|  |
| --- |
| Rapport technique  projet DEVeloppement (v2)  Mise en place d’un réseau de données dans un véhicule électrique  Année scolaire 2016-2017  Projet 34: GARDIN Alan, HACHICHA Anis, KHEDER Abdessalem  Tuteur : Mr LEGALL Guillaume  Destinataire : équipe pédagogique du projet S2 |
|  |

rÉSUMÉ

Ce document présente le travail effectué dans le cadre du projet Développement N°34 de l’année scolaire 2016-2017. L’objectif du projet était la « Mise en place d’un réseau de données dans un véhicule électrique ». C’est une plate-forme Open-Source Vehicle disponible sur le campus IMT Atlantique de Rennes et il s’agissait de fournir au terme du projet, trois applications Java (un client, deux serveurs) permettant d’échanger des données sur cette plate-forme. L’application client fonctionne sur une tablette PC utilisée comme tableau de bord, et les deux serveurs fonctionnent sur un système embarqué gérant un gestionnaire de batterie d’une part, et un contrôleur de moteur d’autre part. L’application client devait être capable de récupérer des ressources disponibles sur les deux serveurs.

Le projet s’est déroulé en deux phases. La première a permis de comprendre le sujet, et de concevoir selon les besoins de l’encadrant un cahier des charges. Celui-ci a ensuite donné lieu à un planning prévisionnel. La seconde phase est une phase de développement. Dans un premier temps, il a fallu réfléchir à l’architecture des applications demandées. Cette étape effectuée, l’effort du groupe s’est concentrée sur la réalisation des solutions techniques dégagées. Les applications demandées sont disponibles et bien intégrées avec le code déjà existant développé par l’équipe de recherche qui travaille sur la plate-forme. mais il reste à intégrer l’application cliente avec un code déjà existant. La dernière étape qui reste pour clôturer le projet c’est de tester ces applications sur le véhicule. Pour cela un déplacement des membres de groupe au campus de Rennes a été programmé le 12 juin 2017.

Alan Gardin

Table des matières

[Introduction 4](#_Toc482130725)

[Expression du besoin et Contraintes 6](#_Toc482130726)

[Technologies utilisées 7](#_Toc482130727)

[Architecture 8](#_Toc482130728)

[Programmation 13](#_Toc482130729)

[Intégration 14](#_Toc482130730)

[Conclusion 15](#_Toc482130731)

[References 16](#_Toc482130732)

[Glossaire 17](#_Toc482130733)

Introduction

De nos jours, l’Internet des objets est en train d’accélérer considérablement l’allure à laquelle apparaissent les innovations dans le secteur des transports, et en particulier en ce qui concerne les voitures et les camions que nous conduisons tous les jours. Aujourd’hui, de nombreuses équipes de recherche scientifique travaillent sur ces sujets.

C’est le cas de l’équipe OCIF (Objets Communicants et Internet du Futur)

qui travaille sur le campus de Rennes sur deux domaines applicatifs à savoir la gestion de l’énergie et le transport intelligent permettant de définir les besoins et les contraintes des réseaux Internet du futur et d’élaborer un terrain d’expérimentation pour valider les concepts développés.

Cette équipe de chercheurs travaille en particulier sur un véhicule électrique Open-source (OSV : Open Source Vehicule). Le moteur et les batteries du véhicule sont respectivement reliés à un contrôleur de moteur « EC : Engine Controller» et un système de contrôle de batteries « BMS Battery Managing System » qui fournissent plusieurs données à un système embarqué « APU » [3]. D’autre part, une application « tableau de bord » pour tablette a été développée. L’objectif de ce projet est alors d’établir la connexion entre l’APU et la tablette et à développer les applications qui permettent au conducteur d’accéder à toutes les données concernant son véhicule, ainsi que découvrir tous les appareils existants sur le réseau et répondant aux critères de définition d’un EC ou d’un BMS. C’est l’objectif de ce projet.

La progression de ce document suit la démarche adoptée par le groupe lors du projet. Dans un premier temps, nous avons commencé par une étape de découverte de tous les outils matériels et logiciels qui nous étaient imposés d’une part, ainsi que la compréhension de l’architecture générale du travail demandé d’autre part. Cette première réflexion s’est appuyée sur la rédaction du rapport technique, et d’un planning prévisionnel. Ensuite, s’en est suivie une phase de programmation et de tests aboutissant à trois applications (un client et deux serveurs) intégrable sur le véhicule.

Abdessalem Kheder

Expression du besoin et Contraintes

Le but du projet est de livrer des programmes Serveur et Client capables de communiquer entre eux et d’échanger des informations autour des données exploitables à partir des EC et BMS. Le langage choisi pour la programmation est Java, et le format retenu pour l’échange des ressources est le JSon [6]. Le programme client communiquera avec les programmes serveur par l’intermédiaire du protocole CoAP. Les programmes demandés utiliseront les bibliothèques Californium [4] et JSon-Simple.

Alan Gardin

Architecture matérielle du véhicule

L’APU PC est connecté par une liaison série au contrôleur de moteur (EC) qui est capable d’acquérir et commander la vitesse du moteur. Il est également connecté par USB au BMS (contrôleur de batterie) qui est capable de commander le chargeur des cellules et de connaitre leur état de charge.

De l’autre côté, l’APU PC est connecté par Ethernet à la tablette sur laquelle tourne une application « tableau de bord ».

Anis Hachicha

**Figure1 : Architecture matérielle du système développé. D’après (2)**

Technologies utilisées

CoAP

CoAP (Constrained Application Protocol) est un protocole proche du protocole HTTP, à l’exception qu’il a été pensé pour l’internet des objets. Il est dont particulièrement adapté aux besoins du projet. CoAP est conçu pour les applications machine à machine et fournit une découverte intégrée des ressources, un modèle d'interaction entre les bouts de l'application et \*à reformerles concepts de clé Web tels que les URLs. En outre, CoAP permet d'observer les ressources dans un modèle de publication / abonnement. Un serveur notifie les clients qui se sont abonnés à une ressource spécifique lorsque celle-ci change.

Avantages de l’API REST

Anis Hachicha

Bibliothèque Californium

Californium est une implémentation en Java du protocole CoAP. Il comprend les classes définissant les messages CoAP et les sous-classes de demande et de réponse dérivées. En outre, il définit des interfaces communes qui sont utilisées pour communiquer entre les packages. Les classes sur lesquelles se sont porté notre attention sont CoapClient et CoapServer.

Alan Gardin

Format JSon

JSon (JavaScript Object Notation – Notation Objet issue de JavaScript) est un format léger d'échange de données. Il se présente de la façon suivante :

**{"id":1, "language":"json", "author":"Groupe\_DEV"}.**

Il sera utilisé pour présenter de façon structurée les données. Il est facile à lire ou à écrire pour des humains. Il est aisément analysé ou généré par des machines.

Abdessalem Kheder

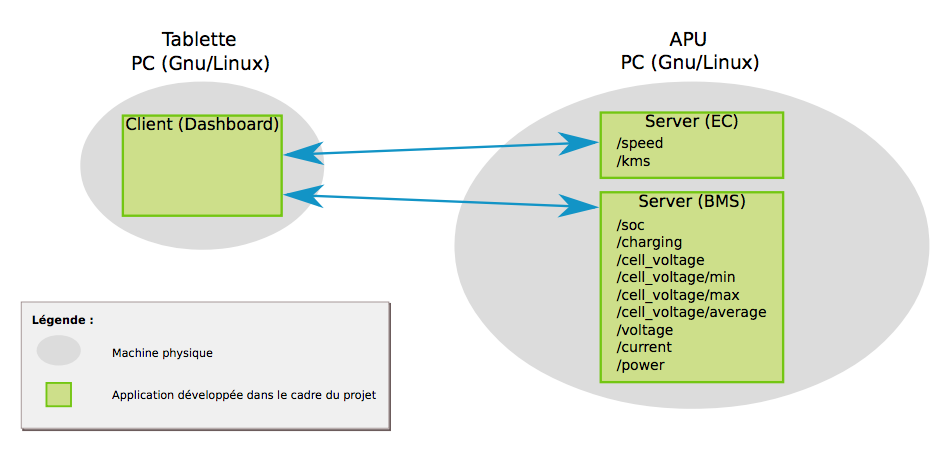
Architecture

Architecture globale du réseau :

Notre réseau est constitué de 3 nœuds :

* Un client qui envoie les requêtes, reçoit les réponses et les affiche sur le tableau de bord du véhicule.
* Un serveur qui fournit au client les données relatives à l’EC.
* Un serveur qui fournit au client les données relatives au BMS.

La figure 2 présente la liste de toutes les ressources du réseau. Comme nous avons mentionné dans les parties précédentes, notre code doit pourvoir s’intégrer avec d’autres applications qui existent déjà.

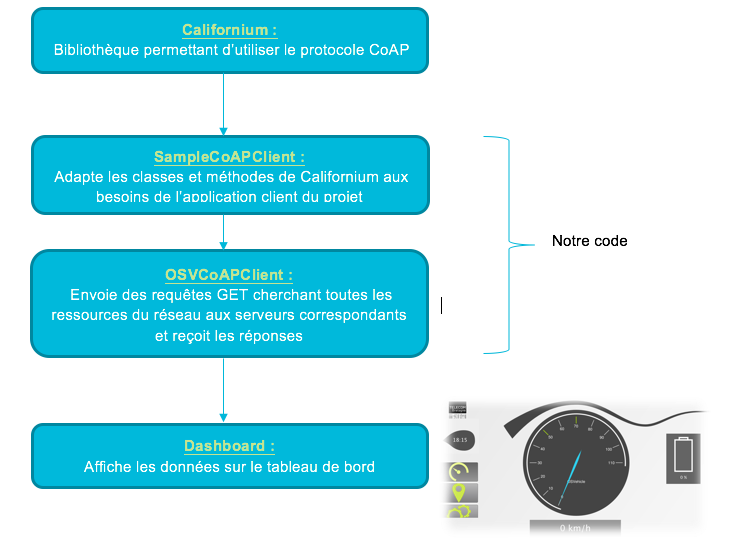


**Figure 2 : Architecture des applications et liste des ressources. D’après (2)**

Alan Gardin

Coté Client :

En effet, du côté du client, une application tableau de bord dashboard avait été développée en Java (voir figure 1) par l’équipe OCIF. Cette application fait appel à la classe Java OSVCoAPClient que nous avons développé. Cette dernière effectue des requêtes vers l’APU, reçoit les réponses sous format JSon et les convertit en données affichables sur le tableau de bord. Pour que cette classe puisse hériter des méthodes de la bibliothèque Californium d’une façon simple et adaptée à notre projet, nous avons créé une classe Java intermédiaire SimpleCoapClientqui s’intéresse seulement à l’aspect réseau. On a donc l’architecture de la figure 3.

**Figure 3 : Architecture client**

Abdessalem Kheder

Coté Serveur

Du côté des serveurs, on a la même architecture pour les deux serveurs. Prenons le cas du contrôleur de moteur EC : les valeurs mesurées par cet appareil sont stockées dans l’APU et peuvent être lues grâce à la classe ECDataReaderqui nous a été fourni par l’encadrant. De notre côté, nous avons développé l’application server-ec et les classes KMSResourceet SpeedResourcequi fonctionnent de la manière suivante :

A chaque fois que server-ecreçoit une requête GET de la part du client demandant par exemple la vitesse du véhicule, elle appelle SpeedResourcequi, en appelant ECDataReader**,** obtient la valeur de la vitesse et la convertit en format JSon (sérialisation) puis server-ecrenvoie cette réponse obtenue au client qui la demande.

Puis, nous avons développé la classe SimpleCoAPServerqui sert d’interface entre la bibliothèque californium, l’implémentation complète du protocol CoAP, et notre application server-ec. Le schéma de la figure 4 représente l’architecture actuelle de l’application.

Anis Hachicha

**Californium :**

Bibliothèque permettant d’utiliser le protocole CoAP

**SimpleCoAPServer :**

Adapte les classes et méthodes de Californium aux besoins de l’application serveur du projet

**Server-ec:**

Reçoit des requêtes GET cherchant les ressources disponibles et renvoie les réponses

Notre code

**SpeedResouce:**

Sérialise la vitesse du véhicule en Json

**KMSResouce**

Sérialise la distance parcourue en Json

**ECDataReader :**

Lit les valeurs mesurées par l’EC

**Figure 4 : Architecture Serveur**

Programmation

Requête GET du protocole CoAP

La fonction la plus basique de notre projet est qu’un client puisse effectuer une requête GET au serveur et que le serveur puisse fournir cette ressource en utilisant le protocole CoAP. Nous avons donc commencé par réaliser cette première tâche en créant les classes SampleCoAPClient et SampleCoAPServer. Pour les tester, nous avons développé deux applications de tests ClientTest et ServerTest et à travers les sorties de ces deux applications, on a pu vérifier que la requête GET se réalise avec succès.

Anis Hachicha

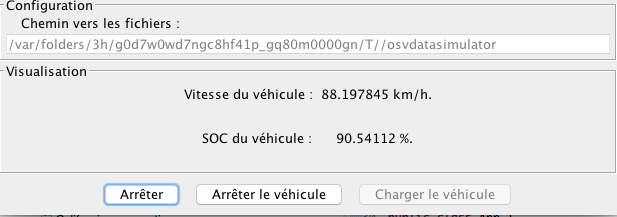
Réponses aux requêtes clients

A rédiger

Alan Gardin

Lecture des données par les serveurs :

Lors de la phase de programmation, notre encadrant, Mr LEGALL nous a fourni un simulateur du véhicule pour qu’on puisse avancer sur le projet sans travailler sur le véhicule réel qui se trouve à Rennes. Il s’agit d’une application qui permet de démarrer, arrêter et charger un véhicule virtuel tout en affichant les valeurs des différentes ressources. Il nous a aussi fourni des programmes qui permettent d’extraire les valeurs des ressources à partir de ce simulateur : ECDataReaderet BMSDataReader. Nous avons donc développé pour chaque ressource une classe permettant d’obtenir les valeurs à partir des Readers et de les convertir en format JSon. Puis nous avons programmé les deux applications serveurs server-ecet server-bms**.**

****

**Figure 5 : Capture écran du simulateur de l’OSV**

Abdessalem Kheder

Observation des ressources :

L’observation des ressources consiste à s’abonner à une ressource particulière, et recevoir une notification du serveur dès que cette ressource change de valeur. Toutes les ressources serveur ont la capacité d’être observée, mais l’application client s’intéresse seulement aux ressources speed et soc (vitesse du véhicule et état de charge).

Actuellement, l’application client est correctement prévenue dès que les ressources soc ou speed changent de valeur. Cela est mis en place par l’intermédiaire de Handler qui sont des classes appelées dès que la notification de changement de la ressource est reçue.

Alan Gardin

Intégration avec le tableau de bord :

Notre code de la partie clientqui permet d’accéder à toutes les ressources doit pouvoir s’interfacer avec l’application tableau de bord déjà existante.

En effet c’est une application avec interface graphique qui affiche les valeurs des ressources. Et en intégrant notre code ces valeurs doivent être mises à jour sur l’interface graphique (le tableau de bord) à chaque fois où elles changent dans le BMS ou l’EC. 

**Figure 6 : interface graphique de l’application tableau de bord**

Alan Gardin

Découverte des serveurs sur le réseau :

Cette étape consiste à découvrir quels sont les équipements sur le réseau qui répondent aux spécifications d’un BMS ou d’un EC. Cet aspect n’a pas été implémenté et sera effectuée si le temps le permet. Une première idée évoquée par l’encadrant serait d’envoyer un certain paquet en multicast, et les équipements compatibles répondrait à l’auteur du paquet leur existence.

Anis Hachica

Intégration

À la fin de la phase de développement, nous avons vérifié que nos programmes fonctionnent correctement en lançant à la fois le simulateur et l’application du tableau de bord et en comparant les valeurs des ressources affichées par ces deux applications. Il reste à intégrer l’application client avec le programme « tableau de bord » existant. Ensuite, il faudra tester notre programme sur le véhicule réel et donc un déplacement au campus de Rennes a été programmé. Ainsi nous pourrons installer les applications serveur sur l’APU du véhicule, placer la tablette en face du volant, et essayer tous les scénarios possibles (déplacement, chargement, accélération etc.…)

Alan Gardin

Conclusion

Bien qu’inachevé, le projet est proche des objectifs fixés. La seule tâche importante restante est celle de l’intégration du code de l’application client avec un code déjà existant. D’autres tâches permettraient de peaufiner le travail effectué, comme la découverte d’équipements répondant à certains critères sur le réseau. Cependant, ces fonctionnalités concernent les perspectives futures d’évolution du véhicule, comme par exemple l’intégration d’un nouveau serveur BMS de façon simple.

Alan Gardin

References

BIBLIOGRAPHIE :

1. PAULI D., OBERSTEG D. «Californium». Research Group For Distributed System, ETH Zurich, printemps 2010.

SITES INTERNET :

2. OSVEHICULE. OSVehicule [en ligne] Disponible sur : <https://www.osvehicle.com/> (consulté le 4/2/2017).

3. PC Engines BmbH. Apu Platform [en ligne] Disponible sur : <http://pcengines.ch/apu.htm> (consulté le 6/2/2017).

4. ECLIPSE. Californium [en ligne] Disponible sur : <https://github.com/eclipse/californium> (consulté le 8/2/2017)

5. INDIANA UNIVERSITY. CoAP Tutorial for Eclipse [en ligne] Disponible sur : <http://cs.iupui.edu/~xiaozhon/course\_tutorials/Coap\_tutorial\_Arduino\_Eclipse> (Consulté le 8/2/2017)

6. AYDAR A. and al. JSON with Java [en ligne] Disponible sur : https://www.tutorialspoint.com/json/json\_java\_example.htm (consulté le 28/2/2017)

Abdessalem Kheder

Glossaire

OCIF : Equipe de chercheur basé sur Rennes.

OSV: Open Source Vehicule.

BMS: Battery Monitoring Station. Il s’agit d’un système de contrôle de la batterie.

EC : Engine Controller. C’est le un système qui contrôle le moteur du véhicule.

APU : Système embarqué qui donne accès aux ressources du BMS et de l’EC.

CoAP : Protocole utilisé pour l’échange de données.

Californium : bibliothèque pour le protocole CoAP en Java.

JSon : format de présentation des données.

JSon-Simple : bibliothèque pour la gestion du format JSon en Java.

Anis Hachicha