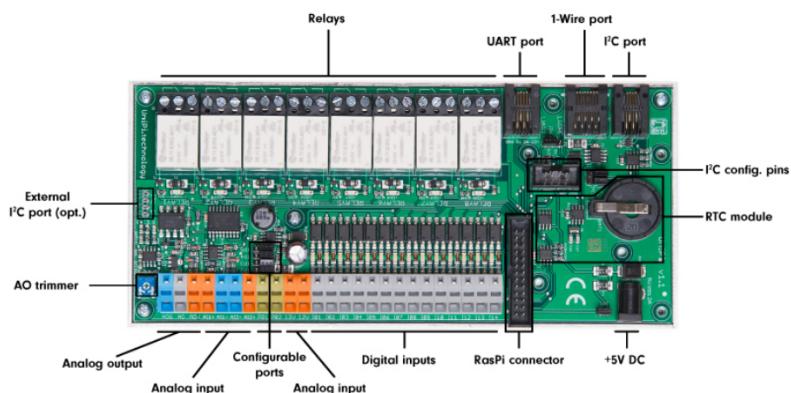


Figure 5: Unipi 1.1

Description de l'Unipi :

Principaux éléments constitutifs de l'Unipi :

- Relais : 8 relais 250VAC@5A ou 24VDC@5A - éléments de commutation de contrôle
- Port UART : Port série standard pour connecter une console série ou de nombreux autres dispositifs (lecteurs NFC, ...).
- Port 1Wire : Fournit une interface de bus 1Wire pour connecter des dispositifs 1Wire tels que des capteurs de température et d'humidité.
- Port I2C : Pour connecter d'autres modules d'extension, par exemple des modules de relais ou de sortie analogique
- Broches de configuration I2C : Pour connecter le bus I2C_0 du RPi (seulement pour les utilisateurs avancés)
- Module RTC (Real Time Clock) : Fournit une horloge en temps réel en cas de panne d'électricité ou d'internet.
(la batterie de secours n'est pas incluse dans l'emballage).

- Alimentation 5V : connecteur de 2,1 mm pour l'alimentation électrique.
- Connecteur RPI : connecteur 26 broches pour Raspberry Pi
- Entrées numériques : 12(+2) entrées numériques isolées galvaniquement
- Sortie 12V Alimentation 12V@200mA - à utiliser uniquement avec les entrées numériques de l'Unipi
- Ports configurables : Pour configurer les entrées numériques en vue d'une utilisation avec une source d'alimentation externe
- Entrée analogique : Deux entrées analogiques 0-10V pour la lecture de signaux analogiques provenant de dispositifs externes
- Sortie analogique : Une sortie analogique 0-10V pour le contrôle proportionnel
- Trimmer AO : Pour un réglage précis de la sortie analogique

2.2 Architecture logicielle :

La plate-forme principalement prise en charge pour la programmation des contrôleurs Unipi est le système Mervis. Mervis est développé conformément à la norme IEC 61131-3 pour la programmation des API et se compose d'une interface SCADA pour le contrôle et la surveillance à distance, d'une interface de développement compréhensible, d'un éditeur HMI intégré et d'une base de données en ligne ou sur site pour stocker les données et effectuer des analyses historiques de l'opération de la technologie contrôlée.

Mervis est une plateforme logicielle à part entière fournie gratuitement pour tous les contrôleurs Unipi. Il s'agit d'une solution professionnelle adaptée à une large gamme d'applications, du contrôle de la maison intelligente à l'automatisation industrielle. L'environnement de développement Mervis IDE, qui est au cœur de la plate-forme, convient aussi bien aux débutants qu'aux programmeurs expérimentés grâce à son interface conviviale et à sa conception claire.

Mervis est une plateforme logicielle fournie par la société Unipi. Il s'agit d'une solution professionnelle adaptée à une large gamme d'applications. L'environnement de développement Mervis IDE, qui est au cœur de la plate-forme, permet de :

- Créer et déboguer des programmes de contrôle pour PLC dans un environnement de développement convivial Mervis IDE.
- Créer et éditer des interfaces utilisateur web pour un contrôle confortable du système.
- Communiquer avec les contrôleurs à distance, de n'importe où, grâce à un accès à distance sécurisé.
- Surveiller et analyser les données historiques à l'aide d'une interface SCADA professionnelle.

- Surveiller le système grâce à une application mobile sur le smartphone ou la tablette.

2.2.1 Mervis IDE :

Mervis IDE (environnement de développement intégré) est la pierre angulaire de la plateforme Mervis. Il vous permet de :

- Se connecter aux unités, même à distance via Internet
- Gérer la configuration des unités, y compris les mises à jour de Mervis RT
- Créer des programmes dans les langages FBD ou ST selon la norme IEC61131-3
- Créer des bibliothèques de fonctions et de périphériques Modbus pour les réutiliser facilement dans d'autres projets
- Construire des IHM (interfaces homme-machine) basées sur le web
- Créer des projets pour Mervis SCADA - environnement pour la gestion des installations à distance
- Déetecter les capteurs 1-Wire et les extensions Unipi connectées
- Télécharger et déboguer les solutions sur les unités exécutant Mervis RT.

L'IDE Mervis supporte deux méthodes de programmation conformes à la norme IEC 61131-3 pour créer le comportement du contrôleur.

- Langage FBD :

FBD est un langage de programmation basé sur des blocs de code prédéfinis. Chaque bloc a sa fonction spécifique et ses entrées (valeur de température, signal de commutation) et sorties (commande de commutation, régulation, etc.). La logique de commande proprement dite est ensuite créée dans une interface graphique claire en reliant simplement les blocs et les variables.

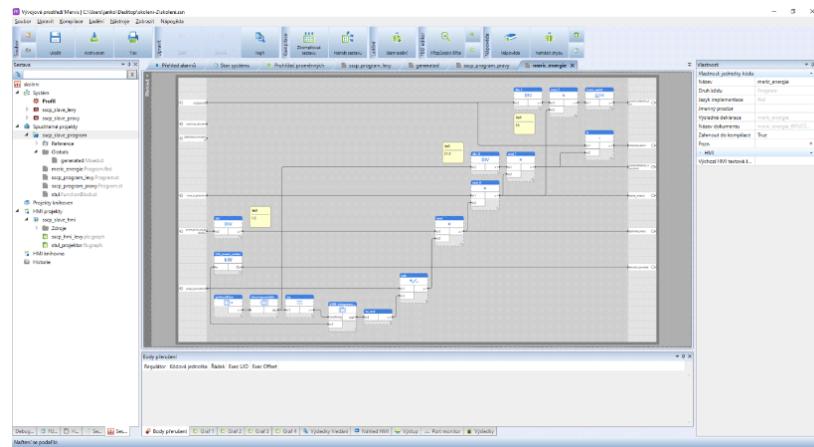


Figure 6: Langage FBD

- Langage ST :

Le langage ST est basé sur un texte structuré capable d'exprimer des fonctions complexes par seulement des lignes de code. Le ST est surtout utile pour les utilisateurs expérimentés et la programmation de projets complexes étendus. Le ST peut également être utilisé pour créer des blocs de fonctions personnalisés si nécessaire.

Figure 7: Language ST

2.2.2 Interface Homme Machine :

Outre la logique de commande proprement dite, Mervis IDE vous permet

également de créer des interfaces graphiques utilisateur pour des pages web locales s'exécutant sur votre API ou pour le service Mervis SCADA. Ces interfaces vous permettent de visualiser l'état de votre système et de contrôler toutes ses fonctions. À cette fin, l'éditeur dispose d'une grande collection de commutateurs, d'indicateurs, de champs de texte, de planificateurs de temps et d'autres éléments. Les différents éléments, chacun avec de larges options de personnalisation, peuvent être placés simplement par glisser-déposer. L'éditeur prend également en charge le téléchargement de vos images ou icônes. Les interfaces complètes peuvent être consultées via un ordinateur, une tablette ou un smartphone en utilisant simplement un navigateur web standard.

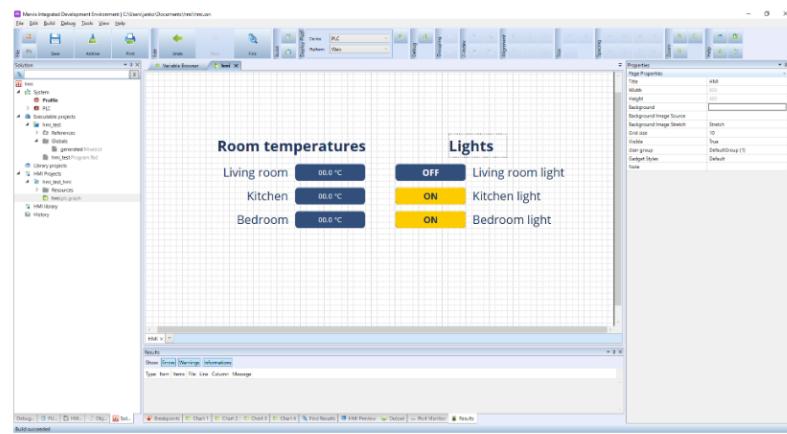


Figure 8: Interface Homme Machine

2.2.3 SCADA :

Mervis SCADA (Supervision, Commande et Acquisition de Données) est un service de cloud pour la gestion, le contrôle et l'analyse des données historiques à distance. SCADA vous donne la possibilité de surveiller et de contrôler tous les systèmes connectés à travers une interface graphique claire, indépendamment de leur emplacement. Ainsi, un seul centre de supervision peut contrôler les technologies installées sur différents continents. L'éditeur graphique de Mervis IDE vous permet ensuite de créer et de modifier vos conceptions de visualisation SCADA.

Cette fonctionnalité est réservée pour la version complète (avec licence).



Figure 9: SCADA

2.2.4 Mervis DB :

L'automate reçoit des informations de capteurs ou de dispositifs d'entrée connectés, traite les données et déclenche des sorties en fonction de paramètres préprogrammés. En fonction des entrées et des sorties, un automate peut surveiller et enregistrer des données de fonctionnement telles que la productivité de la machine ou la température de fonctionnement, lancer et arrêter automatiquement des processus, générer des alarmes en cas de dysfonctionnement de la machine, etc. Les automates programmables constituent une solution de contrôle flexible et robuste, adaptable à presque toutes les applications.

Cette fonctionnalité est réservée pour la version complète (avec licence).

2.2.5 Mervis RT

Mervis RT est l'environnement d'exécution permettant d'exécuter les programmes créés et déployés à partir de Mervis IDE. Le RT permet également d'accéder à l'IHM basée sur le web via son serveur web interne. Mervis Runtime est essentiel pour exécuter des programmes de l'automate créés dans l'IDE Mervis. Sans licence, une période d'essai de 20 minutes commencera au démarrage de l'automate ou de sa logique de commande. Au cours de cet essai, nous pouvons librement explorer toutes les fonctions de Mervis. Une fois la période d'essai expirée, le Mervis RT se désactive et arrête le programme. Si l'Automate Programmable Industriel n'a pas de licence, il fonctionnera en mode démo. Le mode démo ne limite pas la fonctionnalité, mais le programme ne fonctionnera que pendant 20 minutes. Passé ce délai,

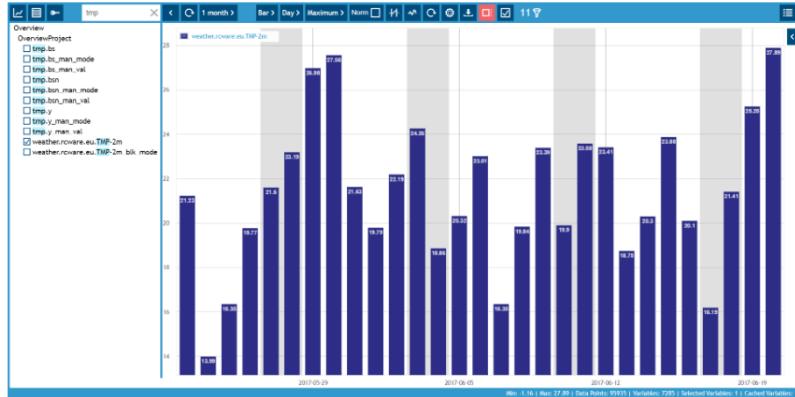


Figure 10: Mervis DB

le programme s'arrêtera et les LED RUN commenceront à clignoter dans un intervalle de 2 secondes (on/off).

2.3 Mise en pratique de la solution :

2.3.1 Description du schéma synoptique du projet :

Le projet consiste à développer un système de pilotage pour une partie opérative destinée au remplissage d'une flotte de véhicules fonctionnant au gaz. Pour cela, nous avons utilisé une carte Raspberry Pi ainsi qu'une carte Unipi 1.1, qui nous ont permis de contrôler différents aspects du processus de remplissage, tels que l'allumage des compresseurs, la sauvegarde des données en base de données, la supervision, l'interface homme-machine, ainsi que la gestion de la sécurité.

Le système développé nous a permis de réaliser un contrôle précis et efficace de toutes les étapes du processus de remplissage, tout en assurant la sécurité des opérateurs et des véhicules. Nous avons également mis en place une interface graphique utilisateur intuitive, permettant aux opérateurs de contrôler facilement toutes les fonctions du système.

2.3.2 Système d'exploitation Mervis

Au niveau de la carte SD, il faut cloner une version particulière de Linux, cette version est fournie par la solution Mervis et qui permet de rendre compatible l'ensemble d'éléments de notre montage. Ce système d'exploitation contient les outils Mervis et les pilotes Modbus TCP par défaut. L'image à

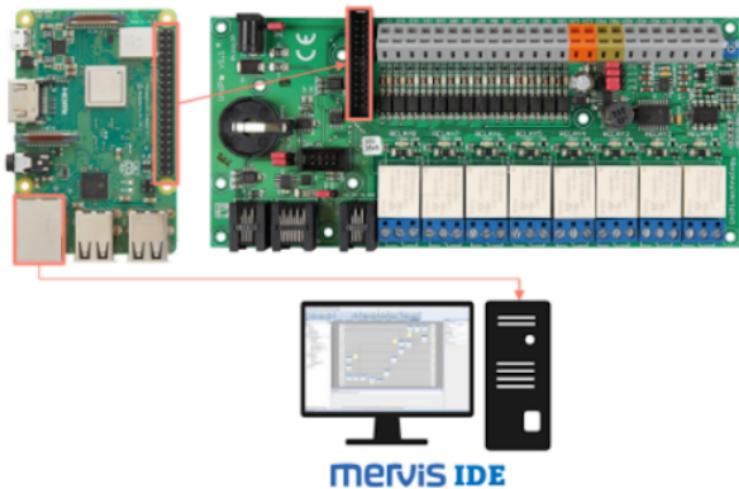


Figure 11: Schéma synoptique du projet

cloner est proposée gratuitement dans le site officiel de la solution. Pour installer ce système d'exploitation, il est nécessaire d'avoir une carte SD d'une capacité d'au moins 2 Go dans l'appareil.

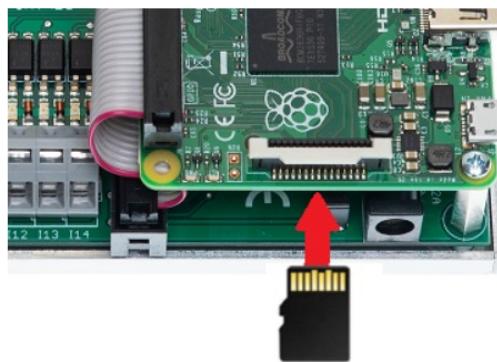


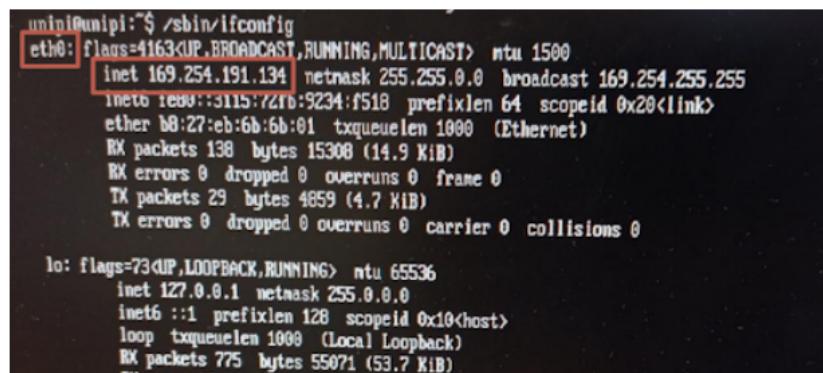
Figure 12: Insertion de la carte SD

Ensuite, nous insérons la carte dans la fente avec l'arrière vers le haut (c'est-à-dire, les connecteurs dorés vers nous), enfin, nous branchons l'alimentation du Raspberry Pi. Le démarrage de l'Unipi 1.1 avec le nouveau système d'exploitation flashé sur la carte SD prendra un peu plus de temps - environ 3 minutes. Ceci est dû à la configuration nécessaire du système d'exploitation.

L'authentification par défaut : Login : unipi Mdp : unipi.technology

2.3.3 Liaison avec l'automate

Après avoir bien mis en place l'OS, il faudra établir une connexion entre l'ordinateur qui contient l'IDE Mervis, et le Raspberry Pi contenant l'OS Mervis. Cela sera possible grâce à une liaison avec un câble Ethernet, en effet, si la connexion est bien établie, nous notons l'adresse IP attribuée à notre Raspberry Pi grâce au protocole DHCP, nous allons avoir la possibilité de fixer cette adresse IP pour les futures utilisations.



```
unipi@unipi:~$ /sbin/ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 169.254.191.134 netmask 255.255.0.0 broadcast 169.254.255.255
                inetb fe80::3115:72fb%eth0 brd 169.254.255.255 mngtmpv 1
                ether b8:27:eb:6b:6b:01 txqueuelen 1000 (Ethernet)
                  RX packets 138 bytes 15308 (14.9 KiB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 29 bytes 4859 (4.7 KiB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
          loop txqueuelen 1000 (Local Loopback)
            RX packets 775 bytes 55071 (53.7 KiB)
            TX packets 775 bytes 55071 (53.7 KiB)
```

Figure 13: Adresse IP attribuée(we need to get our ip adress)

Au niveau de l'IDE Mervis, il est possible de détecter automatiquement le Raspberry Pi lié avec l'ordinateur grâce à un UDP Broadcast.

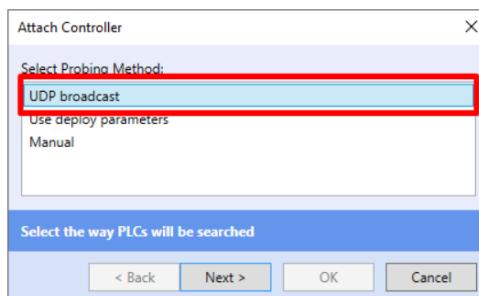


Figure 14: UDP broadcast

2.3.4 configuration matérielle :

Dans notre projet, nous avons utilisé Unipi 1.1/Lite en le connectant à Mervis IDE. Du point de vue de la construction, les contrôleurs Unipi se composent d'un module informatique (Raspberry Pi dans notre cas) et d'une carte de

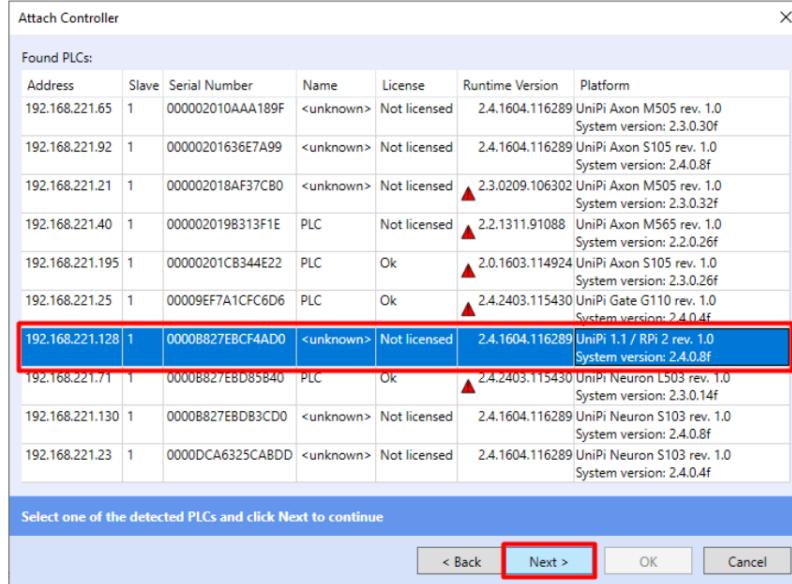


Figure 15: Détection du Raspberry Pi.png

circuit avec des entrées et des sorties. La carte communique avec l'ordinateur en utilisant les protocoles Modbus TCP et I²C. C'est la raison pour laquelle nous avons dû configurer deux canaux de communication pour contrôler les cartes Unipi 1.1/1.1 Lite. Le premier canal permet de contrôler les E/S analogiques et numériques via le protocole Modbus TCP, tandis que le canal I²C contrôle les sorties de relais.

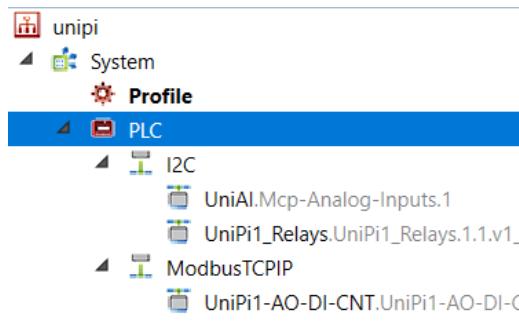


Figure 16: Configuration matérielle

2.3.5 Programme FBD de notre solution

Nous avons choisi d'utiliser le langage de programmation FBD. Dans la section logique de Mervis IDE, nous avons ajouté une fonction "ge(>=)" issue

de la fonction de comparaison du FUPLA Box Explorer. Nous avons placé nos deux entrées sur le côté gauche, qui sont les valeurs de l'entrée analogique 1 de l'Unipi et la consigne. Sur la section droite, nous avons placé notre sortie qui est le relais 2. Notre programme lit la valeur de l'entrée analogique 1 de l'Unipi, puis la compare à la consigne. Si la valeur est supérieure, le relais 2 est activé, sinon, le relais est désactivé.

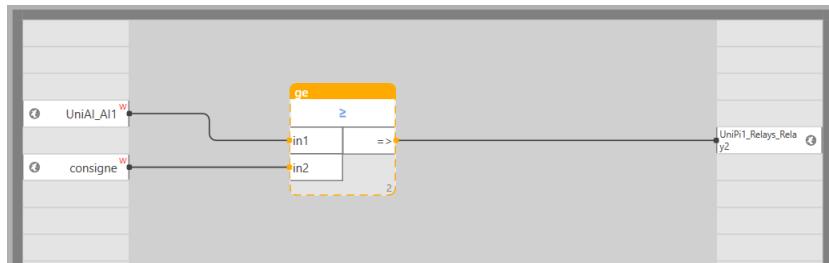


Figure 17: Programme FBD

2.3.6 Conception de l'interface homme-machine

-Création d'un terminal pour IHM externe :

Le terminal de serveur web externe est un périphérique virtuel dans Mervis IDE contenant des données sur les contrôleurs utilisés par le projet en question. Lors de la configuration, il est nécessaire de définir les propriétés nécessaires pour la connexion et le téléchargement du serveur web sur un automate programmable industriel (API). Un serveur web externe est toujours stocké sur un seul API et utilise le protocole SSCP pour communiquer avec les autres API inclus dans la définition du terminal et pour lire ou écrire dans une variable spécifique.

Nous avons commencé par créer une structure de terminal et avons ajouté sa description correspondante. Ensuite, nous avons assigné notre Unipi (PLC) au terminal créé. Cette étape nous a permis d'ajouter une interface homme-machine externe pour notre projet de remplissage de véhicules au gaz.

La dernière étape importante est de définir les paramètres SSCP pour les PLC. Ils servent à lire/écrire des variables des PLC ajoutés dans les étapes précédentes. Pour chaque PLC, les paramètres sont plus ou moins les mêmes, avec simplement des données différentes. Enfin, passons à la configuration proprement dite.

Maintenant, il faut attribuer le modèle au terminal. On clique sur le terminal, on accède à ses propriétés dans la colonne de droite et on sélectionne

Properties	
Terminal Properties	
Name	Terminal
Assigned Devic...	External Webse...
Language (Web)	en
Note	
· Terminal Connection Parameters	
Connection Type	FTP
Note	
... FTP Parameters	
Username	root
Password	root99
Host	localhost
TCP Port	21
Binary transfer	False
Note	
· Web Parameters	
Definition Direc...	
Device Template	
Custom Directo...	

Figure 18: propriétés du terminal

Properties	
Terminal Channel Properties	
Name	terminalChannel
Protocol	SSCP
Link Protocol	Tcp
Note	
· SSCP Protocol Parameters	
Note	
· TCP Parameters	
Note	

Figure 19: propriétés du terminal Channel

”Modèle de périphérique”. Ici, il faut télécharger le modèle que vous avez créé dans les étapes précédentes.

Après cette configuration, nous avons commencé à créer notre IHM en utilisant différents objets HMI. Ensuite, nous avons ajouté un élément indicateur de valeur (value indicator) que nous avons assigné à l'entrée analogique 1 et un régulateur analogique (Analog setter) pour contrôler la variable de

Properties	
Terminal Assigned Device Properties -	
Name	PLC
Connection Na...	Connection_0
Note	<input type="checkbox"/>
· SSCP Parameters	-
EndPoint	HwConfigura... <input type="checkbox"/>
Device Address	1 <input type="checkbox"/>
User Name	user <input type="checkbox"/>
Password	rw <input type="checkbox"/>
Note	<input type="checkbox"/>

Figure 20: Paramètres SSCP

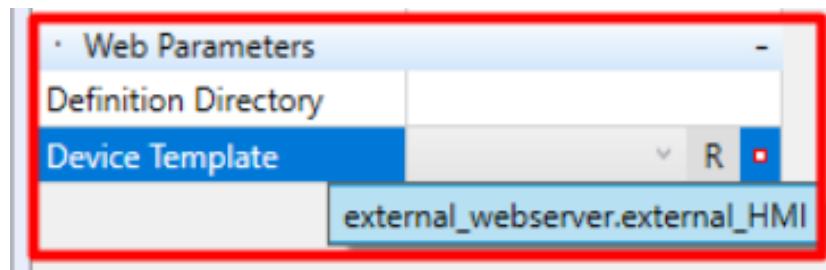


Figure 21: Paramètres WB

consigne. De plus, nous avons ajouté un indicateur numérique(Digital Indicator) pour afficher l'état de notre moteur (Relay 2) et un bouton de réglage numérique(Digital Setter Buttons) pour contrôler le relais 1 qui allumera et éteindra une LED.

-Sécuriser l'interface homme-machine (HMI) :

Sécuriser une IHM est une étape importante dans la définition des utilisateurs et des groupes d'utilisateurs ainsi que dans leur attribution d'accès à l'IHM. Si le login par défaut reste inchangé, tout utilisateur connecté au réseau peut accéder à l'IHM en utilisant les mots de passe par défaut et modifier les valeurs du PLC. Dans les propriétés de notre page IHM, nous avons défini un groupe d'utilisateurs ayant accès à notre page. Chaque utilisateur a été assigné à un groupe. Le groupe a également été configuré pour la lecture et l'écriture.

-Déploiement du serveur web dans notre PLC :

Properties	
Gadget Properties	
Name	Value Indicator
Tooltip	
Gadget Styles	Default
Note	
· HMI	
Variable	hw.UniAI_AI1
Text	
Format	##.##
Unit	

Figure 22: Configuration de l'élément indicateur de valeur (value indicator)

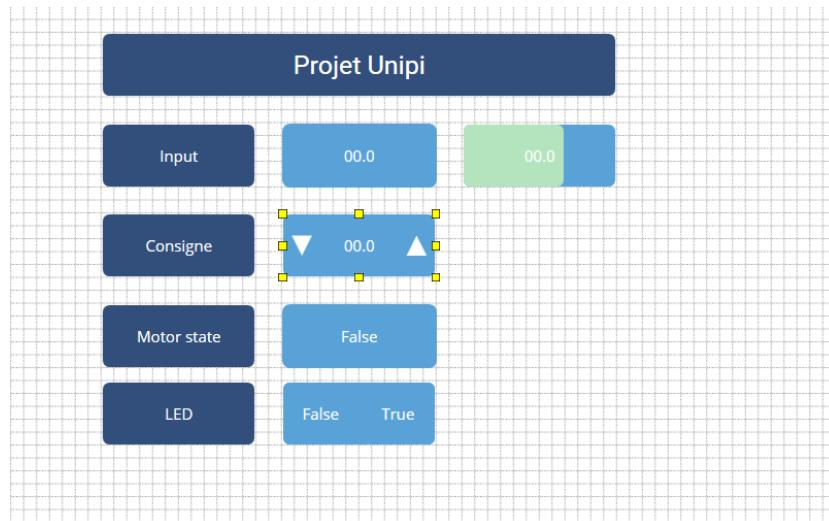


Figure 23: Interface Homme Machine

Pour déployer notre solution, nous avons cliqué sur "Déployer la solution". Une fenêtre de dialogue de déploiement est apparue. Nous avons coché la case Web à côté du PLC/terminal. Avec un serveur web externe, nous avons également vérifié que seul le serveur web spécifique était téléchargé.

2.4 Tests et résultats :

Nous avons implémenté toutes les étapes pour mettre en place une solution de commande de moteur électrique. Le fonctionnement du moteur est conditionné par la valeur d'entrée analogique fournie par le potentiomètre, qui doit

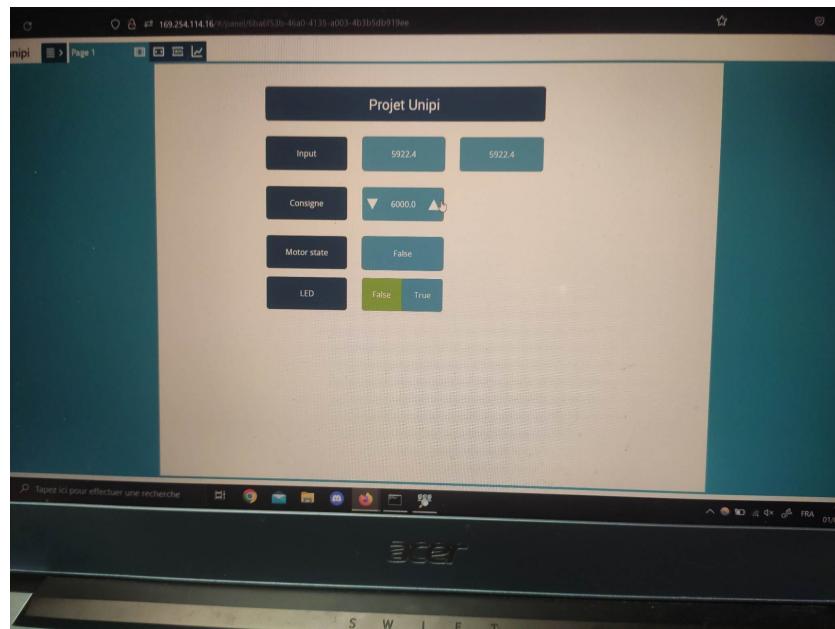


Figure 24: Page Web

dépasser un seuil prédéfini, lequel peut être modifié selon nos besoins.

Branchements du circuit :

-L'entrée analogique est connectée à un potentiomètre et pour mesurer la résistance comme indiqué sur l'image ci-dessous. L'alimentation électrique de 12V sur Unipi 1.1 est conçue pour les entrées numériques et peut être utilisée comme une tension flottante. Pour des lectures précises, nous avons utilisé une alimentation électrique externe.

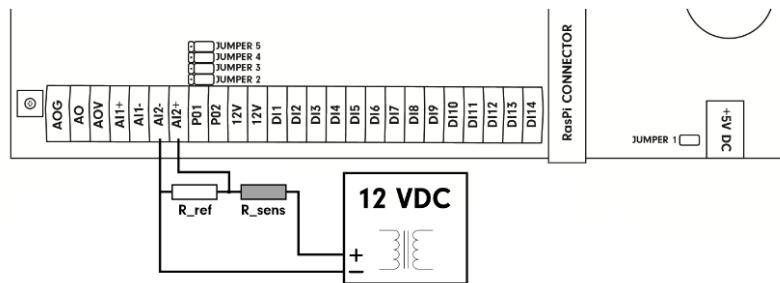


Figure 25: Branchements de l'entree analogique

-Nous avons choisi le relai 2 comme sortie numérique qui est reliée directement à un moteur électrique et une alimentation 6V DC, la bobine du relais sera excitée si la valeur analogique du potentiomètre est supérieure à 6000.

-Nous avons choisi le relais 1 comme sortie numérique, qui est directement connecté à une LED. L'utilisateur peut contrôler la LED via la page web ou HMI.

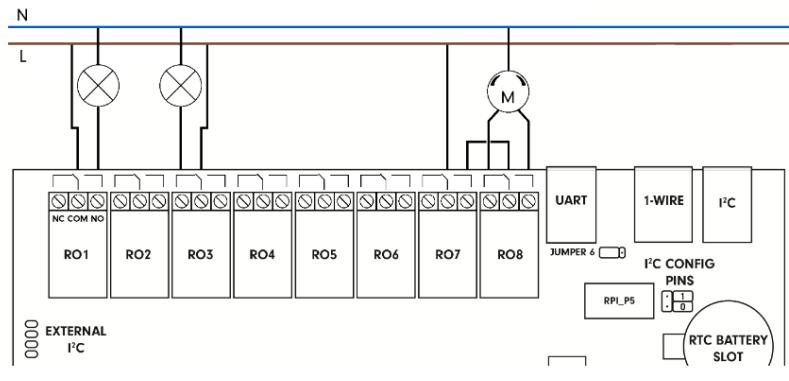


Figure 26: Branchement du relai

Matériel utilisé :

- Raspberry pi 4
- Carte d'extension Unipi 1.1
- Moteur Arduino en Fer et Plastique TT Moteur 3V-12V DC
- LED
- Resistance 250 Ohm
- Potentiomètre 0-1000 Ohm

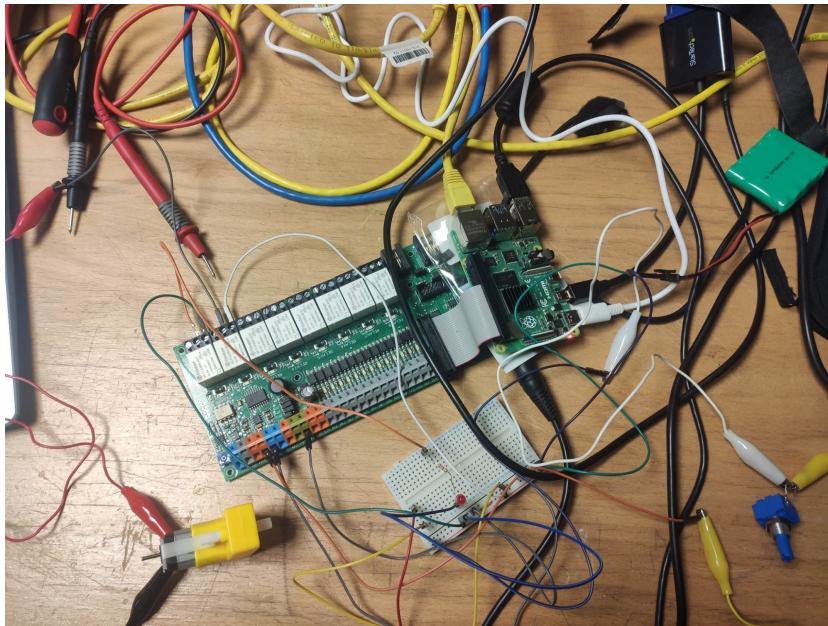


Figure 27: Montage de la solution

3 Mise en place de la solution Unipi sans API (EVOK) et sans Mervis:

3.1 Description de la solution

Dans cette solution, nous n'aurons pas besoin d'utiliser Mervis IDE. Au lieu de cela, nous utiliserons la documentation fournie par la société UNIPI. Nous utiliserons la carte de brochage Raspberry Pi (– Raspberry Pi P1 header map) , qui décrit chaque entrée/sortie Unipi à quelle fonction RPi BCM elles sont connectées. Une fois que nous avons notre carte de brochage, nous pouvons utiliser la carte d'extension Unipi de la même manière que la Raspberry Pi pour contrôler les entrées et sorties. Nous pouvons utiliser différents langages de programmation tels que C, Python et Shell.

3.2 Implémentation de la solution

3.2.1 Schémas de branchement des broches GPIO de la Raspberry Pi

Dans notre solution, nous avons utilisé C pour contrôler nos sorties et entrées numériques. Cependant, pour contrôler les 8 relais, nous devons ajouter un MCP23008 à notre Raspberry Pi.

Unipi	RPi BCM	Function	Description
AO	GPIO18	PWM	Analog Output 0-10V
I01	GPIO04	Digital Input	Digital Input
I02	GPIO17	Digital Input	Digital Input
I03	GPIO27	Digital Input	Digital Input
I04	GPIO23	Digital Input	Digital Input
I05	GPIO22	Digital Input	Digital Input
I06	GPIO24	Digital Input	Digital Input
I07	GPIO11	Digital Input	Digital Input
I08	GPIO07	Digital Input	Digital Input
I09	GPIO08	Digital Input	Digital Input
I10	GPIO09	Digital Input	Digital Input
I11	GPIO25	Digital Input	Digital Input
I12	GPIO10	Digital Input	Digital Input
I2C1_SCL	GPIO02	I2C1_SCL	Internal I ² C_1, RJ11 connector
I2C1_SDA	GPIO03	I2C1_SDA	
UART RX	GPIO15	UART0_RXD	UART RJ11 connector
UART TX	GPIO14	UART0_TXD	

Figure 28: Raspberry Pi P1 header map

Unipi	RPi BCM	Function	Description
I13	GPIO31	Digital Input	Digital Input
I14	GPIO30	Digital Input	Digital Input
I2C0_SCL	GPIO29	I2C0_SCL	External I ² C_1
I2C0_SDA	GPIO28	I2C0_SDA	

Figure 29: Raspberry Pi P1 header map

Relay	MCP23008
8	GP0
7	GP1
6	GP2
5	GP3
4	GP4
3	GP5
2	GP6
1	GP7

Figure 30: MCP23008 pin map

3.2.2 Mise en œuvre du contrôle des broches GPIO de la Raspberry Pi en C

Le code fourni utilise la bibliothèque WiringPi pour accéder aux broches GPIO du Raspberry Pi et définit les broches GPIO04 et GPIO18 comme entrée numérique et sortie PWM respectivement. Ensuite, il exécute une boucle infinie qui lit l'état de l'entrée numérique et met à jour la sortie PWM en fonction de cet état. Si l'entrée numérique est à l'état haut, la sortie PWM est mise à 100%, sinon elle est éteinte. Pour éviter que le processeur ne soit saturé en attendant la prochaine lecture de l'entrée numérique, une petite pause avec la fonction delay est utilisée.

```
#include <wiringPi.h>

#define INPUT_PIN 4
#define OUTPUT_PIN 1

int main(void) {
    if (wiringPiSetup() == -1) {
        return 1;
    }

    pinMode(INPUT_PIN, INPUT);
    pinMode(OUTPUT_PIN, PWM_OUTPUT);

    while(1) {
        if (digitalRead(INPUT_PIN) == HIGH) {
            pwmWrite(OUTPUT_PIN, 1023); // Allume la sortie à 100%
        } else {
            pwmWrite(OUTPUT_PIN, 0); // Éteint la sortie
        }
        delay(10); // Petite pause pour éviter de saturer le processeur
    }
    return 0;
}
```

3.2.3 Mise en œuvre du contrôle des broches du MCP23008 en C

Pour contrôler les 8 relais de notre Unipi à l'aide d'un Raspberry Pi, nous pouvons utiliser un MCP23008 pour étendre le nombre de broches GPIO disponibles. Le MCP23008 est un port d'extension I/O qui utilise le protocole I2C pour communiquer avec le Raspberry Pi. Les branchements sont les suivants :

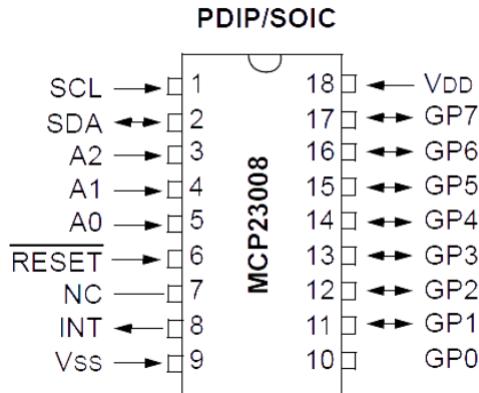


Figure 31: -MCP23008 pinout

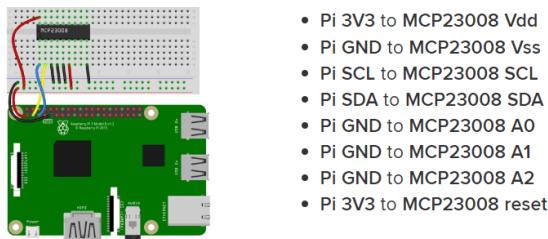


Figure 32: Schéma de câblage du MCP23008 sur le Raspberry Pi en I2C

Le premier code fourni utilise la bibliothèque WiringPi pour accéder à la broche GPIO04 du Raspberry Pi. Il configure également la broche GP6 (relais 2 de l'Unipi) comme sortie. Ensuite, il exécute une boucle infinie qui lit l'état de la broche GPIO04 et met à jour la sortie GP6 en conséquence. Si la broche GPIO04 est à l'état haut, la sortie GP6 (relais 2 de unipi) est mise à 1, sinon elle est mise à 0.

Code 1 : Bibliothèque WiringPi

```
#include <wiringPi.h>
#include <wiringPiI2C.h>

#define MCP23008_I2C_ADDRESS 0x20 // I2C address of MCP23008
#define MCP23008_GPIO_REG_ADDR 0x09 // GPIO register address of MCP23008

int main() {
    int mcp23008_fd = wiringPiI2CSetup(MCP23008_I2C_ADDRESS); // Open connection to MCP23008
    if (mcp23008_fd < 0) {
        printf("Failed to open connection to MCP23008\n");
    }
}
```

```

        return -1;
    }

wiringPiI2CWriteReg8(mcp23008_fd, 0x00, 0x00); // Set all MCP23008 GPIO pins as ou

while (1) {
    int gpio04_val = digitalRead(4); // Read value of GPIO04
    if (gpio04_val == HIGH) {
        wiringPiI2CWriteReg8(mcp23008_fd, MCP23008_GPIO_REG_ADDR, 0x40); // Set GP6 of
    } else {
        wiringPiI2CWriteReg8(mcp23008_fd, MCP23008_GPIO_REG_ADDR, 0x00); // Set GP6 of
    }
    delay(1000);
}
return 0;
}

```

Le deuxième code fourni utilise la bibliothèque Adafruit MCP23008 pour accéder à la broche GPIO04 du Raspberry Pi. Il initialise le circuit MCP23008 et configure la broche GP6 (relais 2 de l'Unipi) comme sortie. Ensuite, il exécute une boucle infinie qui lit l'état de la broche GPIO04 et met à jour la sortie GP6 en conséquence en utilisant les fonctions de la bibliothèque Adafruit MCP23008. Si la broche GPIO04 est à l'état haut, la sortie GP6 (relais 2 de l'Unipi) est mise à 1, sinon elle est mise à 0.

- Code 2 : Bibliothèque Adafruit MCP23008

```

#include <stdio.h>
#include <unistd.h>
#include "Adafruit_MCP23008.h"

#define MCP23008_I2C_ADDRESS 0x20 // I2C address of MCP23008

int main() {
    Adafruit_MCP23008 mcp;
    mcp.begin(MCP23008_I2C_ADDRESS); // Initialize MCP23008

    mcp.pinMode(6, OUTPUT); // Set GP6 of MCP23008 as output

    while (1) {
        int gpio04_val = digitalRead(4); // Read value of GPIO04
        if (gpio04_val == HIGH) {
            mcp.digitalWrite(6, HIGH); // Set GP6 of MCP23008 to HIGH to turn on relay
        } else {
            mcp.digitalWrite(6, LOW); // Set GP6 of MCP23008 to LOW to turn off relay
        }
    }
}

```

```

    }
    delay(1000); // Wait 1 second before reading again
}
return 0;
}

```

3.3 Limitations de la solution Unipi sans l'utilisation de l'API EVOK ni de Mervis

Cette solution présente tout de même quelques inconvénients. Tout d'abord, l'utilisation de la documentation fournie par la société Unipi peut être considérée comme moins intuitive que l'utilisation d'un IDE tel que Mervis. Ensuite, l'ajout d'un MCP23008 pour contrôler les 8 relais nécessite des connaissances en électronique et peut être compliqué pour les débutants. Enfin, le choix de la langue de programmation peut être un frein pour certains utilisateurs qui n'ont pas les compétences requises dans un langage spécifique, notamment en C.

Dans cette solution, il est possible d'utiliser les protocoles I2C et UART pour communiquer avec les périphériques connectés à notre Unipi. L'I2C peut être utilisé pour communiquer avec le MCP23008 qui nous permet de contrôler les 8 relais supplémentaires que nous avons ajoutés. Quant à l'UART, il peut être utilisé pour communiquer avec les capteurs connectés à notre Unipi. Cependant, cela peut prendre du temps pour les mettre en place et les configurer correctement.

En effet, la mise en place d'une telle solution à partir de zéro peut nécessiter plus de temps et de ressources que l'utilisation d'une plateforme de développement existante telle que Mervis ou l'API EVOK.

Il est important de noter que nous ne disposons pas des avantages de l'utilisation des outils de développement Mervis IDE et de l'API EVOK, qui peuvent simplifier le développement et le déploiement de solutions sur l'Unipi. Cependant, cela peut également offrir une plus grande flexibilité et une meilleure intégration avec les besoins spécifiques du projet.

4 Utilisation des solutions Unipi API

Lors de cette solution, nous utiliserons l'API EVOK, les APIs “Application Programming Interface” ou interface de programmation d'application permettent d'offrir des services à d'autres logiciels. L'API EVOK permet un accès à distance aux unités PLC (contrôleurs logiques programmables)

développées par UniPi. L'objectif principal de l'API EVOK est de simplifier l'accès au matériel sans nécessiter une programmation complexe.



Figure 33: APIs proposées par Unipi

Unipi software propose 3 APIs différentes:

- EVOK
- MODBUS
- SysFS

Les APIs MODBUS et SysFS sont plutôt orientés pour les “Vrai” PLCs et ne sont pas compatibles avec notre Unipi 1.1. Les 3 APIs sont des packages Unipi, contrairement à Mervis qui est un OS à part entière.

Il existe également des solutions tierces mais elles ne sont pas supportées officiellement pour la plupart d'entre elles. Nous avons choisi d'utiliser l'API EVOK. Elle est conçue pour permettre aux utilisateurs d'utiliser presque n'importe quel langage de programmation en utilisant six protocoles (ou méthodes unifiées) qui remplacent le besoin d'écrire un code personnalisé.

Cette API propose elle-même 7 protocoles différents :

- REST WebForms
- Bulk JSON
- REST JSON
- SOAP
- WebSocket
- JSON-RPC

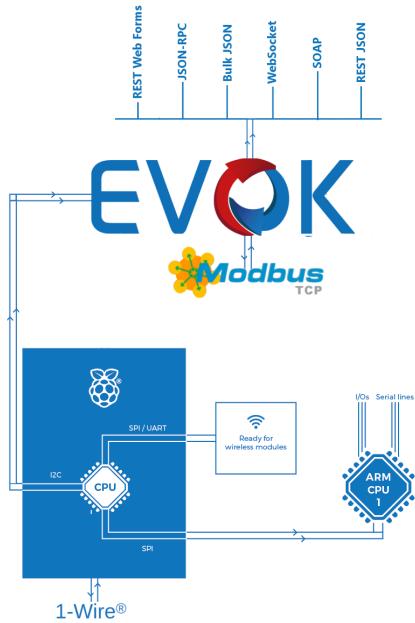


Figure 34: Architecture EVOK

Ils ont chacun leurs avantages et leurs inconvénients en fonction du contexte. Nous avons longement hésité entre le protocole REST WebForms et le JSON-RPC.

Nous avons finalement opté pour le JSON-RPC car il est plus simple à implementer au vu des méthodes qui sont déjà écrites et documentées. De plus, il est efficace pour définir des sorties au cas par cas, ce qui correspond bien à notre projet.

Mais le l'API REST est une bonne option et l'appel de toutes les entrées à la fois et la définition de plusieurs sorties en un seul appel sont des fonctionnalités intéressantes.

4.1 Configuration de notre réseau

Lors de notre utilisation, nous avons utilisé 2 PC et la raspberry connecté à la Unipi. Les 3 appareils se trouvent sur le même réseau isolé, relié par un commutateur. Nous devions télécharger les packets EVOK et souhaitions avoir notre raspberry à jour donc il nous fallait partager une connexion au réseau à partir de l'un des PC. Nous avons utilisé le partage de connexion windows, mais celui-ci impose une IP : 192.168.137.1

La configuration du réseau est la suivante :

- 192.168.137.1 : PC 1 qui partage EDUROAM WiFi au réseau
- 192.168.137.2 : PC 2
- 192.168.137.3 : Unipi

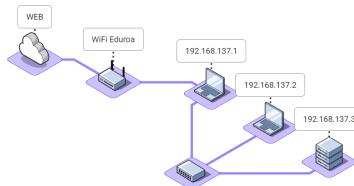


Figure 35: Configuration réseau

Il faut donc préparer le raspberry et mettre en place cette configuration avant d'installer quoique ce soit.

Les étapes d'installations sont les suivantes :

- Installer un OS raspbian avec interface graphique et activer le ssh.
- Retrouver l'adresse IP du raspberry (commandes ARP, IP scanner, ...)
- Se connecter en SSH au raspberry et changer l'adresse IP du raspberry pour l'adresse 192.168.137.3, en statique. Il faut également changer l'adresse du routeur pour notre futur adresse 192.168.137.1. Cela se fait de manière pérène en modifiant le fichier /etc/dhcpcd.conf comme ci dessous :
- Redemarrer la raspberry

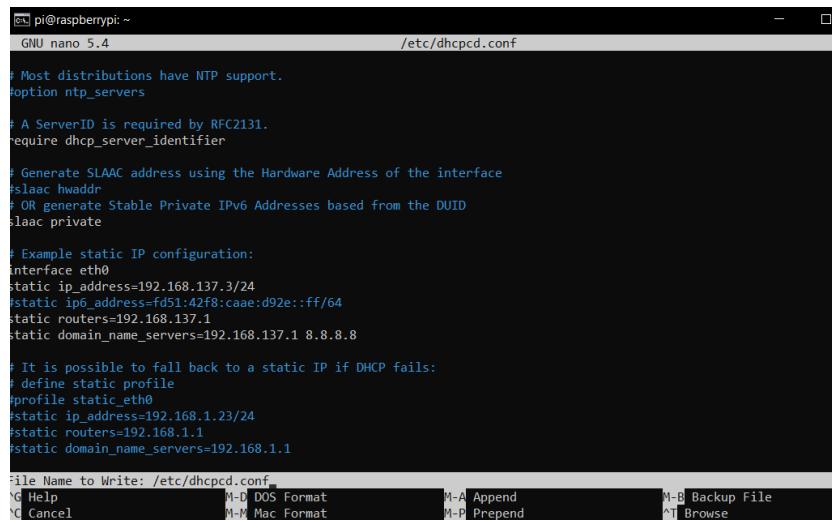
Il faut à présent changer les adresses des PCs pour des addressens en 192.168.137.X et partager la connexion WiFi avec l'interface ethernet.

Cela se fait dans Panneau de configuration\Réseau et Internet\Connexions réseau

- Une fois que les adresses IP sont fixé et que la connexion est partagée, nous pouvons nous connecter à la raspberry et ping une adresse sur le web : le partage fonctionne.
- Nous mettons à jour le système : sudo apt update && sudo apt upgrade

Afin d'avoir un accès avec une interface graphique, on peut configurer un serveur VNC. Cela se fait avec la commande sudo raspi-config. Cette étape n'est pas indispensable mais un serveur VNC peut être pratique pour travailler à distance sur la RaspberryPi.

La raspberry est prête, nous pouvons installer EVOK !



```

pi@raspberrypi: ~
GNU nano 5.4                               /etc/dhcpcd.conf

# Most distributions have NTP support.
option ntp_servers

# A ServerID is required by RFC2131.
require dhcp_server_identifier

# Generate SLAAC address using the Hardware Address of the interface
#slaac_hwaddr
# OR generate Stable Private IPv6 Addresses based from the DUID
#slaac_private

# Example static IP configuration:
interface eth0
static ip_address=192.168.137.3/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.137.1
static domain_name_servers=192.168.137.1 8.8.8.8

# It is possible to fall back to a static IP if DHCP fails:
# define static profile
#profile static_eth0
#static ip_address=192.168.1.23/24
#static routers=192.168.1.1
#static domain_name_servers=192.168.1.1

File Name to Write: /etc/dhcpcd.conf
G Help          M-D DOS Format    M-A Append      M-B Backup File
C Cancel        M-M Mac Format    M-P Prepend    M-T Browse

```

Figure 36: Configuration réseau du raspberrypi

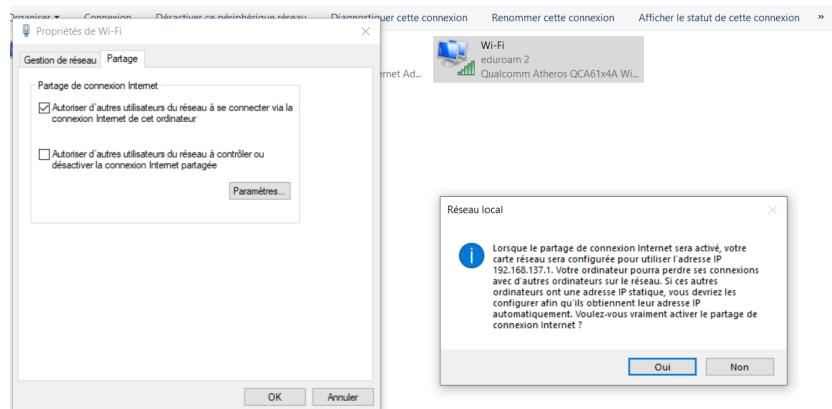


Figure 37: Partage réseau windwos

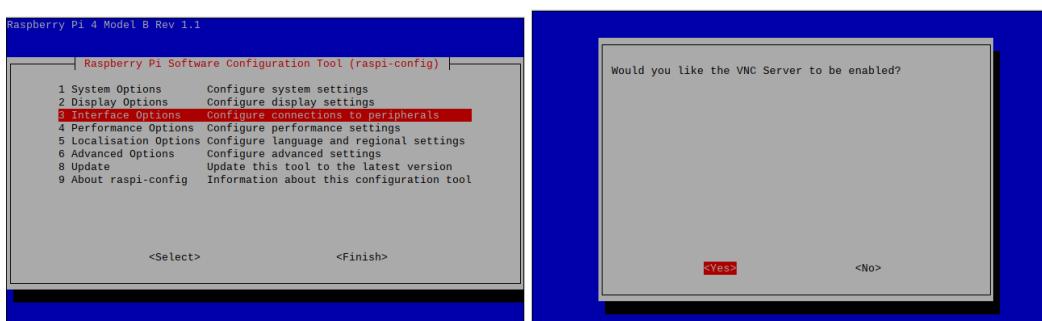


Figure 38: Activation de VNC

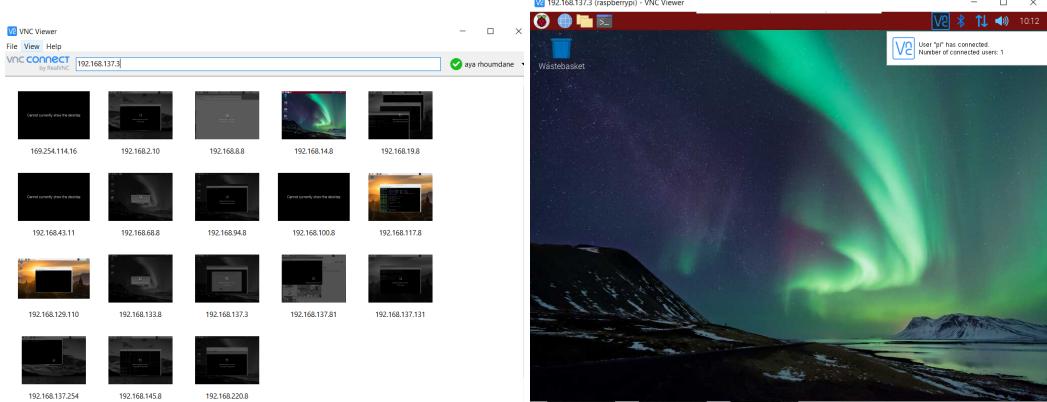


Figure 39: Connexion avec VNC

4.2 Installation d'EVOK

L'installation d'EVOK est plutôt simple. Il suffit de télécharger l'archive, de la décompresser et de lancer le script d'installation (le tout en root).

```
sudo su
wget https://github.com/UniPiTechnology/evok/archive/2.4.24.zip
unzip 2.4.24.zip
```

Notez que nous utilisons ici une archive et non la dernière release car celle-ci ne fonctionnait pas lorsque nous l'avions essayé.

```
cd evok-2.4.24
./install-evok.sh
```

EVOK s'installe alors et nous demandes des paramètres de configuration :

Le port du site web : 80 [à changer si vous souhaitez utiliser un autre serveur WEB sur le même port]

Le port de l'API : 8080

Le modèle du Unipi : UniPi Lite 1.x OU UniPi 1.x. Dans notre cas, nous choisissons UniPi 1.X.

Sur les dernières versions, il nous propose également d'installer le WiFi, nous acceptons pour utiliser le VNC plus simplement.

Une fois l'installation finie, le script affiche :

Il faut redémarrer.

```

#####
## Evok installed sucessfully. ##
## Info: ##
## 1. Edit /etc/evok.conf file ##
## according to your choice. ##
## If you are running Apache or ##
## other daemon at port 80, you ##
## must set either evok or ##
## apache port different than ##
## the other. ##
## 2. Run "service evok" ##
## [start|restart|stop]" to ##
## control the daemon. ##
## (3. To uninstall evok run ##
## /opt/evok/uninstall-evok.sh) ##
#####
Is it OK to reboot now? [y/n]

```

Figure 40: Résultat de l'installation d'EVOK

4.3 Utilisation d'EVOK

Une fois le système redémarré, EVOK est installé et nous pouvons utiliser les APIs.

Voici un petit test qui utilise le protocole REST WebForms :

```
curl --request POST --url http://192.168.137.3:80/rest/relay/1 -d"value=1"
```

Nous pouvons également accéder au panneau de contrôle Unipi en `http://192.168.137.3/` et pouvons accéder à toutes les variables d'un coup en tapant `http://192.168.137.3/rest/all`

Les requêtes REST sont les suivantes :

Digital Input (get)(1-12)

```
curl --request GET --url http://192.168.137.3:80/rest/input/1
```

Analog Input (get) (1-2)

```
curl --request GET --url http://192.168.137.3:80/rest/ai/1
```

Relais (post)(1-8)

```
curl --request POST --url http://192.168.137.3:80/rest/relay/1 -d"value=0"
```

Relais (get)(1-8)

```
curl --request GET --url http://192.168.137.3:80/rest/relay/1
```

Analog Output (post) (1)

```
curl --request POST --url http://192.168.137.3:80/rest/ao/1 -d "value=0"
```

Analog Output (get) (1)

```
curl --request GET --url http://192.168.137.3:80/rest/ao/1
```

Pour les sorties analogiques, nous pouvons mettre une valeur supérieure à 10V mais Unipi restreint la sortie à 10v.

Evok et tous ces protocoles sont à présent installés, mais il n'y a absolument pas de sécurité et EVOOK ne requiert aucun mot de passe et ne fait aucunes distinctions. Ce qui signifie que n'importe qui sur le réseau peut modifier l'état des entrées et des sorties de notre carte.

Pour faire en sorte que la Unipi n'écoute pas les requêtes extérieures, il faut modifier la ligne 1968 du fichier evok.py.

Et changer:

```
httpServer.listen(port)
```

en

```
httpServer.listen(port, address="127.0.0.1")
```

De cette manière, aucun autre appareil que le raspberry utilise l'API.

5 La solution proposé

Pour la réalisation de notre solution, nous avons besoin de plusieurs services différents : Gestion des compresseurs, site web pour l'interface homme machine, base de données.

Pour des raisons de disponibilité serveur, nous avons fait le choix de séparer au maximum notre architecture logicielle. Nous avons donc besoin d'écrire 2 programmes qui interagissent entre eux pour créer une base de données. Pour des questions de simplicité, nous avons choisi d'écrire ces programmes en python et d'utiliser une base de données SQL.

Nous avons donc mis en place une architecture tel que représenté sur le schéma :

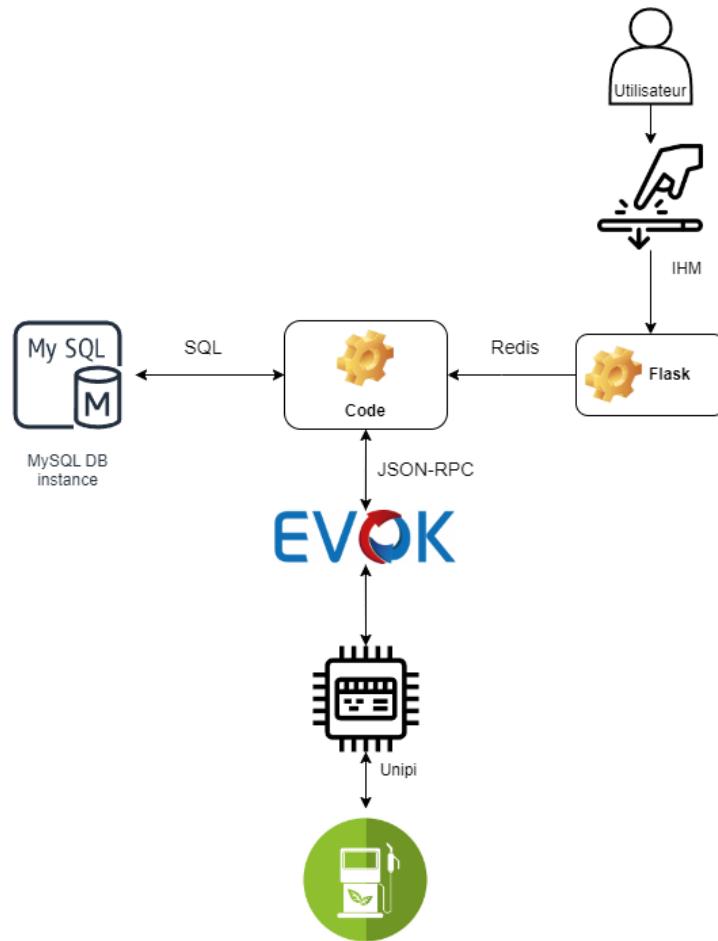


Figure 41: Architecture logicielle

5.1 Json RPC

L'utilisation du protocole Json Remote Procedure Call est assez simple et est documenté par Unipi. Evok fournit une liste de devices ainsi qu'une liste de méthodes, incluant des getters et des setters. Il faut créer une instance de server RPC et lui executer les méthodes voulu :

Liste des devices utilisables :

- relay - Relais
- input or di - digital input - Entrées digitales
- ai - analog input - Entrées Analogiques
- ao - analog output - Sorties Analogiques
- ee - onboard eeprom - Stockage eeprom de la carte

- sensor - 1wire sensor - Bus 1 wire pour des capteurs annexes

Notez que la numérotation des devices commence à 1 et non à 0.

Code python exemple :

```
from jsonrpclib import Server # Import de JSON-RPC
s=Server("http://adresse-ip/rpc")# instance du serveur à partir de l'adresse IP
#du raspberrypi
s.relay\_set(1,1) # le premier relay prend la valeur 1
s.relay\_get(1)
s.relay\_set(1,0)
s.relay\_get(0) # le premier relay prend la valeur 1
s.ai\_get(1)
```

5.2 La base de données

Il nous faut télécharger et installer mysql :

```
sudo apt-get install mysql-server
sudo mysql\_\_secure\_\_installation
```

On démarre mysql et on l'active au démarrage

```
sudo systemctl start mysql
sudo systemctl enable mysql
```

Une fois que MariaDB fonctionne, on peut se créer une table et un utilisateur.

```
mysql -u root -p
create database Test;
create user userex@localhost identified by 'mdp';
GRANT ALL ON Test.* TO userex@localhost;
FLUSH PRIVILEGES;
```

La base de données est maintenant prête à être utilisée.

On crée un petit programme python pour l'essayer :

```
import mysql.connector as mysql

db = mysql.connect(
host = "localhost",
user = "userex",
passwd = "mdp")

print(db)
```

```
pi@raspberrypi:~/DB $ python dbexemple.py
<mysql.connector.connection.MySQLConnection object at 0xb671ebc8>
```

Figure 42: Conneion à la BD à partir du code Python

Nous pouvons maintenant intéragir avec la base de données.

Nous pouvons ensuite créer une base de données et l'alimenter avec les requetes suivantes :

Création :

```
CREATE TABLE `infos`  
(  
    date\_time datetime NOT NULL PRIMARY KEY DEFAULT CURRENT\_TIMESTAMP,  
    km int(10) NOT NULL,  
    Nplace int(2) NOT NULL,  
    Nvehicule int(2) NOT NULL  
) ;
```

Des entiers de 2 suffiront pour le nombre de place et le nombre de véhicules

Alimentation :

```
INSERT INTO infos (km,Nplace,Nvehicule)  
VALUES (123456,1,1)
```

Lors de la récupération des informations, nous nous apercevons que le système est à l'heure Anglaise :

```
pi@raspberrypi:~/DB $ python dbexemple.py
La BDD est :<mysql.connector.connection.MySQLConnection object at 0xb660bc10>
(datetime.datetime(2023, 3, 15, 15, 17, 51), 123456, 1, 1)
(datetime.datetime(2023, 3, 15, 15, 18, 26), 123456, 1, 1)
(datetime.datetime(2023, 3, 15, 15, 21, 48), 123456, 1, 1)
pi@raspberrypi:~/DB $ timedatectl
        Local time: Wed 2023-03-15 15:30:32 GMT
        Universal time: Wed 2023-03-15 15:30:32 UTC
                  RTC time: Wed 2023-03-15 15:30:33
                 Time zone: Europe/London (GMT, +0000)
System clock synchronized: yes
          NTP service: active
      RTC in local TZ: no
pi@raspberrypi:~/DB $
```

Figure 43: Test de la Base de données

Cela se change assez aisément avec timedatectl.

```

pi@raspberrypi:~/DB $ timedatectl set-timezone Europe/Paris
==== AUTHENTICATING FOR org.freedesktop.timedate1.set-timezone ===
Authentication is required to set the system timezone.
Authenticating as: ,,(pi)
Password:
==== AUTHENTICATION COMPLETE ===
pi@raspberrypi:~/DB $ timedatectl
          Local time: Wed 2023-03-15 16:32:30 CET
              Universal time: Wed 2023-03-15 15:32:30 UTC
                    RTC time: Wed 2023-03-15 15:32:30
                   Time zone: Europe/Paris (CET, +0100)
     System clock synchronized: yes
          NTP service: active
    RTC in local TZ: no

```

Figure 44: Changement de fuseau horaire

5.3 Flask

Pour l'IHM, nous avons choisi d'utiliser une interface web gérée par Flask. Flask est un framework simple de développement web en Python. Il requiert au moins une page html et un code python.

Nous avons développé une page web d'ébauche :

The screenshot shows a web-based form for managing vehicle data. At the top, there are two input fields: "N du véhicule ?" and "Kilometrage du véhicule ?". Below these is a numeric keypad with a grid of numbers from 1 to 9. Underneath the keypad are four buttons: "Supprimer" (blue), "0" (red), "Effacer" (red), and "Valider" (green). A note below the keypad says "Please choose a parking spot:". Below this, there is a grid of 11 red rectangular buttons labeled "Spot 1" through "Spot 11".

Figure 45: Page web de renseignement des données

Les données sont saisies à travers 2 text boxes et des radio buttons simulant les places de parking. Elles sont transmises à Flask à l'aide d'une requête POST. Une fois que l'opérateur a saisi ses données, il est améné sur une seconde page qui lui demande de valider (ou d'invalider) ces données déjà transmises afin de lancer le processus de recharge.

Avant de lancer Flask, il faut exporter la variables d'environnement FLASK_APP pour que Flask ait la connaissance du nom de l'application à faire fonctionner

Sur Linux :

```
export FLASK_APP=serv-flask.py
```

serv-flask.py est à modifier en fonction du nom de votre fichier, bien sûr.

Sur windows :

```
$env:FLASK_APP = "serv-flask.py"
```

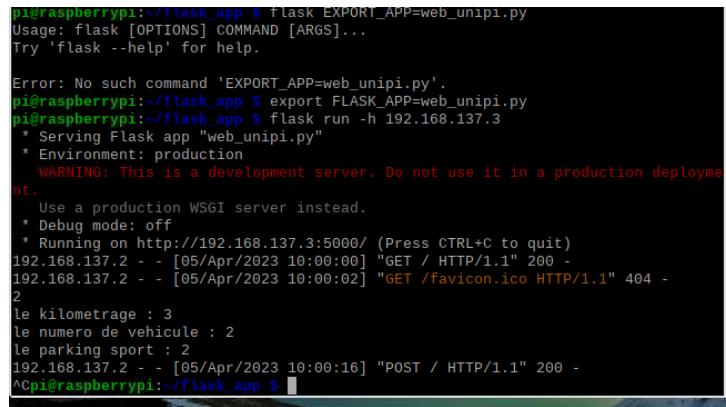
Si vous souhaitez le lancer en mode développement avec des paramètres de debug, il faut figer la variable FLASK_ENV en mode ‘development’.

```
export FLASK_ENV=development
```

Une fois ces variables définies, lancer le serveur flask se fait à l'aide de la commande :

```
flask run
```

Il faut préciser l'IP de la raspberry avec l'option -h (host) si vous souhaitez que Flask écoute tous les IPs.



```
pi@raspberrypi:~/flask_app $ flask run -h 192.168.137.3
Usage: flask [OPTIONS] COMMAND [ARGS]...
Try 'flask --help' for help.

Error: No such command 'EXPORT_APP=web_unipi.py'.
pi@raspberrypi:~/flask_app $ export FLASK_APP=web_unipi.py
pi@raspberrypi:~/flask_app $ flask run -h 192.168.137.3
  * Serving Flask app "web_unipi.py"
  * Environment: production
    WARNING: This is a development server. Do not use it in a production deployment.
    Use a production WSGI server instead.
  * Debug mode: off
  * Running on http://192.168.137.3:5000/ (Press CTRL+C to quit)
192.168.137.2 - - [05/Apr/2023 10:00:00] "GET / HTTP/1.1" 200 -
192.168.137.2 - - [05/Apr/2023 10:00:02] "GET /favicon.ico HTTP/1.1" 404 -
2
le kilometrage : 3
le numero de vehicule : 2
le parking sport : 2
192.168.137.2 - - [05/Apr/2023 10:00:16] "POST / HTTP/1.1" 200 -
^Cpi@raspberrypi:~/flask_app $
```

Figure 46: Lancement de Flask dans une interface en ligne de commande

5.4 Communication entre les scripts python à l'aide de Redis

5.4.1 Description de la solution

Une façon d'assurer une communication en temps réel entre notre application Flask et notre deuxième script est d'utiliser une file de messages (message queue). Une file de messages est un composant logiciel qui permet à différentes parties d'une application de communiquer de manière asynchrone en envoyant et recevant des messages.

Il existe différents systèmes de file de messages que nous pouvons utiliser en Python, tels que RabbitMQ, Apache Kafka ou Redis, mais dans cette solution, nous utiliserons Redis.

Tout d'abord, nous devons installer la bibliothèque Redis pour Python :

```
pip install redis
```

Dans notre solution, nous utiliserons le client Redis pour publier un message sur un canal Redis nommé parking_data. Le message contient les valeurs des variables vehicule_number, kilometrage et parkingspot, séparées par des virgules.

```
if redis_client.ping():
    data = '{}{},{}{}'.format(vehicule_number,kilometrage,parkingspot)
    redis_client.publish('parking_data',data)
    print("Data published successfully")
else:
    print("Redis connection not ready")
```

Maintenant, dans notre deuxième script, nous pouvons nous abonner au canal Redis et recevoir les messages en temps réel. Lorsqu'un message est reçu, la fonction process_message est appelée pour extraire les informations utiles du message et les stocker dans trois variables distinctes: vehicule_number, kilometrage et parkingspot.

```
def process_message(message):
    logging.info('Received new message from Redis queue')
    data = message.decode('utf-8')
    vehicule_number, kilometrage, parkingspot = data.split(',')
    try:
        s.relay_set(int(parkingspot), 1)
        logging.info(f'Relay {parkingspot} turned on')
    except Exception as e:
        logging.error(f'Error turning on relay {parkingspot}: {e}')
```

Cette configuration permet une communication en temps réel entre notre application Flask et notre deuxième script, sans avoir besoin de stocker les données dans un fichier.

5.4.2 Configuration de Redis

Pour vérifier le fichier de configuration Redis, nous avons suivi les étapes suivantes :

- Nous avons ouvert le fichier de configuration Redis à l'aide d'un éditeur de texte. L'emplacement du fichier de configuration varie en fonction de votre méthode d'installation et de votre système d'exploitation, mais il est généralement nommé "redis.conf".
- Nous avons cherché la directive de configuration "bind", qui spécifie l'adresse IP sur laquelle Redis écoutera. Par défaut, elle est définie sur "127.0.0.1", ce qui signifie que Redis écoutera uniquement les connexions de la machine locale. Si nous voulons autoriser les connexions à partir d'autres machines, nous devons modifier cette directive pour l'adresse IP de notre machine ou la définir sur "0.0.0.0" pour écouter sur toutes les interfaces réseau disponibles. Par exemple : bind 0.0.0.0
- Nous avons cherché la directive de configuration "port", qui spécifie le port TCP sur lequel Redis écoutera. Par défaut, il est défini sur 6379. Si nous voulons utiliser un port différent, nous pouvons modifier cette directive. Par exemple : port 6380
- Nous avons vérifié les règles de notre pare-feu pour nous assurer qu'elles permettent les connexions entrantes au port Redis. Les étapes à suivre dépendront de notre système d'exploitation et du logiciel de pare-feu, mais en général, nous pouvons ouvrir le port à l'aide de l'outil de configuration du pare-feu ou en ajoutant une règle au fichier de configuration du pare-feu.

Une fois que nous avons apporté les modifications nécessaires au fichier de configuration Redis et aux règles de pare-feu, nous avons redémarré le serveur Redis pour appliquer les modifications.

Finalement pour vérifier si le serveur Redis s'exécute sur l'adresse IP et le numéro de port spécifiés, nous avons utilisé l'outil en ligne de commande redis-cli pour tester la connexion :

```
redis-cli -h 192.168.137.3 -p 6379 ping
```

5.4.3 Connexion à un serveur Redis distant

Pour se connecter à un serveur Redis exécuté sur une machine distante (c'est-à-dire, pas sur la même machine que notre application Flask), nous avons besoin de spécifier l'adresse IP ou le nom d'hôte de la machine qui exécute le serveur Redis à la place de 'localhost'. Nous avons donc ouvert notre fichier Python et ajouté la ligne suivante pour spécifier l'adresse IP et le port de notre serveur Redis distant :

```
redis_client = redis.Redis(host='192.168.137.3', port=6379)
```

Ensuite, pour vérifier si le serveur Redis est en cours d'exécution à l'adresse IP et au numéro de port spécifiés, on exécute la commande suivante :

```
redis-cli -h 192.168.137.3 -p 6379 ping
```

Si le serveur est en cours d'exécution, nous devrions voir la réponse "PONG". Si la réponse est "Could not connect", cela signifie que le serveur ne s'exécute pas. Si le serveur Redis ne s'exécute pas, nous pouvons le démarrer en exécutant la commande suivante :

```
redis-server
```

Conclusion et points d'amélioration

Parmi nos solutions, l'API EVOK se distingue par son excellence et sa praticité, surpassant de loin le système Mervis en termes de fonctionnalités et de flexibilité. En effet, la conception même de l'EVOK, qui repose sur un système Linux, permet une polyvalence inégalée avec Mervis OS, offrant ainsi un large éventail de fonctionnalités adaptées à divers besoins et applications. Cependant, pour mettre en place cette solution, un effort supplémentaire en termes de développement et d'installation est nécessaire, ce qui peut représenter un défi pour certains utilisateurs. Malgré cela, la qualité et l'efficacité de l'API EVOK en font des cartes Unipi, une solution de choix pour les professionnels et les entreprises cherchant à remplacer les automates traditionnels.

Les points d'amélioration identifiés pour cette solution sont nombreux et variés. Tout d'abord, il est important de développer de manière plus détaillée la gestion de la station de recharge pour en permettre l'utilisation. Ensuite, la sécurisation de la solution est primordiale, notamment en ce qui concerne les problèmes d'écoute IP et d'injections SQL, qui doivent être pris en compte et résolus de manière proactive. Cette sécurisation permettra de créer, une interface de connexion utilisateur/manager sécurisée afin d'ouvrir le site sur le web, tout en assurant la sécurité de l'accès. Enfin, l'implémentation de la solution sur le terrain et l'ouverture du site sur internet sont des étapes importantes pour rendre la solution accessible à un public plus large. En prenant en compte ces points d'amélioration, il est possible de garantir la fiabilité et la durabilité de la solution, tout en offrant une expérience utilisateur optimale.

Code final

Flask.py

```
1 # importing Flask and other modules
2 from flask import Flask, request, render_template, redirect,
3     url_for
4 import redis
5 redis_client = redis.Redis(host='192.168.137.3', port=6379)
6
7 # Flask constructor
8 app = Flask(__name__, template_folder='.')
9 global vehicule_number
10 global parkingspot
11 global kilometrage
12
13 #app = Flask(__name__)
14 # A decorator used to tell the application
15 # which URL is associated function
16 @app.route('/', methods=["GET", "POST"])
17 def test():
18     if request.method == "POST":
19         # getting input with name = fname in HTML form
20         vehicule_number = request.form.get("vehicule-
21         number_form")
22         # getting input with name = lname in HTML form
23         kilometrage = request.form.get("kilometrage_form")
24         parkingspot=request.form.get('spot',False)
25
26         print("le kilometrage : " + str(kilometrage)) #marche
27         print("le numero de vehicule : "+str(vehicule_number))
28         print("la place de parking : "+str(parkingspot))
29         return redirect('/valider.html?kilometrage={}&
30         vehicule_number={}&parkingspot={}'.format(kilometrage ,
31         vehicule_number , parkingspot))
32
33     return render_template("webapp.html")
34
35
36
37
38
39
40 @app.route('/valider.html',methods =["GET", "POST"])
41 def valider():
42     print("Validation")
43     kilometrage=request.args.get('kilometrage')
44     vehicule_number=request.args.get('vehicule_number')
45     parkingspot=request.args.get('parkingspot')
46     if request.method == "POST":
47         # getting input with name = fname in HTML form
48         print("POST")
```

```

41     bon = request.form.get("bon")
42     print(bon)
43     if bon == "true":
44         while True:
45             try:
46                 data = '{} ,{} ,{}' .format(vehicule_number ,
kilometrage , parkingspot)
47                     rcvd = redis_client.publish('parking_data',
, data)
48                         print("Data published successfully")
49                         if rcvd >0:
50                             break
51             except redis.ConnectionError:
52                 pass
53             # handle reconnect attempts
54         else:
55             print("Non valide")
56             return redirect(url_for('test'))
57         return render_template("valider.html",hkilometrage=
kilometrage ,hvehicule_number=vehicule_number ,hparkingspot
=parkingspot)
58
59 if __name__=='__main__':
60     print("passe")
61     app.run(host="0.0.0.0" , port=5000 , debug=True)

```

Source Code 1: Web app Flask

Gestion.py

```
1 import redis
2 import logging
3 from jsonrpclib import Server
4 from time import sleep
5 s=Server("http://192.168.137.3/rpc")
6 redis_client = redis.Redis()
7
8 global estdemarre
9 global aru
10 global DI # DigitalInputs inputs de 4 a 12. {0,1,2,3}
    existent mais ne sont pas utiles dans cette liste
11 DI=[]
12 DI = [0 for i in range(13)]
13 estdemarre = 0
14 global vehicule_number
15 global kilometrage
16 global parkingspot
17
18
19 ## Connecting to the database
20
21 ## importing 'mysql.connector' as mysql for convenient
22 import mysql.connector as mysql
23
24 ## connecting to the database using 'connect()' method
25 ## it takes 3 required parameters 'host', 'user', 'passwd'
26 db = mysql.connect(
27     host = "localhost",
28     user = "userex",
29     passwd = "mdp",
30     database="Test"
31 )
32 cursor = db.cursor()
33
34 # configure logging
35 logging.basicConfig(
36     format='%(asctime)s %(levelname)s: %(message)s',
37     datefmt='%Y-%m-%d %H:%M:%S',
38     level=logging.INFO
39 )
40 def NoARU(): #verifie si il y a des arrêts d'urgences
41     arug = s.input_get_value(1) # arrêt urgence
42     au1 = s.input_get_value(2) # arrêt urgence compresseur 1
43     au2 = s.input_get_value(3) # arrêt urgence compresseur 2
44     return not (arug and au1 and au2)
45
46 def Impulsion(numero,duree): #Cree une impulsion au relai
```

```

    avec son numero et sa duree
47     s.relay_set(numero,1)
48     sleep(duree)
49     s.relay_set(numero,0)
50
51 def RoutineDemarrage(): # routine de demarrage des
52     compressseurs
53     global aru
54     global estdemarkre
55     if aru := NoARU():
56         sleep(5)
57         Inpulsion(1,1) # impulsion de demarrage du
58         compresseur 1
59         sleep(15)
60         Inpulsion(2,1)# impulsion de demarrage du
61         compresseur 2
62         estdemarkre = 1
63
64 def RoutineExtinxion(): # Routine d'extinxion des
65     compressseurs
66     global estdemarkre
67     Inpulsion(3,1) # impulsion d'extinxion du compresseur 1
68     sleep(1)
69     Inpulsion(4,1) # impulsion d'extinxion du compresseur 1
70     estdemarkre = 0
71
72 def ReadInputs(): # Lis les DI non ARU
73     global DI
74     for i in range(4,13): #[de 4 a 12]
75         DI[i] = s.input_get_value(i)
76
77 def process_message(message):
78     logging.info('Received new message from Redis queue')
79     data = message.decode('utf-8')
80     vehicule_number, kilometrage, parkingspot = data.split(',')
81     cursor.execute(f'INSERT INTO infos (km,Nplace,Nvehicule)
82     VALUES ({kilometrage},{parkingspot},{vehicule_number});')
83     db.commit()
84     try:
85         s.relay_set(int(parkingspot), 1)
86         logging.info(f"Relay {parkingspot} turned on")
87     except Exception as e:
88         logging.error(f"Error turning on relay {parkingspot}
89         }: {e}")
90
91 pubsub = redis_client.pubsub()
92 pubsub.subscribe('parking_data')

```

```

87
88
89 while True:
90     try:
91         message = pubsub.get_message()
92     except redis.ConnectionError:
93         # Do reconnection attempts here such as sleeping and
94         # retrying
95         pubsub = redis.pubsub()
96         pubsub.subscribe('parking_data')
97
98     if message:
99         print(message)
100        print(type(message))
101        if message['type']=='message': ## verifie si le
102            message est bien un message
103            process_message(message['data'])
104
105        # do something withthe message
106    else:
107        print("pas de message")
108        sleep(0.05) # be nice to the system :)
109
109 db.close()

```

Source Code 2: Gestion de la station

Page Html principale

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title>Projet UNIPI</title>
6   <style>
7     input[type="radio"]:checked+label {
8       background-color: blue;
9       color: white;
10      transform: scale(1.2);
11      transition: all 0.2s ease-in-out;
12    }
13
14   .parking-spots {
15     display: flex;
16     flex-wrap: wrap;
17   }
18
19   .parking-spot {
20     display: inline-block;
21     margin-right: 10px;
22   }
23
24   div {
25     display: flex;
26     position: relative;
27   }
28
29   label {
30     display: flex;
31     justify-content: center;
32     align-items: center;
33     width: 100px;
34     height: 100px;
35     background: red;
36     color: white;
37     border-radius: 10px;
38     margin: 0 10px 0 0;
39     cursor: pointer;
40   }
41
42   input:checked+label {
43     background: blue;
44   }
45 </style>
46 </head>
47
48 <body>
49   <div style="display: inline-block; margin-right: 48px;">
50     <h3>N du vehicule ?</h3>
51   </div>
52   <div style="display: inline-block;">
53     <h3> Kilometrage du vehicule ?</h3>
54   </div>
55   <div>
56     <input type="text" id="vehicle-number" autofocus="autofocus" readonly>
57     <input type="text" id="kilometrage" readonly>
58     <br>
59   </div>
60   <br>
61   <div style="margin-left: 65px;">
62     <button onclick="addNumber(1)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">1</button>
63     <button onclick="addNumber(2)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">2</button>
64     <button onclick="addNumber(3)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">3</button>
65
66   </div>
67   <div style="margin-left: 65px;">
68     <button onclick="addNumber(4)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">4</button>
69     <button onclick="addNumber(5)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">5</button>
```

```

76   <button onclick="addNumber(6)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">6</button>
77
78 </div>
79 <div style="margin-left: 65px;">
80   <button onclick="addNumber(7)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">7</button>
81   <button onclick="addNumber(8)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">8</button>
82   <button onclick="addNumber(9)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">9</button>
83
84 </div>
85 <div style="margin-top: 5px;">
86   <button onclick="deleteNumber()" style="background-color: #F4D03F; color: white; font-size: 16px; padding: 10px 20px ;">Supprimer</button>
87   <button onclick="addNumber(0)" style="background-color: blue; color: white; font-size: 16px; padding: 10px 20px ;">0</button>
88   <button onclick="clearNumber()" style="background-color: red; color: white; font-size: 16px; padding: 10px 20px ;">Effacer</button>
89   <button onclick="validateNumber()" style="background-color: green; color: white; font-size: 16px; padding: 10px 20px ;">Valider</button>
90
91 </div>
92 <script>
93   let vehicleNumberInput = document.getElementById("vehicle-number");
94   let kilometrageInput = document.getElementById("kilometrage");
95   let focusedInput = null;
96
97   vehicleNumberInput.addEventListener("focus", function () {
98     focusedInput = vehicleNumberInput;
99   });
100
101  kilometrageInput.addEventListener("focus", function () {
102    focusedInput = kilometrageInput;
103  });
104
105  function addNumber(num) {
106    if (focusedInput !== null) {
107      focusedInput.value += num;
108    }
109  }
110
111  function clearNumber() {
112    if (focusedInput !== null && focusedInput.value.length > 0) {
113      focusedInput.value = "";
114    }
115  }
116
117  function deleteNumber() {
118    if (focusedInput !== null && focusedInput.value.length > 0) {
119      focusedInput.value = focusedInput.value.slice(0, -1);
120    }
121  }
122
123  function validateNumber() {
124    if (vehicleNumberInput.value.length > 0 && kilometrageInput.value.length > 0) {
125      //mettre les variables km et Nvehicule dans des inputs cache du form
126      document.getElementById("kilometrage_form").value = kilometrageInput.value;
127      document.getElementById("vehicule-number_form").value = vehicleNumberInput.value;
128
129      //envoyer le form
130      document.getElementById("myForm").submit();
131      alert("Vehicle Number " + vehicleNumberInput.value + " and Kilometrage " +
132            kilometrageInput.value + " validated.");
133    } else {
134      alert("Please enter a valid 8-digit vehicle number and kilometrage.");
135    } // action="{{ url_for("test") }}" en dessous lol
136  }
137
138 </script>
139
140
141
142
143
144
145
146
147
148

```

```

149 <form id="myForm" method="post">
150   <h3>Please choose a parking spot:</h3>
151   <! 2 inputs hidden pour les envoyer dans le form -->
152   <input type="hidden" name="kilometrage_form" id="kilometrage_form">
153   <input type="hidden" name="vehicule-number_form" id="vehicule-number_form">
154
155   <! premiere ligne de radio buttons -->
156   <div class="parking-spots" style="margin-left: 170px;">
157     <div class="parking-spot">
158       <input type="radio" id="spot1" name="spot" value="1" style="display:none;">
159       <label for="spot1">Spot 1</label>
160     </div>
161     <div class="parking-spot">
162       <input type="radio" id="spot2" name="spot" value="2" style="display:none;">
163       <label for="spot2">Spot 2</label>
164     </div>
165     <div class="parking-spot">
166       <input type="radio" id="spot3" name="spot" value="3" style="display:none;">
167       <label for="spot3">Spot 3</label>
168     </div>
169     <div class="parking-spot">
170       <input type="radio" id="spot4" name="spot" value="4" style="display:none;">
171       <label for="spot4">Spot 4</label>
172     </div>
173   </div>
174
175   <! deuxieme ligne-->
176   <div class="parking-spots" style="margin-top: 10px;">
177     <div class="parking-spot">
178       <input type="radio" id="spot5" name="spot" value="5" style="display:none;">
179       <label for="spot5">Spot 5</label>
180     </div>
181     <div class="parking-spot">
182       <input type="radio" id="spot6" name="spot" value="6" style="display:none;">
183       <label for="spot6">Spot 6</label>
184     </div>
185     <div class="parking-spot">
186       <input type="radio" id="spot7" name="spot" value="7" style="display:none;">
187       <label for="spot7">Spot 7</label>
188     </div>
189     <div class="parking-spot">
190       <input type="radio" id="spot8" name="spot" value="8" style="display:none;">
191       <label for="spot8">Spot 8</label>
192     </div>
193     <div class="parking-spot">
194       <input type="radio" id="spot9" name="spot" value="9" style="display:none;">
195       <label for="spot9">Spot 9</label>
196     </div>
197     <div class="parking-spot">
198       <input type="radio" id="spot10" name="spot" value="10" style="display:none;">
199       <label for="spot10">Spot 10</label>
200     </div>
201     <div class="parking-spot">
202       <input type="radio" id="spot11" name="spot" value="11" style="display:none;">
203       <label for="spot11">Spot 11</label>
204     </div>
205   </div>
206 </form>
207
208
209 </body>
210
211 </html>

```

Source Code 3: La page principale, elle permet de renseigner les informations

Page Html de validation

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title>Projet UNIPI</title>
6   <style>
7     input[type="radio"]:checked+label {
8       background-color: blue;
9       color: white;
10      transform: scale(1.2);
11      transition: all 0.2s ease-in-out;
12    }
13
14   .parking-spots {
15     display: flex;
16     flex-wrap: wrap;
17   }
18
19   .parking-spot {
20     display: inline-block;
21     margin-right: 10px;
22   }
23
24   div {
25     display: flex;
26     position: relative;
27   }
28
29   label {
30     display: flex;
31     justify-content: center;
32     align-items: center;
33     width: 100px;
34     height: 100px;
35     background: red;
36     color: white;
37     border-radius: 10px;
38     margin: 0 10px 0 0;
39     cursor: pointer;
40   }
41
42   input:checked+label {
43     background: blue;
44   }
45 </style>
46 </head>
47
48 <body>
49   <div style="display: inline-block; margin-right: 48px;">
50     <h3>N du vehicule : {{ hvehicule_number }}</h3>
51   </div>
52   <div style="display: inline-block; margin-right: 48px;">
53     <h3> Kilometrage du vehicule : {{ hkilometrage }} </h3>
54   </div>
55   <div style="display: inline-block; margin-right: 48px;">
56     <h3> Place de parking : {{ hparkingspot }} </h3>
57   </div>
58   <br>
59
60   <script>
61     function valider() {
62       //envoyer le form
63       document.getElementById("bon").value = true;
64       document.getElementById("myForm").submit();
65     } // action="{{ url_for("test") }}" en dessous lol
66
67     function recommencer() {
68       //envoyer le form
69       document.getElementById("bon").value = false;
70       document.getElementById("myForm").submit();
71     } // action="{{ url_for("test") }}" en dessous lol
72   </script>
73
74   <form id="myForm" method="post">
75     <h3>Les donnees sont-elles correctes ?</h3>
76     <input type="hidden" name="bon" id="bon">
77   </form>
78
79   </div>
80   <button onclick="valider()"
```

```
81     style="background-color: green; color: white; font-size: 16px; padding: 10px 20px;">
82     ">Valider</button>
83 <button onclick="recommencer()" 
84     style="background-color: red; color: white; font-size: 16px; padding: 10px 20px;">
85     Recommencer</button>
86 </div>
87 </body>
88 </html>
```

Source Code 4: Page Html de validation, l'opérateur peut confirmer ou infirmer les informations fournies