LABORATOIRE
DES SCIENCES
DU NUMÉRIQUE
DE NANTES

RAPPORT DE TER

RAFFINEMENT DE PROTOCOLES DE COMMUNICATIONS PAR TRANSFORMATION DE MODÈLES

ÉTUDIANTS

GUERIN Antoine, ROZEN Anthony, GICQUEL Alexandre

ENCADRANT

Pascal ANDRÉ

Table des matières

1 Introduction

Dans le cadre de notre Master, nous devons effectuer un projet de recherche encadré par un chercheur ou enseignant chercheur, dit TER. Ce projet était à réaliser tout au long du second semestre de l'année 2020/2021, en parallèle des enseignements. Malgré ces derniers, nous avons eu assez de temps pour travailler dessus grâce à une journée entière dédiée à celui-ci par semaine et grâce à notre temps libre. La date butoir de ce projet à été fixé le 19 mai, ce qui correspond à 18 semaines de travaux, plus une semaine dédiées à la réalisation du rapport et des soutenances.

Après avoir contacté l'encadrant du sujet de recherche « Raffinement de protocoles de communications par transformation de modèles », Monsieur Pascal ANDRÉ, pour avoir un peu plus d'informations sur ce sujet, nous avons établi un rendez-vous pour faire pars de nos motivations et pour comprendre plus en détail le sujet. Suite a cette réunion, nous avons compris que le sujet est tiré du besoin de réduire le temps de réalisation d'un projet logiciel en réduisant le temps d'implémentation des développeurs. Pour réduire ce temps on peut agir sur la transformation d'un modèle UML en code Java, en déléguant ce travail à un algorithme informatique. Ensuite lors de la deuxième réunion avec l'encadrant, nous avons compris que notre sujet de recherche se concentrais sur la transformations de modèles pour un logiciel où intervient la communication à distance. C'est à dire pour un logiciel où il y a une communication à distance entre deux système informatique comme par exemple entre un robot et un portail, quand le robot s'approche du portail, le portail s'ouvre. Et nous avons aussi compris que le terme raffinement de protocoles voulait dire détailler les protocoles de communication entre deux système informatique pour pouvoir automatiser une génération de code à partir d'un modèle représentant cette communication.

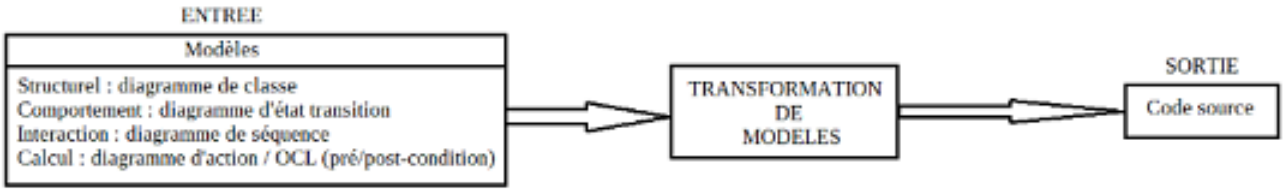

Image 1 : Le but du projet

Par la suite on a établi une fiche de route, les étapes a réalisé lors de ce sujet de recherche. La sélection d'une étude de cas, l'expérimentation

manuelle, l'introspection et l'automatisation de cette étude.

Dès le début, à l'annonce des sujet de TER disponible, nous avons été attiré par ce sujet de recherche dû fait de nos antécédent scolaire, Science de l'Ingénieure et Système d'Information et Numérique et de notre formation actuel. Car avant de commencer à étudier le projet, notre encadrant Pascal Andre nous as informer que le projet se réaliserai à l'aide d'un robot. Et on trouvais qu'il y avait un petit lien avec l'historique de nos formations, c'est ce petit lien qui nous à d'abord attiré vers ce sujet. Mais ensuite, après la réunion avec l'encadrant du projet, c'est le besoin d'aider les développeur à maximiser leur temps en réduisant le temps entre analyse et conception d'un logiciel. Ce qui nous as confirmé notre choix car en étant développeur on comprend l'avantage/le besoins d'automatisé la conception de certaines parties d'un logiciel.

Mais, il ne faut pas que oublier le sujet de recherche est assez récent, donc on n'attend pas une réponse aux besoins des développeur mais une avancé pour se rapprocher de plus en plus de cette réponse.

Pour mener a bien ce projet, nous avons à disposition les éléments suivants :

- Le Robot EV3 qui représentera un système informatique, grâce a ça brique EV3 Lego Mindstorm qui est le système programmable du robot. Sur cette brique, on peut y brancher différent éléments qui nous permettrons de simuler par exemple une communications entre un portail et un robot.
- Le JVM, Java Virtual Machine, leJos sur lequel tourne l'EV3.
- L'IDE Eclipse qui nous permettra de concevoir le code qui permettra une communication entre deux systèmes leJos
- L'IDE Android Studio qui nous permettra de concevoir le code qui permettra une communication entre deux systèmes Android
- Internet qui nous permettra de trouver des articles de recherche, les documentations sur les robots EV3, les informations sur les protocoles Wifi eet Bluetooth, etc
- Wireshark qui nous permettra de voir/saisir les communication entre les systèmes informatique.

2 Robot EV3

Le robot EV3 est est composé d'une brique EV3 programmable qui en plus d'être compatible avec les anciennes brique RCX et NXT, ce qui permet d'établir une connexion entres elles (si seulement la brique accepte le type de connexion), a un vaste domaine de connexion. En plus des connexions disponibles sur la brique NXT (connexion via Bluetooth ou USB), la brique EV3 dispose du WiFi et d'un port de sortie supplémentaire. En plus, la brique EV3 dipose aussi des composants suivants qui permettent d'effectuer différentes tâche suivant les besoins.

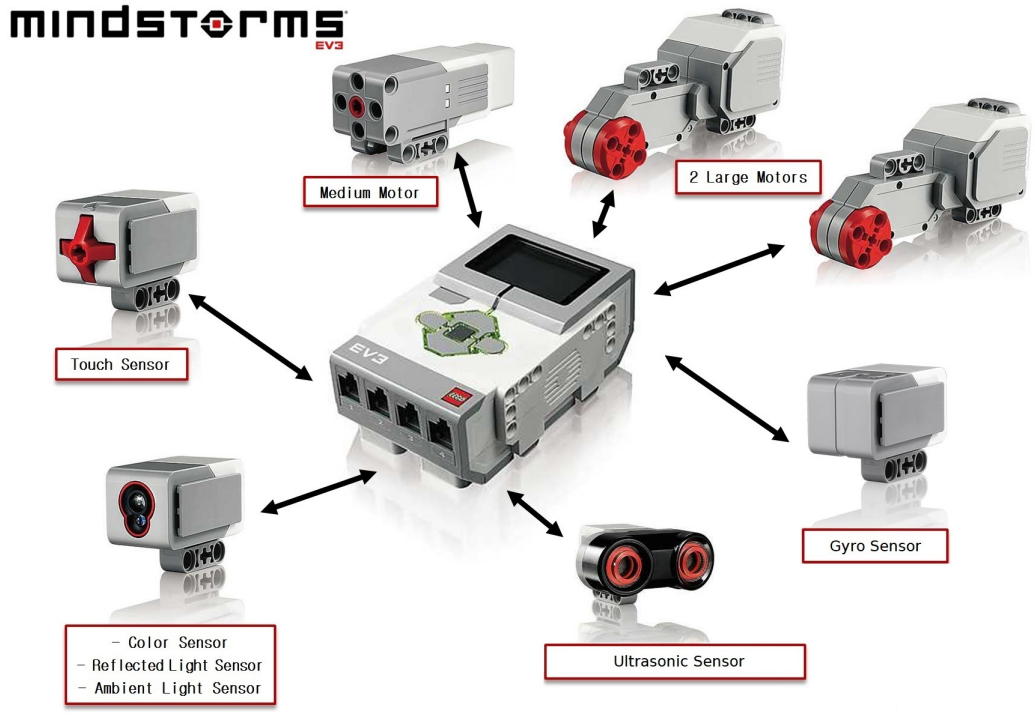

Image 2 : Les composants du robot Mindstorm EV3

Mais d'autres composants peuvent être ajouté comme le capteur infrarouge avec sa télécommande infrarouge. Pour la programmation de la brique EV3, nous utiliserons le framework open source leJos EV3 sortie en 2014 qui est souvent utilisé pour programmer ces types de robot. De plus il existe des plugins de ce framework pour les 2 principaux IDE Java, Eclipse et NetBeans.

3 Etude de cas

3.1 Introduction de l'étude de cas

Lors de ce projet de recherche, nous devons mettre en place une étude de cas pour visualiser comment marche une communication entre deux systèmes informatique. Cette visualisation va permettre d'identifier les protocoles de communication identiques par rapport a d'autres logiciel où intervient une communication entres systèmes. Tout cela pour arriver peut-être a une future automatisation de ces protocoles.

Dans ce TER, pour s'intéresser aux communications, nous nous focaliserons sur une des 3 études de cas suivantes possibles :

- Communications entre un portail et une voiture : Le portail s'ouvre quand il capte la bonne adresse IP émis par une voiture
- Communications entres deux voitures : Les deux voitures (automatiques) communique pour éviter une collision.
- Communications entre une tablette Android et une voiture : La contrôlabilité à distance d'une voiture depuis une tablette Android.

Nous avons choisie l'étude de cas comportant une communication entre un portail et une voiture car de nos jours, c'est la communication la plus proche de notre quotidien parce que les voitures automatiques ne sont pas omniprésente actuellement et ne sont pas encore fiable à 100% pour éviter les collision. Et concernant le contrôle d'une voiture depuis une tablette, ceci se produit pour les voitures télécommandées, soit pour les enfants, soit lors de cascades pour le cinéma, donc ce n'est pas une communication qui peut arriver quotidiennement.

De plus, nous avons choisi cette étude de cas car étant un sujet de recherche actif sur plusieurs années, les anciens groupes ont déjà réalisés une étude de cas similaire comportant qu'une seule communication. Cela nous permet de ne pas partir de 0, et donc de pouvoir aller plus loin que les groupes des années précédentents. Notre objectif cette année est de s'appuyer sur les anciennes étude de cas pour en créer une plus complexe, avec plusieurs communications de types différents.

Le système déterminé par l'étude de cas fonctionne de la manière suivante : Imaginons qu'un portail protège l'accès à notre domicile, et ce portail est initialement fermé. Lorsqu'une voiture s'approche du portail avec l'intention de le franchir, le portail interroge le véhicule pour savoir si celui-ci a le droit de le franchir. Si l'accès est autorisé, le portail s'ouvre, sinon le portail

reste fermé. Le problème dans tout cela, est de savoir comment interroger le véhicule, comment avancer le véhicule et avoir une issue de secours pour ouvrir le portail ?

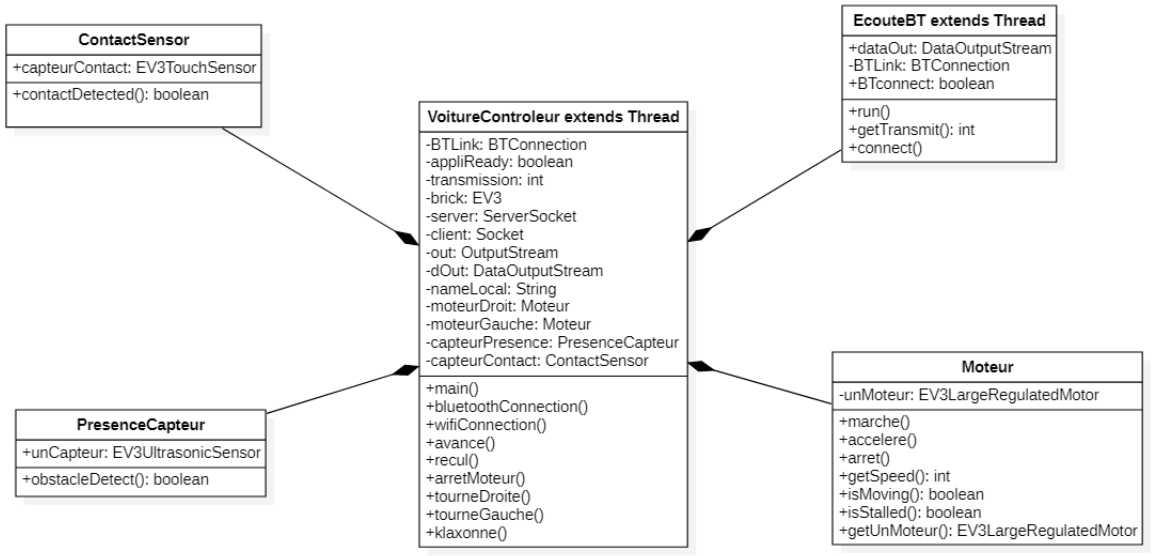

Figure 3 : Class diagram - Vehicle part

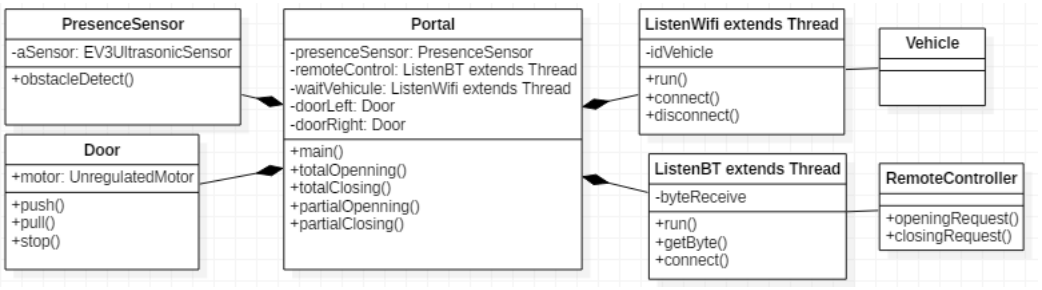

Figure 4 : Class diagram - Portal part

3.2 Expérimentation manuelle

Pour que les différents systèmes (véhicule, portail te télécommande) puissent s'envoyer des informations, il faut pouvoir établir une connexion

entre eux. Entre le véhicule et sa télécommande, et le portail et sa télécommande, on a choisie d'effectuer des connexions Bluetooth. Cependant un système accepte qu'une connexion Bluetooth à la fois, donc entre le véhicule et le portail on a choisi d'effectuer une connexion Wifi. De plus, cela permet de regarder différents types de connexion et de communication.

3.2.1 La connexion/communication entre le portail et la télécommande

Le portail est controlé à distance par une télécommande. Afin d'initialisé la connexion, le portail gère simplement la réception en Bluetooth des données. C'est à dire, tant que la télécommande ne demande pas de se connecter, le portail attends. Lorsque l'application Androïd se lance , l'essai de connexion est démarrer. Pour établir cette connexion, il faut que chacun des appareils aient différent paramètre. Du coté du portail, qui est serveur, puisqu'il attends que quelqu'un se connecte à lui, il lui faut comme paramètres le mode de connexion, ainsi que le temps maximal d'attente d'une connexion. Pour les differents mode de connexion qu'on peut mettre :

- NXTConnection.RAW (pour appareil de type tablette, telephone, ...)
- NXTConnection.PACKET (pour appareil de type brique NXT)
- NXTConnection.LCP (pour accéder à distance au menus de la brique)

Du coté de la télécommande, qui est elle client, les paramètres sont le nom de l'appareil serveur et le mode de connexion également.

Une fois connecté, le portail possède une boucle tournant tant que l'application fonctionne est ajoutée dans le « main », celle-ci s'occupe d'appeler la bonne méthode en fonction du message reçu par Bluetooth. En fonction du type de message reçu par la télécommande, le portail se charge en local d'appeler les bonnes fonctions sur les bons composants, afin de refléter l'action demandée par l'émetteur. Une grande partie des étapes citées précédemment peut être automatisée. Toutefois, certains paramètres ont tout de même besoin d'être passés dans le convertisseur par le développeur/expert métier.

Au niveau de la télécommande, le code est basé sur une architecture Android Studio. Comme coté portail, le code utilise beaucoup de fonctions de bibliothèques spécifiques à la technologie utilisée, Lejos pour le portail, Android Studio ici. La majorité de ces fonctionnalités sont appelés dans la classe MainActivity, ainsi que dans la classe ConnectionBluetoothActivity, utilisée dans MainActivity. L'idée générale de MainActivity est de créer une

liste d'action, ici appelées "activité", pouvant être utilisée via l'interface graphique. Lorsque l'utilisateur appuie sur la tablette, ou tout autre support physique faisant tourner l'application, le système extrait l'activité devant être lancée dans la liste des activités possibles (relatif au bouton sur lequel l'utilisateur a appuyé).

3.2.2 La connexion/communication entre le véhicule et la télécommande

Le fonctionnement de la connexion et des communications entre le véhicule et la télécommande est le même que pour un portail et une télécommande. La seule différence viens du nombre d'actions possibles (représentés par les différents boutons de la télécommande). Sur la télécommande du portail, on a le choix d'ouvrir totalement le portail, de l'ouvrir partiellement, ou bien de le fermer, tandis que sur la télécommande du véhicule, on a des actions de mouvements ou de connexions.

Le code de la télécommande du véhicule devait être celui du groupe qui a effectué ce sujet de recherche l'année dernière, car le code Android convenait parfaitement à notre étude de cas. Cependant, après avoir eu plusieurs problèmes sur l'envoie de byte par Bluetooth, nous avons choisis de calquer le code Android de la télécommande du portail et de l'ajuster pour en faire le nouveau code Android de la télécommande.

3.2.3 La connexion/communication entre le véhicule et la portail

Le véhicule et le portail communique à distance par Wifi. Le portail est le client et le véhicule est le serveur

Afin d'initialisé la connexion, le véhicule gère simplement la réception en Wifi des données. C'est à dire, tant que le portail ne détecte pas le véhicule, celui-ci attends parallèlement. Lorsque le portail détecte le véhicule, l'essai de connexion est démarrer. Pour établir cette connexion, il faut que chacun des appareils aient différents paramètres.

Du coté du véhicule, qui est serveur, il faut comme paramètre seulement un port de connexion.

Du coté du portail, qui est client, les paramètres sont le même port de connexion et l'adresse IP du serveur.

Une fois connecté, le véhicule demande de passer, et si le portail connais ce véhicule alors il s'ouvre. Quelque seconde après le passage du véhicule, le

portail se referme et se déconnecte du véhicule.

Pour réalisé cette connexion nous avons utiliser les Sockets (Socket et SocketServeur) du framework Java. Cependant, la connexion wifi doit être effectuer en parallele car autrement pendant que le vehicule ou le portail établiera une connexion wifi avec l'autre système, il ne pourra effectuer aucune tâche tant que la connexion n'est pas établie. Pour réaliser cette parallelisation, il faut implenter la connexion wifi dans un thread. De plus, on doit aussi mettre les communications dans des threads pour les mêmes raisons.

3.3 Introspection du code et des Connexions/Communications

Dans l'étude de cas réalisé au dessus, on peut observer qu'il y a deux types de Connexions/Communications, une WIFI et l'autre Bluetooth. Donc on doit observer ce qui se passe lors d'une Connexion/Communication WIFI et lors d'une Connexion/Communication Bluetooth, pour pouvoir ensuite réaliser une surcouche de communication qui généralisera les différents types de Connexions/Communications.

3.3.1 Protocole WIFI

Pour capturer les paquets envoyés par une connexion et une communication wifi entre 2 robots Mindstorm EV3, l'un correspondant au véhicule et l'autre au portail, on peut utiliser le logiciel Wireshark. Cependant, ce logiciel n'est que utilisable sur un ordinateur. Donc pour pouvoir récupérer les paquets envoyés, nous avons d'abord pensé à utiliser un câble USB pour connecter l'ordinateur a une des deux briques EV3. Cependant, après quelques recherches sur internet et quelque essai, nous avons pas réussi à récupérer les paquets transmis. Ensuite nous avons pensé a mettre l'ordinateur entre les deux brique EV3 lors des envoies de paquet. Ce qui veut dire que l'ordinateur sera l'intermédiaire des deux brique pour pouvoir récuperer les paquets. Ce qui sera concluant (voir l'image d'en dessous)

Image 3 : Simplification de la transmission Wifi

Pour faciliter cette configuration, on lancera le progrmamme de deux manière différentes, une fois avec l'ordinateur désigné comme portail et une autre fois l'ordinateur désigné comme véhicule. Ce qui revient à séparer la situation d'au-dessus en deux parties.

Tout d'abord, pour réaliser une connexion avec l'ordinateur comme serveur (dans le cas portail <- véhicule (ordinateur)), il faut obtenir l'adresse IP (Identifer Protocol) locale de l'ordinateur pour que le client sache à quel adresse il doit se connecter. Pour obtenir cette adresse IP, on peut taper dans un invité de commande, la commande "ipconfig". L'adresse IP locale correspondra à l'adresse qui se trouve au bout de la ligne "Adresse Ipv4".

Il ne faut pas oublier que dans ce projet de recherche, le serveur correspond au véhicule et le client au portail.

Voici les observations et les analyses des paquets transmis lors des deux situations :

1. Transmission véhicule -> portail (ordinateur) :

   Le protocole utilisé pour la connexion et la communication wifi est le même, ce sont des protocoles TCP (Transmision Control Protocol).

No.	Time	Source	Destination	Protocol	Length	Info
125	11.434720	192.168.1.16	192.168.1.22	TCP	66	52130 → 1234 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
128	11.483888	192.168.1.22	192.168.1.16	TCP	66	1234 → 52130 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=2
129	11.484105	192.168.1.16	192.168.1.22	TCP	54	52130 → 1234 [ACK] Seq=1 Ack=1 Win=131328 Len=0
130	11.632187	192.168.1.22	192.168.1.16	TCP	59	1234 → 52130 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=5
131	11.682699	192.168.1.16	192.168.1.22	TCP	54	52130 → 1234 [ACK] Seq=1 Ack=6 Win=131328 Len=0

Image 4 : Capture des paquets pour la situation "véhicule -> portail (ordinateur)"

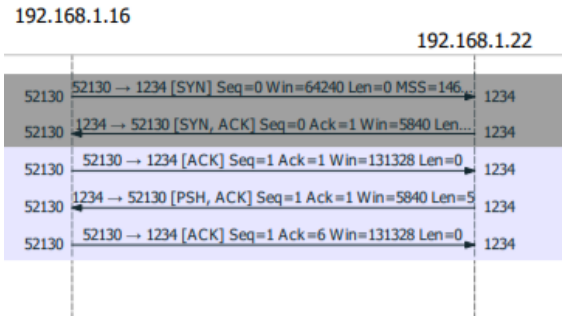

Image 5 : Graphe de flux TCP de la capture des paquets de l'image 1

Paquet n°1 (ligne ....) :

Ce premier paquet comporte le "flag" SYN, ce qui correspond à une demande de synchronisation ou à un établissement de connexion TCP du client vers le serveur.

Chaque session TCP commence par un numéro de séquence de zéro. De même, le numéro d'accusé de réception est également zéro, car il n'y a pas encore de côté complémentaire de la conversation à acquitter.

Paquet n°2 (ligne ....) :

Ce deuxième paquet comporte plusieurs "flags", un flag ACK qui correspond à un accusé de réception du paquet SYN du client. Et un flag SYN qui indique que le serveur souhaite aussi établir une connexion TCP.

Le serveur répond au client avec un numéro de séquence de zéro, car il s'agit de son premier paquet dans cette session TCP. Et un numéro d'accusé de réception de 1 pour indiquer la réception de l'indicateur SYN du client dans le paquet n°1 .

Paquet n°3 (ligne ....) :

Ce troisième paquet qui comporte le "flag" ACK correspond à l'accusé de réception du paquet SYN du serveur. Ce paquet termine l'établissement d'une connexion TCP.

Comme dans le paquet n ° 2, le client répond au numéro de séquence du serveur de zéro avec un numéro d'accusé de réception de 1. Le client inclut son propre numéro de séquence de 1 (incrémenté de zéro à cause du SYN).

Le numéro de séquence pour les deux hôtes est 1. Cet incrément initial de 1 sur les numéros de séquence des deux hôtes se produit pendant l'établissement de toutes les sessions TCP.

Paquet n°4 (ligne ....) :

Ce quatrième paquet comporte le "flag" PSH, ce qui correspond à l'envoie immédiate de données, aussi appelé charge utile. Il comporte aussi le "flag" ACK qui est l'accusé de réception du paquet du client. La charge utile de ce paquet a une longueur de 5 bytes.

Le numéro de séquence est laissé à 1, car aucune donnée n'a été transmise depuis le dernier paquet de ce flux. Le numéro d'accusé de réception est également laissé à 1, car aucune donnée n'a été reçue du serveur.

Paquet n°5 (ligne ....) :

Ce cinquième paquet comporte le "flag" ACK correspondant à l'accusé de réception du paque PSH du serveur.

L'accusé de réception à augmenté de 5, ce qui correspond à la longueur de la charge utile du paquet n°4. Donc maintenant l'ACK du client à pour valeur 6.

2. Transmission portail <- véhicule (ordinateur) :

   Le protocole utilisé pour la connexion et la communication wifi est le même, ce sont des protocoles TCP (Transmision Control Protocol).

No.	Time	Source	Destination	Protocol	Length	Info
20	5.742605	192.168.1.22	192.168.1.16	TCP	74	45437 → 1236 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=4294951957 TSecr=0 WS=2
21	5.742834	192.168.1.16	192.168.1.22	TCP	66	1236 → 45437 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
22	5.765369	192.168.1.22	192.168.1.16	TCP	54	45437 → 1236 [ACK] Seq=1 Ack=1 Win=5840 Len=0
23	5.823410	192.168.1.16	192.168.1.22	TCP	59	1236 → 45437 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=5
24	5.863014	192.168.1.22	192.168.1.16	TCP	54	45437 → 1236 [ACK] Seq=1 Ack=6 Win=5840 Len=0

Image 6 : Capture des paquets pour la situation "véhicule (ordinateur) -> portail"

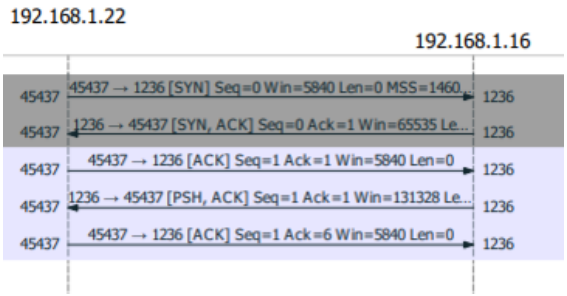

Image 7 : Graphe de flux TCP de la capture des paquets de l'image 3

Comme nous pouvons le remarquer, la procédure d'envoie de messages est la même que dans la situation "véhicule -> portail (ordinateur)". Cependant, concernant l'établissement d'une connexion, il y a quelques changement, les options suivantes se sont rajoutés :

— TSval,

— Tsecr

Nous pouvons constater que dans les deux situations d'au-dessus, il y a les options suivantes :

— MSS : Maximum Segment Size

— WS : Window Scale

— SACK_PERM : SACK Permitted

Pour conclure, nous avons constater qu'il y fallait 3 paquets pour établire une connexion et qu'il fallait seulement 2 paquets pour envoyé une communication. Nous avons aussi remarqué qu'il y avait différentes option pour une connexion TCP suivant quel système était le serveur ou le client.

3.3.2 Protocole Bluetooth

Tout comme pour capturer les différents paquets envoyés entre deux robots Mindstorm EV3, nous allons utiliser le logiciel Wireshark pour observer les différents paquets envoyés entre un EV3 et une télécommande (qui peut être en réalité un téléphone ou un émulateur sur ordinateur). Cependant, le logiciel Wireshark n'intègre pas automatiquement une solution afin de pouvoir analyser les protocoles Bluetooth sous Windows.[Analyse de protocol Bluetooth sous Windows via Wireshark] Il faut penser, au moment de l'intallation de Wireshark, de bien choisir d'installer le package optionel USBPcap, permettant de capturer le traffic USB, car sur une majorité des ordinateurs, le Bluetooth est utilisé en passant par le bus USB. Ensuite, pour capturer des paquets, il faudra lancer la capture sur l'interface USBPcap 1.

Le problème intervenant ensuite est le fait que, comme nous cherchons à intercepter les paquets sur l'ordinateur, le téléphone ne peut pas être la télécommande, il faut impérativement que ce soit l'émulateur sur Android Studio qui soit la télécommande. Cependant, l'émulateur d'Android Studio n'inclut pas de logiciel virtuel pour le Bluetooth, signifiant donc que nous ne pouvons pas utiliser le Bluetooth via l'émulateur.[Limitation de l'émulateur d'Android Studio] Cela implique que nous ne pouvons pas se servir de l'ordiateur comme télécommande, seul les téléphones le peuvent, mais nous ne pouvons pas nous servir du téléphone pour capter des paquets, seul l'ordinateur le peut. Une solution possible serait de capter les paquets Bluetooth via une application sur téléphone, mais le rendu est illisble et inexploitable.

3.3.3 Surcouche de communication

Tout d'abord une surcouche de communication est une couche qui se trouve au dessus de toutes communicatuions. Celle-ci consiste à intégrer les différents types de communications en une seule couche en généralisant les actions communes effectués lors d'une communication. Donc cette surcouche sera utilisé de la même façon peut importe le type de communication. Et

celle-ci ne devra pas se préocupper du type du message envoyé pour rester le plus générique possible. Cette surcouche pourrait être utilisé par un utilisateur souhaitant réaliser facilement des communications lors d'un projet car celui-ci n'aurait pu à implémenter ces communications suivant le type de celle-ci. Cependant dans notre cas, cette surcouche a objectif différent. Celle-ci permettra la réalisation du programme de transformation de modèles contenant des communications, car celui-ci utilisera cette surcouche pour générer le code des communications.

Afin de réaliser le code de cette surcouche, nous devons analyser les actions communes effectués par les différents types de communication (appellé primitives), analyser les paramètres des différents types de connexion et les services qui devront être intégrés. Cette analyse est basé dans un premier temps sur l'étude de cas efféctué, mais pourra par la suite être compléter avec de nouvelles types de communications.

1. Les services : Les services correspondent aux différents types de connexion possible. Dans un premier temps nous allons implémenter que les services utilisé dans l'étude de cas. Nous allons donc avoir les services suivant :
   - Une implémentaion Wifi serveur
   - Une implémentaion Wifi client
   - Une implémentaion Bluetooth serveur en utilisant le framework leJos
   - Une implémentaion Bluetooth client en utilisant le framework leJos
   - Une implémentaion Bluetooth client en utilisant le framework Android
2. Les paramètres : Les paramètres correspondent au données qu'il faudra donner à la surcouche lors de l'initialisation de celle-ci. Les paramètres diffèrent suivant le types de connexion à réaliser, voici les différents paramètres suivant le type de connexion :
   - Pour une connexion Wifi serveur :
     — Le port sur lequel va s'effectuer la connexion
   - Pour une connexion Wifi client :
     — Le port du serveur sur lequel le client doit se connecter
     — l'IP du serveur sur lequel le client doit se connecter
   - Pour une connexion Bluetooth serveur (leJos) :

— Le mode de connexion qui définie le type d'appareil pris en charge lors de la connexion
— Le temps maximum d'attente d'une connexion

- Pour une connexion Bluetooth client (leJos) :
  — Le nom de l'appareil sur lequel le client doit se connecter
  — Le mode de connexion qui définie le type d'appareil pris en charge lors de la connexion
- Pour une connexion Bluetooth client (Android) :
  — L'adresse MAC sur lequel le client soit se connecter

3. Les primitives : Les primitives correspondent aux actions communes effectuées par différents types de Connexions/Communications. Voici les fonctions communes qu'on va implémenter :
   - openConnection
   - closeConnection
   - sendMessage
   - sendMessageSynchronized
   - sendMessageAsynchronized
   - recieveMessage
   - sendACK
   - receiveACK

Nous pouvons donc en déduire que lors d'une communication entre deux systèmes (par exemple les systèmes de l'étude de cas), peut importe le type de la communication et du message envoyé, nous devons avoir quelque chose semblable a cela :

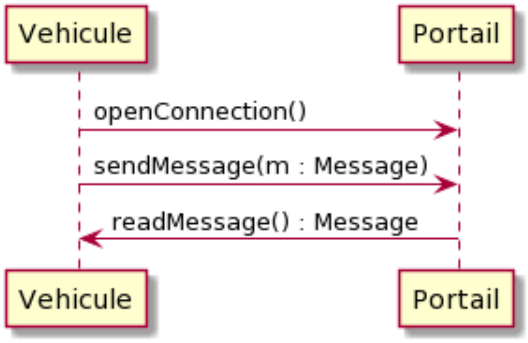

Image 8 : Communication de l'étude de cas

En observant de un plus près, nous devrons avoir quelque chose de ce type la :

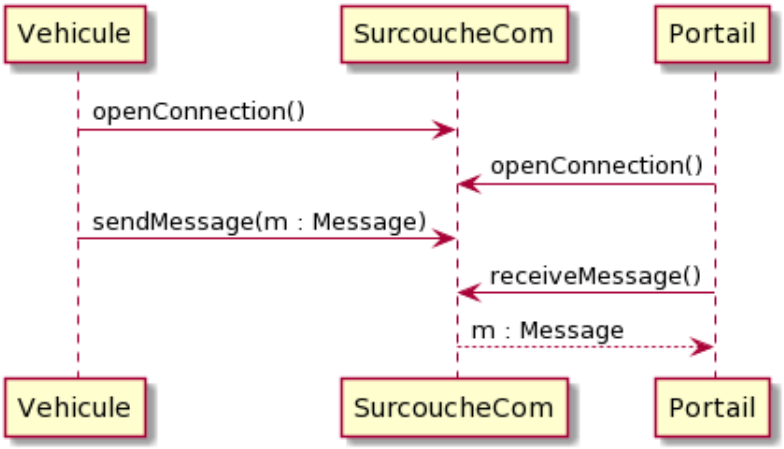

Image 9 : Communication de l'étude de cas en observant le lien avec la surcouche

Tout d'abord, avant de commencer à implémenter le code décrit par l'analyse, il nous a fallu créer un type Message générique, ce qui veut dire qu'un utilisateur n'aura plus à se préocupper du type du message qu'il doit recevoir. Et cela evitera aussi les erreurs qui surviennent quand un système veut recevoir un message d'un certain type, et à la place il recoit un message d'un autre type. Pour créer ce type nous avons réalisée une interface nommé "IMessage" qui prend en paramètre un type générique <T>, et réalisé une classe abstraite "AMessage" qui implémente cette interface. Les attributs de la classe abstraite décrivent le contenu d'un message et les fonctions implémenter permettent d'obtenir ou de modifier ces attributs, ou de renvoyer une chaîne de caratère corrspondante à la classe. Il y a aussi une fonction qui convertie une chaîne de caratère en un Message String car c'est le seul type de message qui peut être convertie dans les autres type de message (le message de type byte peut effectué la même chose). Un message est composé d'un attribut représentant les informations de connexion lié au message, autrement dit le sytème qui envoie le message et le système qui va recevoir ce message. Un attribut représentant les informations du message, qui comprend l'identifiant du message, le type du message et si le message doit recevoir un accusé de reception. Et elle est composé d'un attribut comportant le contenu du message. Puis nous avons réalisées les sous classes de la classe "AMessage", où chaque sous classe correspond à un type de message différent. De plus nous avons réalisé une classe factory qui permet de créer une instance

d'un message suivant le type indiqué dans un message string, et une classe encoduer/decodeur qui permet de encoder/decoder un message.

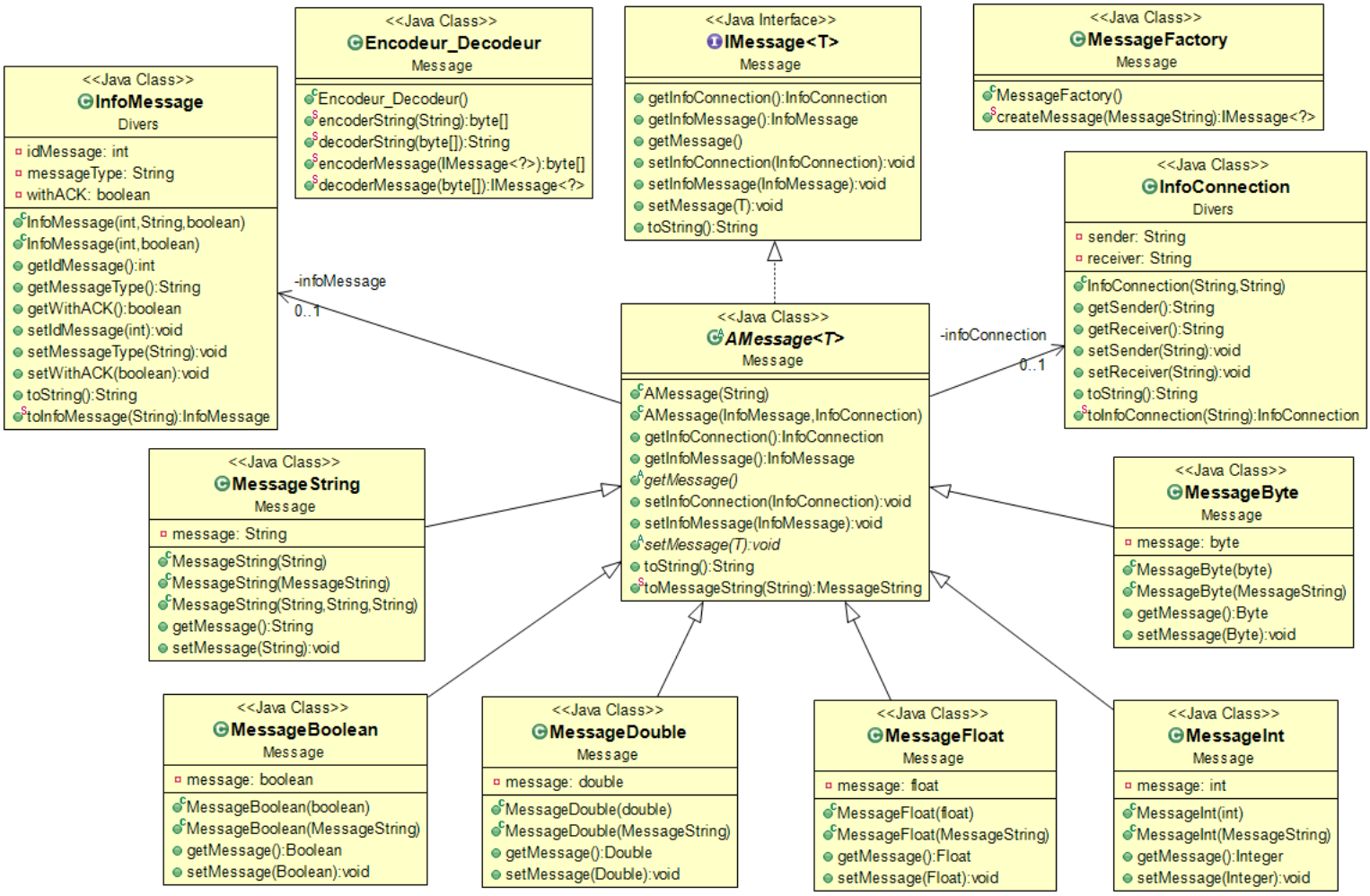

Image 10 : Diagramme de classe du type Message

L'analyse de la surcouche nous a indiqué les fonctions a mettre dans l'interface "IConnectionCommunication". Ces fonctions correspondent aux primitives. Lors de l'analyse on a distingué qu'il pouvait y avoir plusieurs types d'envoie de message : asynchrone et synchrone. Un envoie asynchrone se caractérise par le mécanisme des échanges en mode différé, ce qui correspond à un envoi de message où le système continue à effectué des tâches même si il n'a pas encore recu un accusé de réception pour le message en-

voyé. Un envoie synchrone se base sur le principe des échanges en temps réel, ce qui correspond à un envoi de message où le système attend l'accusé de récéption du message pour pouvoir continuer. Dans notre cas nous avons distinguer 2 communication asynchrone à réaliser, une qui envoie un message et qui n'attend pas d'accusé de récéption et une qui envoie un message mais qui attend un accusé de récéption. Après nous avons implémenter une classe abstraite "AConnectionCommunication" qui implémente l'interface. Une instance de ConnectionCommunication est composé d'un timeOut qui correspond au temps maximale avant de renvoyé un message si on n'a pas recu un accusé de récéption, d'un flux d'entrer et de sortie, d'un attribut représentant les informations de connexion lié à la connexion, et d'un identifiant qui correspond a un compteur du nombre de message, cela évitera que des messages aient le même identifiant. Ensuite nous devions réaliser plusieurs sous classes de "AconnectionCommunication" qui correspondent chacune à un service différent. Dans chaque sous classes il a fallu implémenter l'ouverture et la fermeture de la connexion car celle-ci diffère d'un service à l'autre.

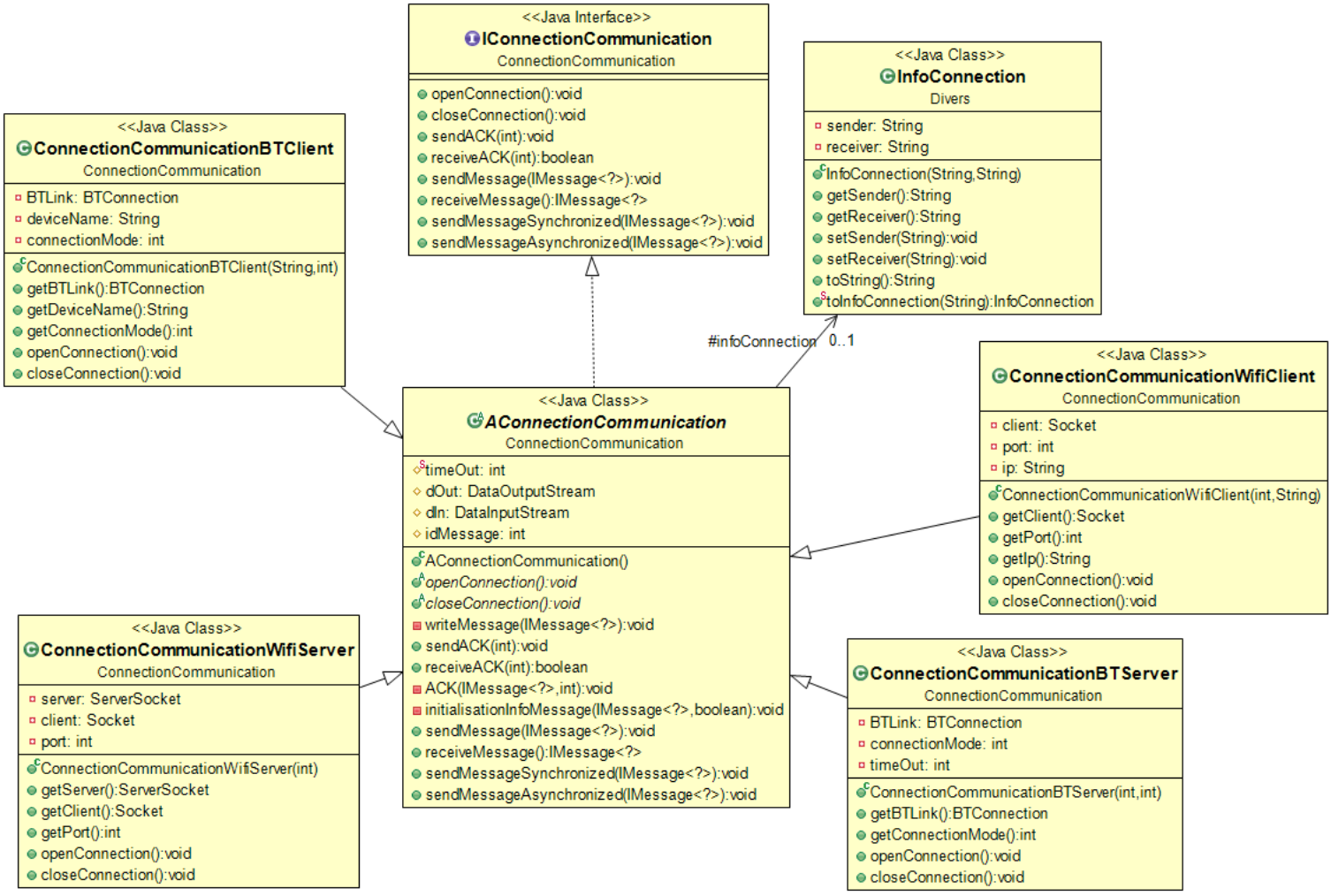

Image 11 : Diagramme de classe de la surcouche de communication

Pour conclure, maintenant que la surcouche de communication à été effectuée, pour confirmer que celle-ci fonctionne correctement, on l'a intégré à l'étude de cas initial.

4 Les différentes implémentations d'une connexion Bluetooth

Lors de l'implémentation du serveur et du client Bluetooth sous le framework leJos, nous avons constater qu'on pouvait exécuter ces programmes que sur les systèmes programmés sous leJos (une brique NXT dans notre cas), et non les exécuter sur d'autres systèmes programmés sous d'autres framework. Donc avons dû implémenter une connexion/communication sous le framework Android pour pouvoir utiliser la surcouche de communication pour les télécommandes des systèmes. Une question c'est donc poser, est ce qu'il y a d'autres implémentations de client/serveur Bluetooth différentes selon les framework ?

Après quelque recherche, nous avons trouvé qu'il y avait bien plusieurs implémentations différentes d'une connexion/communication Bluetooth selon le framework utilisé. Par exemple sur le site developer appel [doc framework Apple], on peut retrouver les documentations concernant une implémentations d'une connexion/communication Bluetooth selon le framework Core Bluetooth d'Appel.

5 Difficultés rencontrées

Lors de ce projet nous avons dû faire façe à plusieurs difficultés, comme par exemple la connexion wifi entre 2 EV3 car de base on c'était basé sur le framework leJos pour effectuer cette connexion. Or les classes qui permettent une telle connexion dans le framework leJos ne sont pas complète. Donc on c'est dit que c'était impossible d'effectuer une connexion wifi entre 2 EV3. Par conséquent nous avons réaliser une connexion Bluetooth entre les 2 EV3. Mais de ce fait nous devions changer les connexions Bluetooth entre les EV3 et leur télécommande, par une connexion wifi car un système ne peut pas avoir 2 connexion Bluetooth en simultané. Donc après quelque recherche pour effectuer ces deux nouvelles connexions, nous avons trouvé sur internet, sur le site variant press [connexion wifi entre 2 EV3], un exemple de code de

conexion wifi entre un EV3 et un ordinateur utilisant le framework de Java. Cependant, on c'est dis que peut etre cette implémentation de connexion wifi pourrait marché entre 2 EV3. Après quelque test, cette implémentation à été concluante pour établie cette connexion. Ensuite nous devions réaliser un envoie de données entre les deux EV3, cependant cela n'a pas marché du premier coup car quand on envoie des données a l'aide d'une des fonctions *write* de la classe DataOutputStream, il faut rajouter l'appel à la fonction *flush* qui permet de vider le flux de sortie et de force l'écriture de tous les octets de sortie mis en mémoire tampon, ce que nous n'avions pas fait.

Sur le code de la télécommande du véhicule que nous avons récupéré, nous avions remarqué que si nous appuyons sur une touche pour mouvoir le véhicule, alors le mouvement s'effectuait sur le véhicule bien plus longtemps que le temps que nous restions appuyé sur la touche en question. En plus, on a remarqué que nous ne pouvions faire qu'une seule transmissions entre la télécommande et le véhicule, probablement à cause du mouvement rallongé sur le véhicule.On a aussi constaté qu'une erreur s'affichait sur la brique au bout d'un certain temps d'exécution, et on pensait donc que les 2 erreurs étaient liées (l'erreur d'affichage pouvait venir d'une surcharge du flux de données liés aux envois de données par la télécommande). On a donc décider de calquer le code de la télécommande du véhicule sur celui de la télécommande du portail, étant donné que cette dernière n'avait aucun soucis. Cependant, le mouvement du véhicule sera donc moins controlable. En effet, on a choisi d'effectuer l'action demandé par la télécommande sur 1 secondes, pour ne pas avoir de problèmes de surcharge de flux. Malgré ces changements, l'erreur d'affichage survenait toujours. Cependant, on a remarqué que une fois que l'erreur apparaissait, la communication entre la télécommande et le véhicule n'était plus bloqué qu'à 1 seul transmission, on pouvait mouvoir le véhicule de la manière dont nous l'avons conçu. D'un côté on pouvait bouger le véhicule librement, mais d'un autre côté on ne pouvait plus rien observer sur la brique EV3.

Nous avons aussi du faire face aux difficultés lié à la Covid-19 car à cause de cette épidémie nous n'avons pas pu se voir toutes les semaines pour cause de confinement et de cas contact. Ce qui à engendré quelque petit retard car pour ces tests nous devions avoir accès a plusieurs EV3 en même temps. Et même en prétant nos EV3, nous ne pouvions pas réalisé certains de ces test car il y vait que un seul ordinateur dans notre groupe qui supportait

l'exécution de l'IDE Android Studio.

6 Perspective

Comme expliqué au dessus, du a la crise sanitaire nous avons eu quelque retard. Et du a ces retard, nous n'avons pas pu tester plusieurs chose comme la nouvelle télécommande qui permet au portail d'être le serveur wifi, comme le faite que est ce que les informations lié à une connexion Bluetooth sont correctement rempli, mais aussi le test de l'étude de cas modifié avec la surcouche de communication. Donc il reste a tester ces partie du projet. La surcouche de comunication implémenté permet à un serveur wifi de n'accepter qu'un et un seul client. Or un serveur wifi peut accepter plusieurs clients en même temps. Cependant dans le dosssier prototype du dossier code_java, du TER de cette année se trouve plusieurs piste d'implémentation de cette nouvelle surcouche comprenant un serveur qui accepte plusieurs client. Donc le prochain groupe qui travaillera sur ce sujet de recherche devra chosir la meilleure implémentation de cette nouvelles surcouche. Maintenant que l'étude de cas et la première abstraction ont été effectué les prochains groupe devront étudier comment implenter la surcouche de communication dans différent projet où interveinnent des communications (comment implenter la surcouche dans un modèles UML). Ensuite, il devront en déduire un premier prototype de transformation de modèle à partir de la généralisation des modèles UML effectués.

7 Conclusion

Ce projet de recherche était le premier contact que nous avions avec le domaine de la recherche scientifique. De ce fait, nous étions un peu perdu au début et nous ne savions pas vraiment par quoi débuter ou la démarche à suivre. Mais notre encadrant Mr. Pascal ANDRÉ nous a aidés a bien comprendre les attendus du sujet, et nous a bien indiqués les pistes à suivre quand nous étions bloqués. Ce projet nous a permis de comprendre comment se déroule le travail au sein d'un laboratoire de recherche et cela nous à permis de mettre en pratique nos connaissance d'ingénieurie. En effet, en plus de faire de la recherche pour trouver comment effectuer certaine tâche, nous devions aussi réfléchir à la meilleure implémentation possible dans le but d'effectuer

la meilleure abstraction possible des communications. Cependant, nous devions aussi réflechir à la meilleure implémentation de l'étude de cas, pour que celle-ci se rapproche le plus possible de la réalité.

8 Référence

[Analyse de protocole Bluetooth sous Windows via Wireshark]
https://tewarid.github.io/2020/08/20/analyze-bluetooth-protocols-on-windows-using
html
[Limitation de l'émulateur d'Android Studio]
https://developer.android.com/studio/run/emulator#wifi
[doc framework Apple]
https://developer.apple.com/documentation/corebluetooth
[connexion wifi entre 2 EV3]
http://variantpress.com/books/maximum-lego-mindstorms-ev3/