TER REPORT

REFINEMENT OF COMMUNICATION PROTOCOLS BY MODEL TRANSFORMATION

STUDENTS

GUERIN Antoine, ROZEN Anthony, GICQUEL Alexandre

SUPERVISING TEACHER

ANDRÉ Pascal

Contents

Chapter 1

Introduction

As part of our Master's degree, we have to carry out a research project supervised by a researcher or teacher researcher, called TER. This project was to be carried out throughout the second semester of the year 2020/2021, in parallel with the teachings. In spite of the latter, we had enough time to work on it thanks to a whole day dedicated to it per week and thanks to our free time. The deadline for this project was set on May 19Th, which corresponds to 18 weeks of work, plus one week dedicated to the realization of the report and the defense.

After having contacted the supervisor of the research subject "Refinement of communication protocols by model transformation", Mr. Pascal ANDRÉ, to have a little more information on this subject, we established a meeting to share our motivations and to understand the subject in more detail. Following this meeting, we understood that the subject is drawn from the need to reduce the time of realization of a software project by reducing the time of implementation of the developers. To reduce this time we can act on the transformation of a UML model into Java code, by delegating this work to a computer algorithm. Then, during the second meeting with the supervisor, we understood that our research topic was focused on the transformation of models for a software where remote communication is involved. That is to say for a software where there is a remote communication between two computer systems as for example between a robot and a portal, when the robot approaches the portal, the portal opens. And we also understood that the term protocol refinement meant to detail the communication protocols between two computer systems in order to automate a code generation from a model representing this communication.

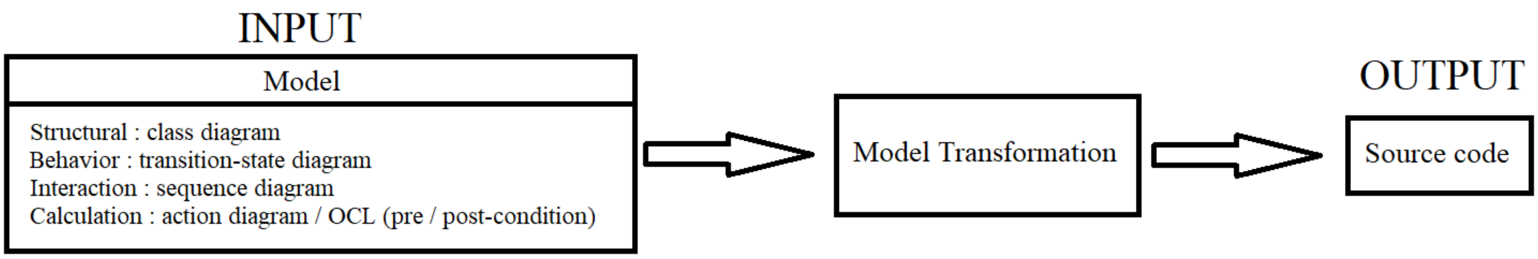

INPUT
Model
Structural : class diagram
Behavior : transition-state diagram
Interaction : sequence diagram
Calculation : action diagram / OCL (pre / post-condition)
Model Transformation
OUTPUT
Source code

Figure 1.1: The purpose of the project

Subsequently, a road-map was established, the steps to be carried out during this research topic. The selection of a case study, the manual experimentation, the introspection and the automation of this study.

From the beginning, when the available TER subjects were announced, we were attracted by this research subject because of our school background, Engineering Science and Information and Digital System and our current training. Because before starting to study the project, our supervisor Pascal Andre informed us that the project would be realized with a robot. And we found that there was a small link with the history of our training's, it is this small link that first attracted us to this subject. But then, after the meeting with the project supervisor, it was the need to help the developers to maximize their time by reducing the time between analysis and design of a software. This confirmed our choice because as a developer we understand the advantage/need to automate the design of certain parts of a software.

But, we must not forget that the subject of research is quite recent, so we do not expect an answer to the needs of developers but an advance to get closer and closer to this answer.

To carry out this project, we have at our disposal the following elements:

- The EV3 Robot which will represent a computer system, thanks to the EV3 Lego Mindstorm brick which is the programmable system of the robot. On this brick, we can connect different elements that will allow us to simulate for example a communication between a portal and a robot.
- The JVM, Java Virtual Machine, LeJos on which the EV3 runs.
- The Eclipse IDE that will allow us to design the code that will allow a communication between two systems leJos.

- The Android Studio IDE that will allow us to design the code that will allow communication between two Android systems.
- Internet which will allow us to find research articles, documentations on EV3 robots, information on WiFi and Bluetooth protocols, etc.
- Wireshark which will allow us to see/record the communication between computer systems.

1.1 Background

The objective is to create a software production chain from models for connected systems. In particular, it is about devices communicating with each other with a "real" execution environment that takes into account the constraints of operation, security, operational reliability and performance, for example.[1, 2]

Some features are general, others depend on the environment or the system itself. We can use one or several modeling languages (UML [3], SysML [4]), associated verification tools, simulation tools, etc.

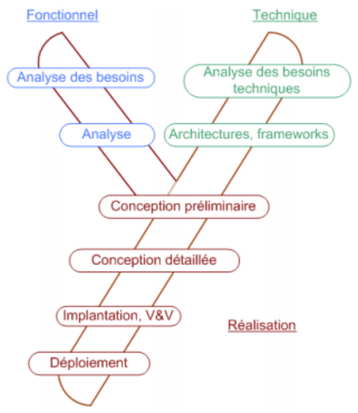


Figure 1.2: Unified process 2TUP

The analysis model immersed in a technical environment will be called a design model, as illustrated by the Y-shaped development model in Fig. 1.2 inspired by [5]. For this purpose, communication tools based on the PLCs of automaton, robots, sensors and actuators are used. Intermediate levels can be implemented to facilitate the implementation of the code production chain, inspired by the model-driven approach and software production lines
.

One of the major constraints is the method of sending UML messages. If we do the UML message sending and the model transformation on a single machine, the problem is simple, one method is enough. Here, the sending of UML messages is done by network, that makes the problem much more complex.

In the MDA vision, a transformation process is a sequence of transformations allowing to go from a Platform Independent Model to a more concrete Platform Specific Model. The logical models used as input to the process abstract from the technical environment and non-functional requirements. The design consists in "weaving" the logical model on the technical infrastructure to obtain an executable model. The code generation itself is not conceivable as the only transformation step, because of the semantic distance between the logical model and the technical target, composed of orthogonal but correlated aspects, called domains on which the initial model must be "woven".

A process can only be industrialized if all the cogs are known with precision. Practice has shown us that transformations are efficient when the models are semantically close (e.g. class diagram and relational model for persistence). Transformations are written in algorithmic form.

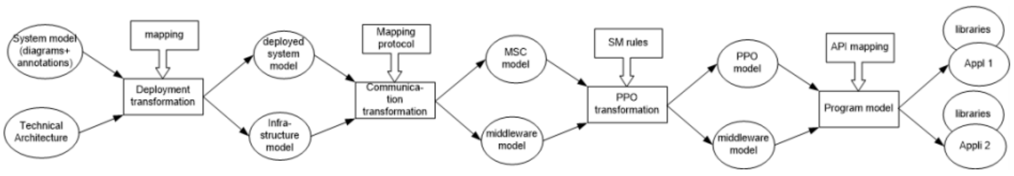


Figure 1.3: General transformation process

Working with simple transformations reduces consistency and completeness problems. Based on these considerations, we adopt a principle that we call small step transformations. The complexity is not in the transformation but in the transformation process.

A complex transformation is hierarchically composed of other transformations, down to elementary transformations. These macro transformations use configuration information. Obviously, if the starting model includes a component diagram and a deployment diagram, the deployment transformation will be simplified. It should be noted that all the parameters and decisions of the transformation must be kept in order to replay the transformation process if the initial model changes.

The process in Fig. 1.3 is abstract but generic. [1, 2]

1.2 Case Study

During this research project, we must set up a case study to visualize how a communication between two computer systems works. This visualization will allow us to identify the identical communication protocols compared to other software where a communication between systems intervenes. All this to arrive perhaps has a future automation of these protocols.

1.2.1 Functional Requirements

In this TER, to focus on communications, we will focus on one of the following 3 possible model:

- Communications between a portal and a vehicle : The portal opens when it receives the right IP address emitted by a vehicle
- Communications between two vehicles : The two vehicles (automatic) communicate to avoid a collision.
- Communications between an Android tablet and a vehicle : The remote controllability of a vehicle from an Android tablet.

We choose to study the communications between a portal and a vehicle. So we build our study case in such a way that not only could we study the different communications between the portal and the vehicle, but also we could remotely operate the vehicle or the gate using remote control.

1.2.2 Technical Environment

EV3 Robot

The EV3 robot is composed of a programmable EV3 brick that in addition to being compatible with the old RCX and NXT bricks, which allows to establish a connection between them (if only the brick accepts the type of connection), has a wide range of connections. In addition to the connections available on the NXT brick (connection via Bluetooth or USB), the EV3 brick has WiFi and an additional output port. In addition, the EV3 brick also has the following components that allow to perform different tasks according to the needs.

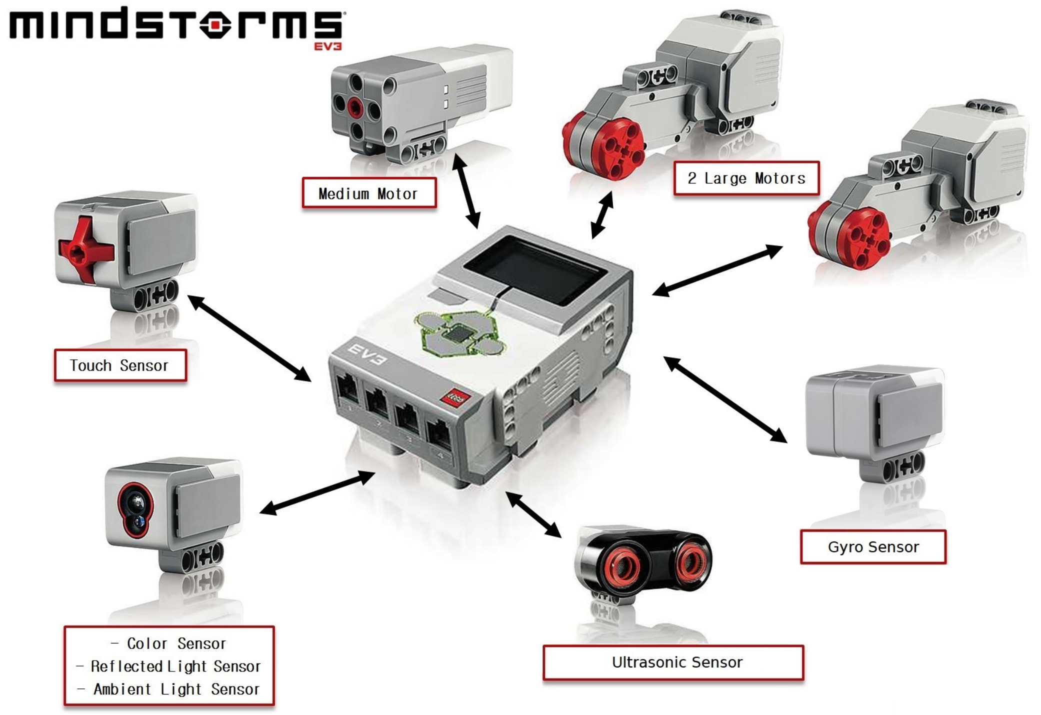


Figure 1.4: The components of the Mindstorm EV3 robot

But other components can be added like the infrared sensor with its infrared remote control. For the programming of the EV3 brick, we will use the open source framework leJos EV3 released in 2014 which is often used to program these types of robots. In addition there are plugins of this framework for the 2 main Java IDEs, Eclipse and NetBeans. Eclipse is the IDE chosen to program our EV3 robots with leJos.

Android

The 2 remotes, actually phone applications, are developed on Android Studio. It's the official IDE (Integrated Development Environment) for Android OS (Operating System). An application developed on Android Studio is divided into 2 parts : the User Interface, coded in XML, and the activities, which manage all possible interactions with the user, coded in Java. On Android Studio, there are 2 way to test an application under development. The first one is by connect a phone to a computer via a USB cable. Android Studio will automatically detect your device and, when the application will be compiled, it will automatically run on the phone. The second way to test an application is by using the emulator in Android Studio. Its clearly less powerful than your phone, but the emulator provides almost all the capabilities of a real Android device. The real advantage of the emulator is that an application can be tested on a variety of devices without needing the physics one.

1.2.3 Introduction of the case study

In addition, we chose this case study because being an active research topic over several years, previous groups have already done a similar case study with only one communication. This allows us not to start from 0, and therefore to go further than previous years' groups. Our goal this year is to build on the previous case studies to create a more complex case study with several different types of communications.

The system determined by the case study works as follows: Let's say a portal protects access to our home, and that portal is initially closed. When a car approaches the portal with the intention of passing through it, the portal polls the vehicle to see if it has the right to pass through. If it is allowed, the portal opens, otherwise the portal remains closed. The problem in all this is how to interrogate the vehicle, how to move the vehicle forward and have an escape route to open the portal?

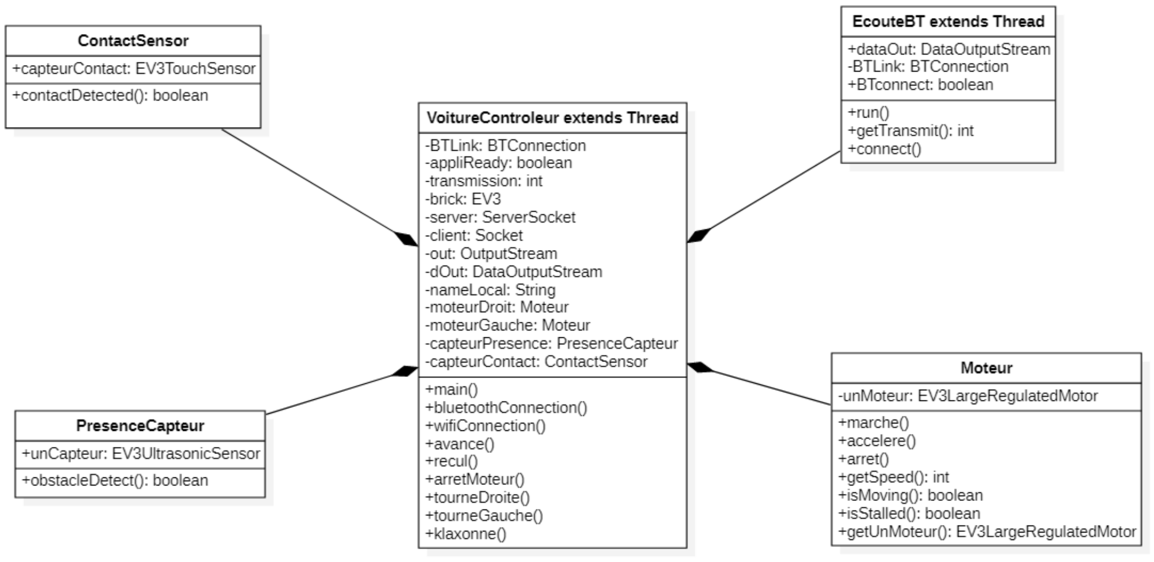


Figure 1.5: Class diagram - Vehicle part

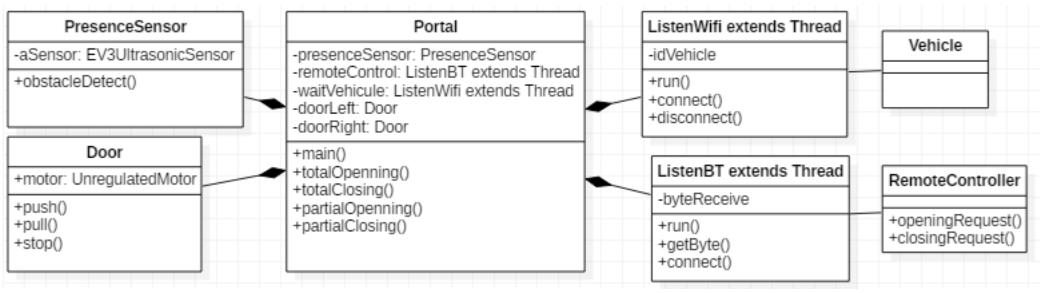


Figure 1.6: Class diagram - Portal part

Chapter 2

Manual experimentation

In order for the different systems (vehicle, portal and remote control) to send information to each other, a connection must be established between them. Between the vehicle and its remote control, and the portal and its remote control, we have chosen to make Bluetooth connections. However a system accepts only one Bluetooth connection at a time, so between the vehicle and the portal we chose to make a WiFi connection. Moreover, this allows to watch different types of connection and communication.

2.1 The connection/communication between the portal and the remote control

The portal is controlled remotely by a remote control. In order to initialize the connection, the portal simply manages the Bluetooth reception of data. That is, as long as the remote control does not ask to connect, the portal waits. When the Android application is launched, the connection test is started. To establish this connection, it is necessary that each of the devices have different parameters. On the side of the portal, which is a server, since it waits for someone to connect to it, it needs as parameters the connection mode, as well as the maximum time to wait for a connection. For the different connection modes we can set :

- NXTConnection.RAW (for device of type tablet, telephone, etc)
- NXTConnection.PACKET (for device of type NXT brick)
- NXTConnection.LCP (to access remotely to the menus of the brick)

On the remote control side, which is a client, the parameters are the name of the server device and the connection mode as well.

Once connected, the portal has a loop that runs as long as the application is added in the "main", which takes care of calling the right method according to the message received by Bluetooth. Depending on the type of message received by the remote control, the portal takes care locally to call the right functions on the right components, to reflect the action requested by the transmitter. Many of the above steps can be automated. However, some parameters still need to be passed into the converter by the developer/business expert.

At the remote control level, the code is based on an Android Studio architecture. As on the portal side, the code uses a lot of functions from libraries specific to the technology used, Lejos for the portal, Android Studio here. Most of these features are called in the MainActivity class, as well as in the ConnectionBluetoothActivity class, used in MainActivity. The general idea of MainActivity is to create a list of actions, here called "activity", that can be used via the GUI. When the user presses on the tablet, or any other physical support running the application, the system extracts the activity to be launched in the list of possible activities (relative to the button on which the user has pressed).

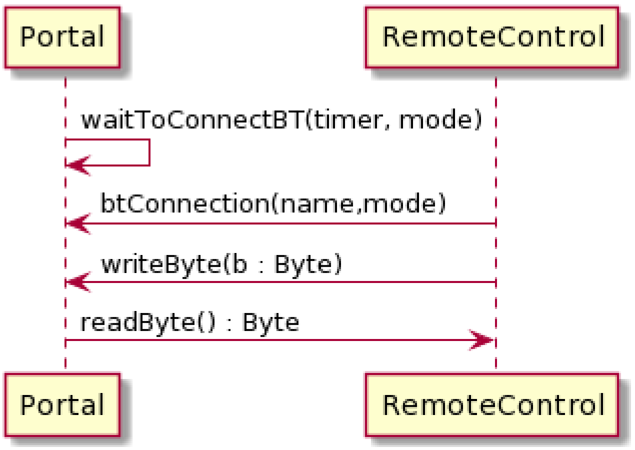


Figure 2.1: The connection/communication between the portal and the remote control

2.2 The connection/communication between the vehicle and the remote control

The connection and communication between the vehicle and the remote control is the same as for a portal and a remote control. The only difference is the number of possible actions (represented by the different buttons on the remote control). On the portal remote control, we have the choice to open the portal completely, to open it partially, or to close it, while on the vehicle remote control, we have movement or connection actions.

The code of the vehicle remote control was supposed to be the one of the group that did this research topic last year, as the Android code was perfectly suitable for our case study. However, after having several problems on the Bluetooth byte sending, we chose to copy the Android code of the portal remote control and adjust it to make the new Android code of the remote control.

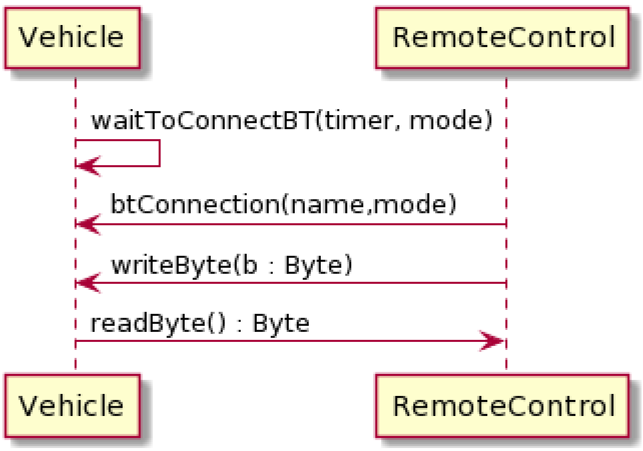


Figure 2.2: The connection/communication between the vehicle and the remote control

2.3 The connection/communication between the vehicle and the portal

The vehicle and the portal communicate remotely via WiFi. The portal is the client and the vehicle is the server

In order to initialize the connection, the vehicle simply manages the reception of data in WiFi. That is to say, as long as the portal does not detect the vehicle, it waits in parallel. When the portal detects the vehicle, the connection test is started. To establish this connection, each of the devices must have different parameters.

On the side of the vehicle, which is a server, only one connection port is required as a parameter.

On the side of the portal, which is a client, the parameters are the same connection port and the IP address of the server.

Once connected, the vehicle asks to pass, and if the portal knows this vehicle then it opens. A few seconds after the vehicle passes, the portal closes and disconnects from the vehicle.

To realize this connection we have used the Sockets (Socket and Socket-Server) of the Java framework. However, the WiFi connection must be done in parallel because otherwise while the vehicle or the portal will establish a WiFi connection with the other system, it will not be able to perform any task until the connection is established. To achieve this parallelization, we need to implement the WiFi connection in a thread. Moreover, we must also put the communications in threads for the same reasons.

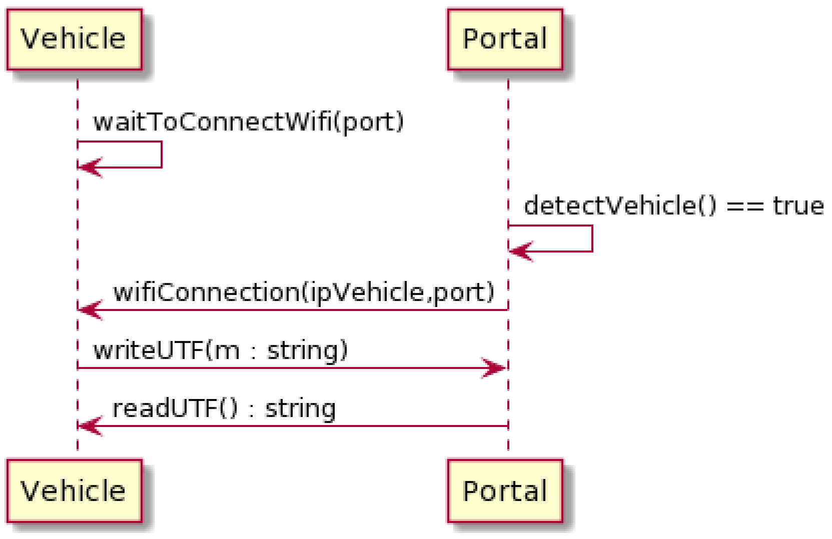


Figure 2.3: The connection/communication between the vehicle and the portal

Chapter 3

Introspection of the code and the Connections/Communications

In the case study above, we can observe that there are two types of Connections/Communications, one WIFI and the other Bluetooth. So we must observe what happens during a WIFI Connection/Communication and during a Bluetooth Connection/Communication, in order to be able to realize a communication overlay that will generalize the different types of Connections/Communications.

3.1 WIFI protocol

To capture the packets sent by a WiFi connection and communication between 2 Mindstorm EV3 robots, one corresponding to the vehicle and the other to the portal, we can use the Wireshark software. However, this software is only usable on a computer. So to be able to retrieve the sent packets, we first thought of using a USB cable to connect the computer to one of the two EV3 bricks. However, after some research on the internet and some tests, we didn't succeed in recovering the transmitted packets. Then we thought of putting the computer between the two EV3 bricks when sending packets. This means that the computer will be the intermediary of the two bricks to recover the packets. What will be conclusive (see the image below)

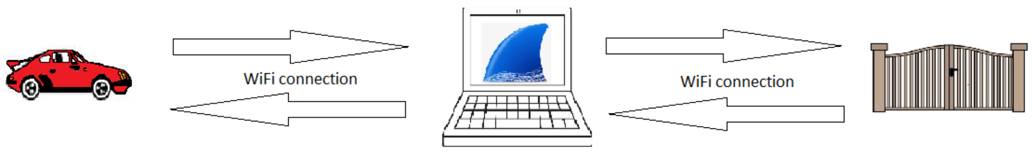


Figure 3.1: Simplification of WiFi transmission

To facilitate this configuration, we will launch the program in two different ways, once with the computer designated as the portal and another time with the computer designated as the vehicle. This means that the situation above is divided into two parts.

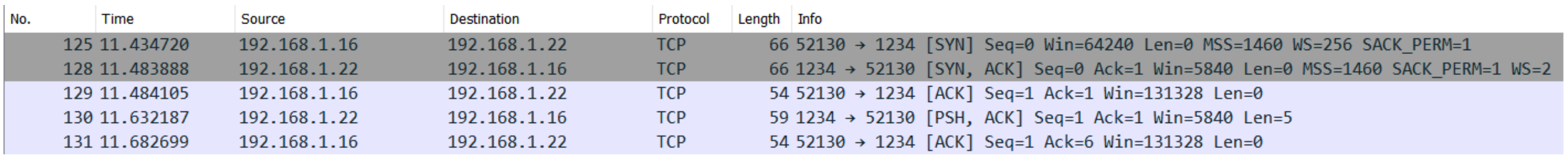
First, to make a connection with the computer as a server (in the case of portal <- vehicle (computer)), we need to obtain the local IP (Identifer Protocol) address of the computer so that the client knows which address it should connect to. To obtain this IP address, we can type in a command prompt, the command "ipconfig". The local IP address will correspond to the address at the end of the line "IPv4 address".

Remember that in this research project, the server is the vehicle and the client is the portal.

Here are the observations and analyses of the packets transmitted in the two situations:

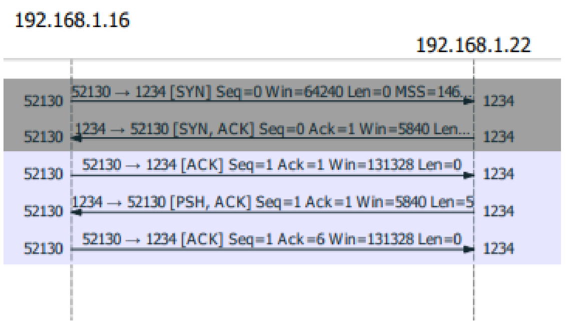
1. Transmission vehicle -> portal (computer):

   The protocol used for the connection and the WiFi communication is the same, they are 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

Figure 3.2: Capture of packets for the "vehicle -> portal (computer) situation"



```
192.168.1.16
                                                                192.168.1.22
52130  52130 → 1234 [SYN] Seq=0 Win=64240 Len=0 MSS=146...         1234
52130  1234 → 52130 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len...         1234
52130  52130 → 1234 [ACK] Seq=1 Ack=1 Win=131328 Len=0             1234
52130  1234 → 52130 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=5          1234
52130  52130 → 1234 [ACK] Seq=1 Ack=6 Win=131328 Len=0             1234
```

Figure 3.3: Packet capture TCP flow graph in Fig. 3.2

Package n°1 (line 125): This first packet contains the SYN flag, which corresponds to a synchronization request or a TCP connection establishment from the client to the server.

Each TCP session starts with a sequence number of zero. Similarly, the acknowledgement number is also zero, because there is no additional side of the conversation to acknowledge yet.

Package n°2 (line 128): This second packet has several "flags", an ACK flag which corresponds to an acknowledgement of receipt of the SYN packet from the client. And a SYN flag which indicates that the server also wants to establish a TCP connection.

The server responds to the client with a sequence number of zero, as this is its first packet in this TCP session. And an acknowledgement number of 1 to indicate receipt of the client's SYN flag in packet 1.

Package n°3 (line 129): This third packet, which includes the ACK flag, corresponds to the acknowledgement of the SYN packet from the server. This packet completes the establishment of a TCP connection.

As in packet 2, the client responds to the server's sequence number of zero with an acknowledgement number of 1. The client includes its own sequence number of 1 (incremented from zero because of the SYN).

The sequence number for both hosts is 1. This initial increment of 1 on both hosts' sequence numbers occurs during the establishment of all TCP sessions.

Package n°4 (line 130): This fourth packet contains the PSH flag, which corresponds to the immediate sending of data, also called payload. It also contains the ACK flag, which is the acknowledgement of receipt of the client's packet. The payload of this packet has a length of 5 bytes.

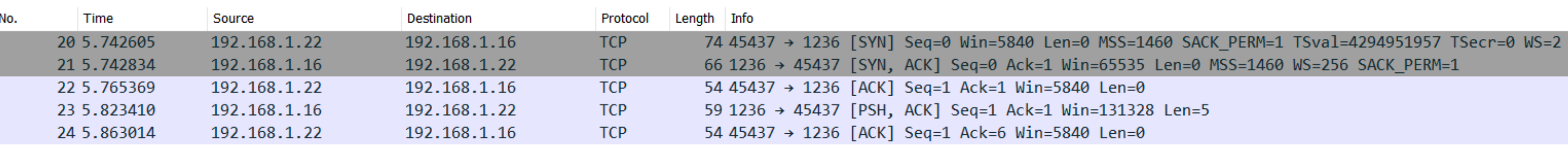
The sequence number is left at 1, because no data has been transmitted since the last packet of this flow. The acknowledgement number is also left at 1, because no data has been received from the server.

Package n°5 (line 131): This fifth packet has the ACK flag corresponding to the acknowledgement of the server's PSH packet.

The acknowledgement has increased by 5, which corresponds to the length of the payload of packet 4. So now the ACK of the client is 6.

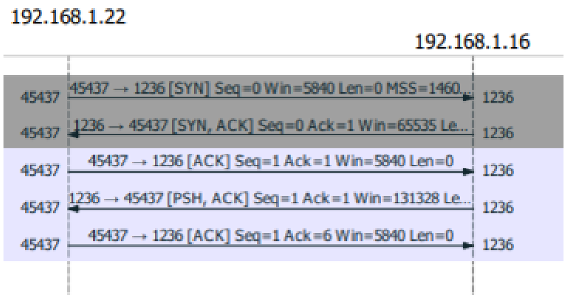
2. Transmission portal <- vehicle (computer):

   The protocol used for the connection and communication wifi is the same, they are 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

Figure 3.4: Capture of packets for the "vehicle (computer) -> portal situation"



```
192.168.1.22
                                                          192.168.1.16
45437  45437 → 1236 [SYN] Seq=0 Win=5840 Len=0 MSS=1460...       1236
45437  1236 → 45437 [SYN, ACK] Seq=0 Ack=1 Win=65535 Le...       1236
45437  45437 → 1236 [ACK] Seq=1 Ack=1 Win=5840 Len=0             1236
45437  1236 → 45437 [PSH, ACK] Seq=1 Ack=1 Win=131328 Le...      1236
45437  45437 → 1236 [ACK] Seq=1 Ack=6 Win=5840 Len=0             1236
```

Figure 3.5: Packet capture TCP flow graph in Fig. 3.4

As we can see, the procedure of sending messages is the same as in the situation "vehicle -> portal (computer)". However, regarding the establishment of a connection, there are some changes, the following options have been added:

- TSval,
- Tsecr

We can see that in the two situations above, there are the following options :

- MSS : Maximum Segment Size
- WS : Window Scale
- SACK_PERM : SACK Permitted

To conclude, we have noticed that 3 packets were needed to establish a connection (the first three packets) and that only 2 packets were needed to send a communication (the last two packets). We also noticed that there were different options for a TCP connection depending on which system was the server or the client.(Fig. **??**, Fig. **??**)

3.2 Bluetooth protocol

Just like capturing the different packets sent between two Mindstorm EV3 robots, we are going to use the Wireshark software to observe the different packets sent between an EV3 and a remote control (which can actually be a phone or a computer emulator). However, the Wireshark software does not automatically integrate a solution to analyze Bluetooth protocols under Windows.[6] You have to think, when installing Wireshark, to choose to install the optional USBPcap package, allowing to capture USB traffic, because on most computers, Bluetooth is used through the USB bus. Then, to capture packets, you have to launch the capture on the USBPcap 1 interface.

The next problem is that, as we are trying to intercept packets on the computer, the phone cannot be the remote control, so the emulator on Android Studio must be the remote control. However, the Android Studio emulator does not include virtual software for Bluetooth, which means that we cannot use Bluetooth via the emulator.[7] This implies that we cannot use the computer as a remote control, only the phone can, but we cannot use the phone to pick up packets, only the computer can. A possible solution would be to capture Bluetooth packets via an application on the phone, but the rendering is unreadable and unusable.

Chapter 4

Communication Primitives

A communication overlay is a layer that is on top of all communications. It consists in integrating the different types of communications in a single layer by generalizing the common actions performed during a communication. So this overlay will be used in the same way no matter the type of communication. And this one will not have to worry about the type of the message sent to remain the most generic possible. This overlay could be used by a user wishing to realize easily communications during a project because this one would not have to implement these communications according to the type of this one. However in our case, this overlay has a different objective. This one will allow the realization of the program of transformation of models containing communications, because this one will use this overlay to generate the code of communications.

4.1 Primitives analysis

In order to realize the code of this overlay, we have to analyse the common actions performed by the different types of communication (called primitives), analyse the parameters of the different types of connection and the services that will have to be integrated. This analysis is initially based on the case study, but can be completed later with new types of communication. Thus, the case study allowed us to observe that between the Bluetooth and WiFi services, there was no difference for sending a message, so we will have a single implementation, for all the services, for the primitive allowing a message to be sent. On the other hand, concerning the establishment and closure of the connection, each service has a different implementation. Moreover, depending on whether the system is a client or a server of the connection, this changes the internal implementation of the service. So there will be several different implementations for each role of each service. Therefore, the communication overlay will be done in a way to avoid code redundancy, so create an abstract class implementing an interface, and which will be declined in several sub-classes each representing a protocol and a role (the role is either server or client). We also noticed that to establish a connection between two systems, there were different parameters depending on the service and its role. To remain as generic as possible, sending a message should not depend on the type of message sent. For this reason, a Message type was created to allow this generalisation of messages. After a more detailed study of the possible primitives, we have created several types of message sending. One synchronous message sending, which means that the system waits for an acknowledgement before continuing. And two asynchronous message sending, which means that the system can continue to perform tasks even if it has not received an acknowledgement. The primitives corresponding to these message sending and acknowledgement have therefore been added. So we have the following list of services, parameters and primitives:

1. Les services : The services correspond to the different types of possible connections.
    - A WiFi server implementation
    - A WiFi client implementation
    - A Bluetooth server implementation using the leJos framework
    - A Bluetooth client implementation using the leJos framework
    - A Bluetooth client implementation using the Android framework
2. The parameters : The parameters correspond to the data that will be given to the overlay when it is initialized. The parameters differ according to the type of connection to be made, here are the different parameters according to the type of connection:
    - For a WiFi server connection:
        - The port on which the connection will be made
    - For a client WiFi connection:
        - The port of the server on which the client must connect
        - the IP of the server on which the client must connect
    - For a Bluetooth server connection (leJos):

  - The connection mode that defines the type of device supported when connecting
  - The maximum time to wait for a connection
- For a Bluetooth client connection (leJos):
  - The name of the device on which the client should connect
  - The connection mode that defines the type of device supported when connecting
- For a Bluetooth client connection (Android):
  - The MAC address on which the client is to connect

3. The primitives: Primitives correspond to common actions performed by different types of Connections/Communications. Here are the common functions that we will implement:
   - openConnection
   - closeConnection
   - sendMessage
   - sendMessageSynchronized
   - sendMessageAsynchronized
   - receiveMessage
   - sendACK
   - receiveACK

If we look more closely, after the overlay is made, two systems should communicate in the following way:

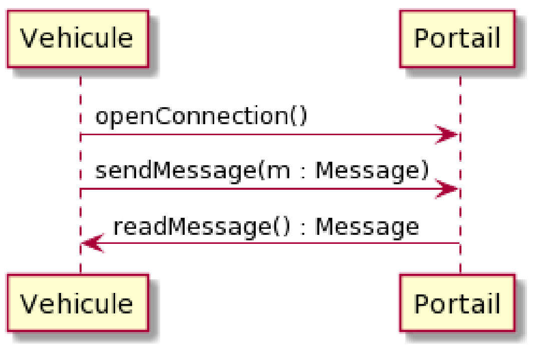


Figure 4.1: Communication of the case study

And within this communication, we can see what role the communication overlay plays through the following sequence diagram:

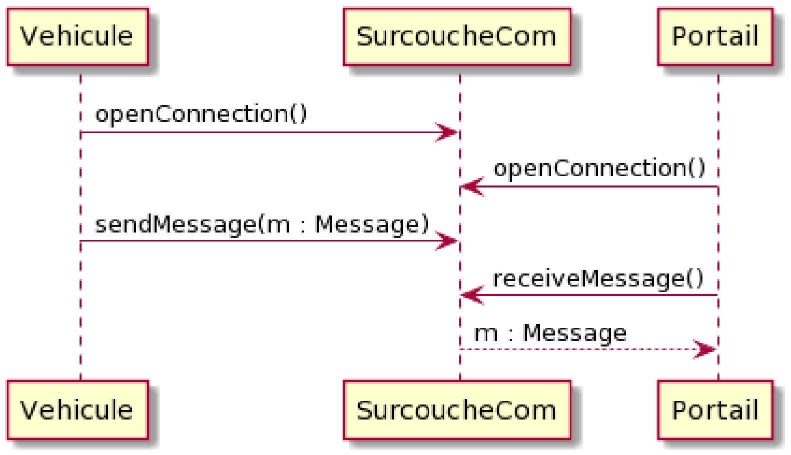


Figure 4.2: Communication of the case study by observing the link with the overlay

4.2 Generic message

Before we started implementing the code described in the analysis, we had to create a generic message type, which means that a user will not have to worry about what type of message they should receive. This will also avoid errors that occur when a system wants to receive a message of a certain type, and instead receives a message of another type. To create this type we made an interface named "IMessage" which takes as parameter a generic type <T>, and made an abstract class "AMessage" which implements this interface. The attributes of the abstract class describe the content of a message and the implemented functions allow to get or modify these attributes, or to return to the class a corresponding string. There is also a function that converts a string into a Message String because it is the only type of message that can be converted into other types of messages (the message byte can do the same thing). A message is composed of an attribute representing the connection information related to the message, i.e. the system sending the message and the system receiving the message. An attribute representing the message information, which includes the message identifier, the message type and whether the message should be acknowledged. And it is composed of an attribute containing the content of the message. Then we made the sub-classes of the "AMessage" class, where each sub-class corresponds to a different type of message. Furthermore we have made a factory class which allows to create an instance of a message according to the type indicated in a message string, and an encode/decode class which allows to encode a message in byte or to decode a message in String.

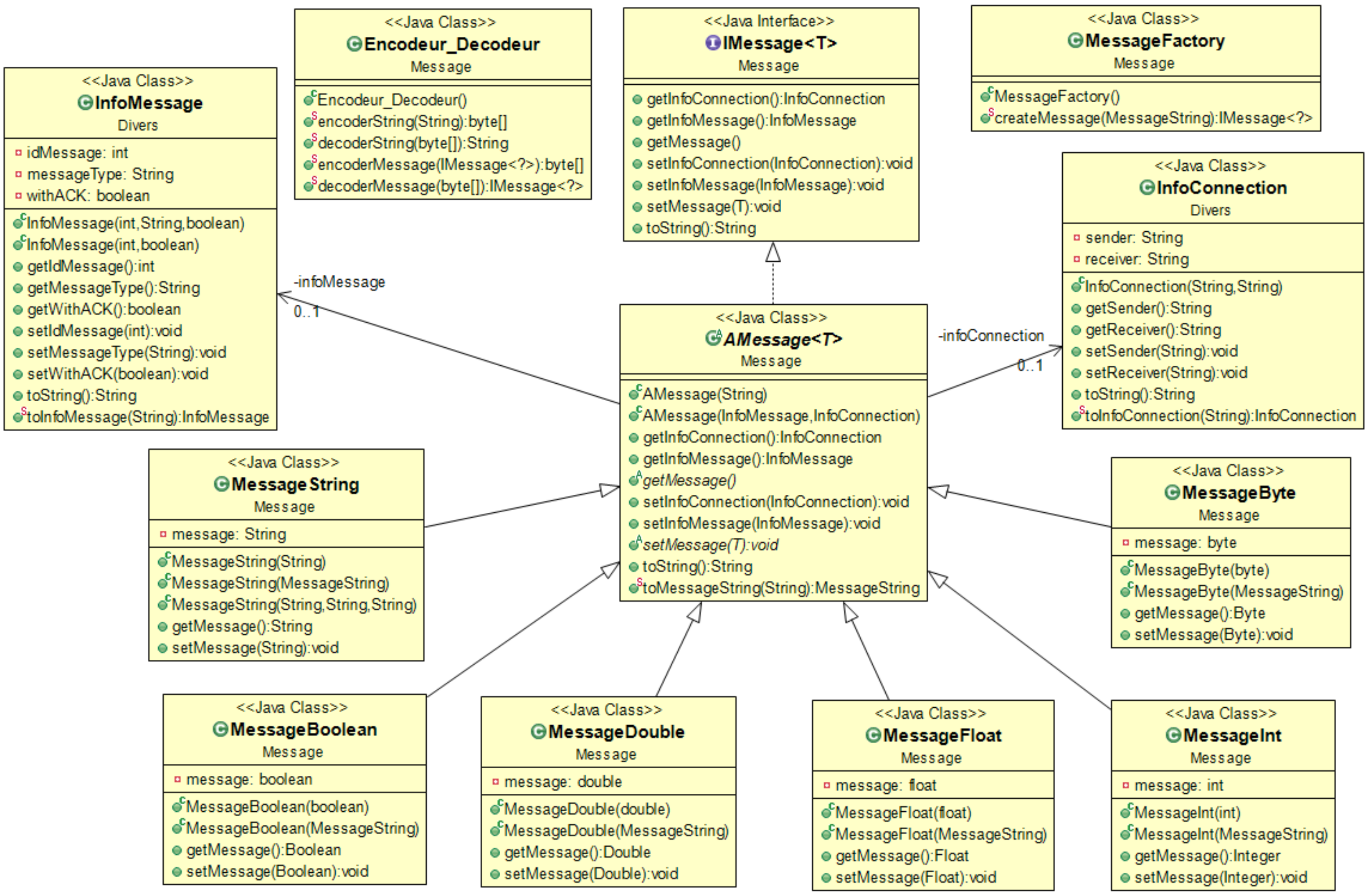


Figure 4.3: Message type class diagram

Since we don't care about the type of the message when we send or receive it, and since we receive or send a message converted into a byte, we must convert the message into the right type. To do this, we convert the message into a MessageString (by first converting the message into a String) and then, depending on the type indicated in the MessageString, we convert it into the correct type of the Message class. This is why there is a function which allows, from a String, to convert the message into a MessageString.

peut etre mettre des reference aux codes

4.3 Communication overlay

The analysis of the overlay has indicated the functions to be put in the "IConnectionCommunication" interface. These functions correspond to the primitives. During the analysis, we distinguished that there could be several types of message sending: asynchronous and synchronous. An asynchronous sending is characterised by the mechanism of delayed exchanges, which corresponds to a message sending where the system continues to perform tasks even if it has not yet received an acknowledgement of receipt for the message sent. Synchronous sending is based on the principle of real-time exchanges, which corresponds to sending a message where the system waits for the acknowledgement of the message before continuing. In our case we have to distinguish 2 asynchronous communications to be carried out, the one that sends a message and does not wait for an acknowledgement of receipt and the one that sends a message but waits for an acknowledgement of receipt. We have therefore implemented an abstract class "AConnectionCommunication" which implements the interface. An instance of "AConnectionCommunication" is composed of a timeOut which corresponds to the maximum time before sending a message if we have not received an acknowledgement, an input and output flow, an attribute representing the connection information related to the connection, and an identifier which corresponds to a counter of the number of messages, this will avoid that the messages have the same identifier. Then we had to make several sub-classes of "AconnectionCommunication" which correspond to a different service. In each sub-class we had to implement the opening and closing of the connection because it differs from one service to another. Despite the parameters of "AConnectionCommunication" expressed above, when creating an instance of this class, especially when creating an instance of one of these sub-classes, because it is impossible to instantiate an abstract class. The user does not need to provide the values of these parameters because they are initialized when establishing a connection or are initialized internally when creating an instance. The user will just have to give the parameters that allow to establish the connection of the chosen service. The message sending primitives that wait for an acknowledgement of receipt are implemented in such a way that if the message does not receive an acknowledgement of receipt in timeOut seconds, the system automatically resend the message without modifying its identifier.

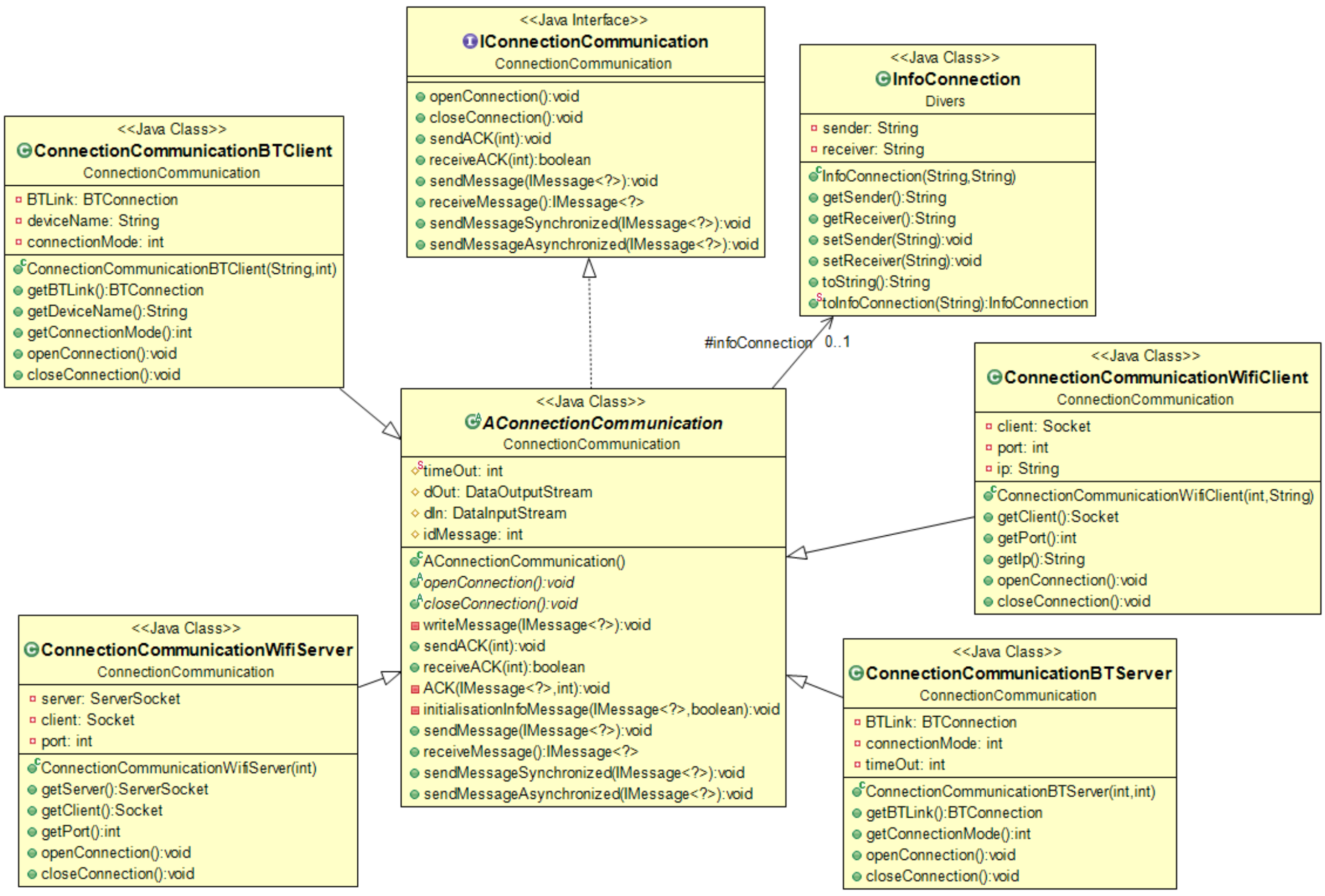


Figure 4.4: Communication overlay class diagram

To conclude, now that the communication overlay has been done, to confirm that it works properly, it has been integrated to the initial case study.

4.4 The conception of message sending

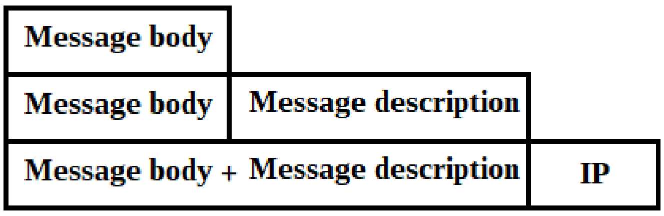


Figure 4.5: Layers of the message sent

When a message is sent by the communication overlay, it fills the different layers composing the message. However, the layer corresponding to the body of the message is filled in beforehand by the user of the overlay, then the "Message description" layer is filled in when the Message type containing the body of the message is initialised. This layer is composed of the message identifier, the type of message and whether the message should be acknowledged or not. Nevertheless this layer can be modified when using the communication overlay because it modifies the "acknowledgement of receipt" parameter according to the type of sending used (synchronous thus sending of acknowledgement of receipt, or asynchronous thus sending or not of an acknowledgement of receipt according to the primitive used). And it modifies the "identifier" parameter according to the number of messages sent by the two connected systems, to force an identifier to be unique. And finally the IP layer of the message corresponding to the IP of the sender and the IP of the receiver is filled in by the overlay just before the message is sent.

4.5 The different implementations of a Bluetooth connection

When implementing the Bluetooth server and client under the LeJos framework, we found that we could only run these programs on systems programmed under LeJos (an NXT brick in our case), and not run them on other systems programmed under other frameworks. So we had to implement a connection/communication under the Android framework to be able to use the communication overlay for the systems' remote controls. So a question arose, are there other Bluetooth client/server implementations that are different for different frameworks?

After some research, we found that there are several different implementations of a Bluetooth connection/communication depending on the framework used. For example, on the developer Apple [8] site, you can find the documentations concerning an implementation of a Bluetooth connection/communication according to Apple's Core Bluetooth framework.

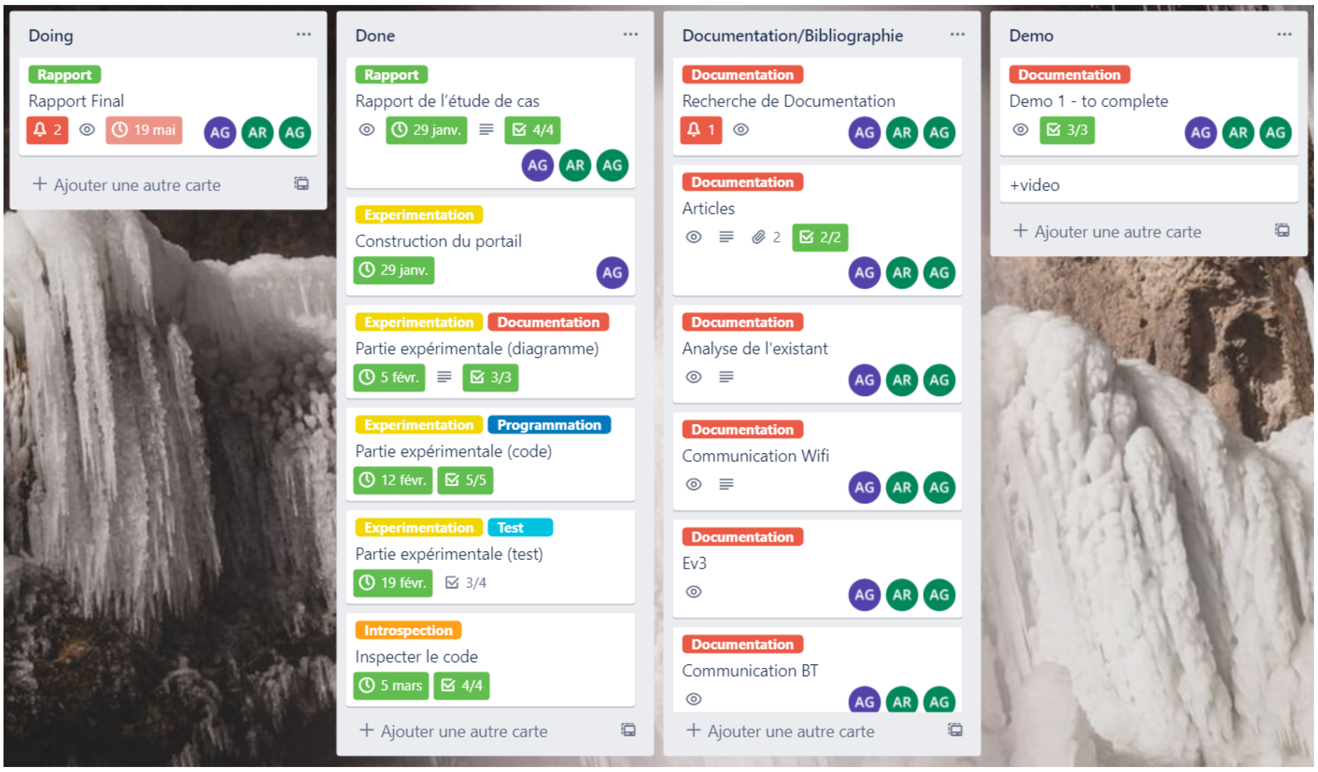
Chapter 5

Project Management

During this research project, we had to use several tools in order to communicate and manage our time as well as possible. This section highlights the tools used, but also the different difficulties encountered and the perspectives for the future of this project.

5.1 Organization

For the organisation of our working time, our supervisor Mr. Pascal ANDRÉ advised us to use the Trello software which allows us to create an interactive agenda in which we can put the tasks we have to do, that we are doing or the tasks that have been done. In addition, we can give a start and end date for each task (you can see a preview of the final Trello on Fig. 5.1). In order to be able to make progress on the project, we had at our disposal, in addition to our free time, one day per week. In addition, in order to interact with our supervisor despite the health crisis, we had a weekly meeting with him to report on the progress of the project. In addition, this meeting allowed us to clarify certain tasks that remained to be done and to express the difficulties encountered during the week, in order to help us.


Doing ···

Rapport
Rapport Final
2 · 19 mai · AG AR AG

+ Ajouter une autre carte

Done ···

Rapport
Rapport de l'étude de cas
29 janv. · 4/4 · AG AR AG

Experimentation
Construction du portail
29 janv. · AG

Experimentation · Documentation
Partie expérimentale (diagramme)
5 févr. · 3/3

Experimentation · Programmation
Partie expérimentale (code)
12 févr. · 5/5

Experimentation · Test
Partie expérimentale (test)
19 févr. · 3/4

Introspection
Inspecter le code
5 mars · 4/4

+ Ajouter une autre carte

Documentation/Bibliographie ···

Documentation
Recherche de Documentation
1 · AG AR AG

Documentation
Articles
2 · 2/2 · AG AR AG

Documentation
Analyse de l'existant
AG AR AG

Documentation
Communication Wifi
AG AR AG

Documentation
Ev3
AG AR AG

Documentation
Communication BT
AG AR AG

+ Ajouter une autre carte

Demo ···

Documentation
Demo 1 - to complete
3/3 · AG AR AG

+video

+ Ajouter une autre carte

Figure 5.1: Overview of the final Trello

In order to progress as quickly as possible in the project, we divided the different tasks to be done in the three main parts of the project, as can be seen in the Table. 5.1. In spite of this distribution, when one of us was stuck in a task, we helped each other to finish it as soon as possible in order to be able to reach the next step of the project as soon as possible.

Task		GICQUEL Alexandre	GUERIN Antoine	ROZEN Anthony
Manuel experimentation	Remote Control Connection and Portal	X		
	Remote Control Connection and Vehicle		X	
	Vehicle Connection and Portal			X
Introspection	WiFi protocol			X
	Bluetooth Protocol		X	
	Code study	X		
Communication overlay	Message type	X		X
	Overlay implementation	X		X
Improved remote controls			X	
Bibliography			X	

Table 5.1: Task allocation table

And finally we used the free forge software [9] based on git, GitLab, which allowed us to work in a collaborative way at distance. It is on this site that since the setting up of this research topic that the different TER groups of the previous years have worked (mettre ref git du projet). So we continued the project on it, especially as our supervisors and various researchers and teacher-researchers have access to it. This also allowed our supervisor to follow the progress of the project outside the weekly meetings.

5.2 Communication

During this research project we used 3 different remote communication tools. The first tool was Slack which is a collaborative communication platform and allowed us to discuss our difficulties and questions with our supervisor. In addition, on this site we had the opportunity to discuss with other researchers or teacher-researchers, but also with students from previous years who participated in this project. The second tool was Discord, which is an instant messaging program in which you can also make phone calls to several people. This software was useful for us students working on the project this year, as it was our means of text and voice discussion to progress on the project on our side, while staying in touch with our fellow students. The last tool we used was the Zoom software which is a teleconference platform. This was the software on which we had our weekly meetings with our supervisor.

5.3 Encountered difficulties

During this project we had to face several difficulties, as for example the WiFi connection between 2 EV3 because we were based on the leJos framework to make this connection. But the classes that allow such a connection in the leJos framework are not complete. So we said that it was impossible to make a WiFi connection between 2 EV3. Therefore we realized a Bluetooth connection between the 2 EV3. But because of this we had to change the Bluetooth connections between the EV3 and their remote control, by a WiFi connection because a system can not have 2 Bluetooth connections simultaneously. So after some research to make these two new connections, we found on the internet, on the site variant press [10], an example of WiFi connection code between an EV3 and a computer using the Java framework. However, we thought that maybe this WiFi connection implementation could work between 2 EV3. After some tests, this implementation was conclusive to establish this connection. Then we had to realize a data sending between the two EV3, however that did not work at first because when we send data using one of the functions *write* of the DataOutputStream class, it is necessary to add the call to the function *flush* which makes it possible to empty the output stream and to force the writing of all the bytes of output put in buffer memory, what we did not make.

On the code of the remote control of the vehicle that we recovered, we had noticed that if we press a key to move the vehicle, then the movement was carried out on the vehicle much longer than the time that we remained pressed on the key in question. We also noticed that we could only make one transmission between the remote control and the vehicle, probably because of the longer movement on the vehicle. We also noticed that an error was displayed on the brick after a certain execution time, and we thought that the 2 errors were related (the display error could come from an overload of the data flow related to the data sent by the remote control). So we decided to copy the code of the vehicle remote control on the one of the gate remote control, as the latter had no problem. However, the movement of the vehicle will be less controllable. Indeed, we have chosen to perform the action requested by the remote control on 1 second, to avoid problems of overloading the flow. Despite these changes, the display error still occurred. However, we noticed that once the error appeared, the communication between the remote control and the vehicle was no longer blocked to only 1 transmission, we could move the vehicle in the way we designed it. On the one hand you could move the vehicle freely, but on the other hand you could not observe anything on the EV3 brick.

We also had to face the difficulties linked to the Covid-19 epidemic because we couldn't meet every week due to confinement and contact cases. This caused some delay because for these tests we had to have access to several EV3 at the same time. And even if we had our EV3s ready, we could not perform some of these tests because there was only one computer in our group that supported the execution of the Android Studio IDE.

5.4 Perspective

As explained above, due to the health crisis we had some delay. And due to this delay, we could not test several things like the new remote control that allows the portal to be the WiFi server, like the fact that is the information related to a Bluetooth connection is correctly filled, but also the test of the modified case study with the communication overlay. So it remains to test these parts of the project. The implemented communication overlay allows a WiFi server to accept one and only one client. But a WiFi server can accept several clients at the same time. However in the prototype file of the code-java file, of the TER of this year is several track of implementation of this new overlay including a server which accepts several client. So the next group that will work on this research topic will have to choose the best implementation of this new overlay. Now that the case study and the first abstraction have been done the next group will have to study how to implement the communication overlay in different projects where communications are involved (how to implement the overlay in a UML model). Then, they will have to deduce a first prototype of model transformation from the generalization of the UML models made.

Chapter 6

Conclusion

This research project was our first contact with the field of scientific research. Therefore, we were a bit lost at the beginning and did not really know where to start or what steps to take. But our supervisor Mr. Pascal ANDRÉ helped us to understand the expectations of the subject, and pointed us in the right direction when we were stuck. This project allowed us to understand how work is done in a research laboratory and it allowed us to put our engineering knowledge into practice. Indeed, in addition to researching how to perform a certain task, we also had to think about the best possible implementation in order to achieve the best possible abstraction of communications. However, we also had to think about the best implementation of the case study, so that it would be as close to reality as possible. Moreover, this project allowed us to prepare for our Master 2 internship. Because even if it was a research project, we had to carry out certain tasks within a time limit, trying not to exceed it or not to exceed it too much, we had to follow a guideline to reach a precise goal. Concerning the project, we were happy to have had the opportunity to work on it because it is a project that will help us later on in our job as a developer, because it will allow developers to focus on more complex and specific tasks.

Bibliography

[1] André, P.: Case studies in model-driven reverse engineering. In: Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2019, Prague, Czech Republic, February 20-22, 2019. (2019) 256–263

[2] André, P., Tebib, M.E.A.: Refining automation system control with MDE. In Hammoudi, S., Pires, L.F., Selic, B., eds.: Proceedings of MODELSWARD 2020, Valletta, Malta, February 25-27, 2020, SCITEPRESS (2020) 425–432

[3] André, P., Vailly, A.: GÉNIE LOGICIEL - Développement de logiciels avec UML 2 et OCL - Cours, études de cas et exercices corrigés. Edition ellipses edn. Volume 6. Collection Technosup (2013)

[4] Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML: Systems Modeling Language. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008)

[5] Vallé, F., Roques, P.: UML 2 en action: De l'analyse des besoins à la conception. Collectio Architect Logiciel (2011)

[6] : Analyse bluetooth protocols on windows using wireshark. `https://tewarid.github.io/2020/08/20/analyze-bluetooth-protocols-on-windows-using-wireshark.html`

[7] : Android studio documentation. `https://developer.android.com/studio/run/emulator#wifi`

[8] : Apple framework documentation. `https://developer.apple.com/documentation/corebluetooth`

[9] : Explanation of what a forge site is. `https://en.wikipedia.org/wiki/Forge_(software)`

[10] : Wifi connection between 2 ev3. http://variantpress.com/books/maximum-lego-mindstorms-ev3/

Appendix A

Implementation

A.1 Architecture

The communication overlay is composed of 5 related components. The "IConnectionCommunication" interface is implemented in the abstract class "AConnectionCommuniction". And this one is declined in 5 sub-classes:

- The "ConnectionCommunicationWifiServer" sub-class
- The "ConnectionCommunicationWifiClient" sub-class
- The "ConnectionCommunicationBTServer" sub-class
- The "ConnectionCommunicationBTClient" sub-class
- The "ConnectionCommunicationBTAndroidClient" sub-class

This communication overlay uses the "InfoConnection" class as a type for an attribute. Furthermore it uses the "Encodeur_Decodeur" class to encode a message when sending or to decode a message when receiving. You can find the class diagram of this communication overlay on Fig. 4.4

The communication overlay uses a Message type, which has been implemented as follows. The Message type is composed of 8 linked components. The "IMessage" interface is implemented in the abstract class "AMessage". And this one is declined in 6 sub-classes:

- The "MessageString" sub-class
- The "MessageInt" sub-class
- The "MessageBoolean" sub-class

- The "MessageDouble" sub-class
- The "MessageByte" sub-class
- The "MessageFloat" sub-class

This Message type uses the "InfoConnection" and "InfoMessage" classes as the type for two of their attributes. In addition, a design pattern has been used, Factory, to create an instance of Message from a message type. You can find the class diagram of this Message type on Fig. 4.3

A.2 The role of classes

A.2.1 Communication overlay

In the communication overlay, the "IConnectionCommunication" class corresponds to the interface grouping all the primitives. The "AConnection-Communication" class is the abstract class implementing the "IConnection-Communication" interface. This class implements certain primitives which are general to all types of communication. Several classes are inherited from the abstract class "AConnectionCommunication", each of these sub-classes correspond to a type of communication:

- The "ConnectionCommunicationWifiServer" sub-class corresponds to a WiFi type communication for a system having the role of server
- The "ConnectionCommunicationWifiClient" sub-class corresponds to a WiFi type communication for a system having the role of client
- The "ConnectionCommunicationBTServer" sub-class corresponds to a Bluetooth type communication for a system having the role of server, under the leJos framework
- The "ConnectionCommunicationBTClient" sub-class corresponds to a Bluetooth type communication for a system having the role of client, under the leJos framework
- The "ConnectionCommunicationBTAndroidClient" sub-class corresponds to a Bluetooth type communication for a system having the role of client, under the android framework

All these classes correspond to the communication overlay, which provides a first layer of abstraction for communications. This means that we no longer have to worry about the type of communication used.

A.2.2 Message type

In the Message type, which corresponds to a generic message type, the "IMessage" class corresponds to the interface grouping all the functions related to a message or to the content of a message. The "AMessage" class is the abstract class implementing the "IMessage" interface. This class allows to implement the functions common to any type of Message. Several classes are inherited from the "AMessage" abstract class, each of these sub-classes corresponds to a type of message:

- The "MessageString" sub-class corresponds to a message of type String
- The "MessageInt" sub-class corresponds to a message of type int
- The "MessageBoolean" sub-class corresponds to a message of type boolean
- The "MessageDouble" sub-class corresponds to a message of type double
- The "MessageByte" sub-class corresponds to a message of type byte
- The "MessageFloat" sub-class corresponds to a message of type float

The "MessageFactory" class allows you to create a Message type from a message type given in a "MessageString" object (even if the "MessageString" object must contain a message of type String, in certain cases to facilitate implementation, we have converted a message into a "MessageString" even if it was not of type String).

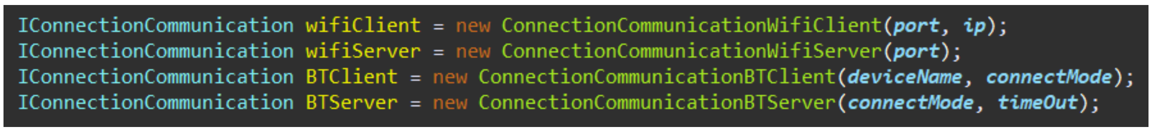
The "Encodeur_Decodeur" class is used to encode or decode a message.

The "InfoMessage" class groups all the information related to a message into one type.
The "InfoConnection" class groups all the information related to a connection into one type.

Appendix B

Primitive Language

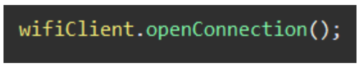
To be able to use the communication overlay, you must first initialise an instance of this overlay in the following way depending on the type of service and role you want to use:



```
IConnectionCommunication wifiClient = new ConnectionCommunicationWifiClient(port, ip);
IConnectionCommunication wifiServer = new ConnectionCommunicationWifiServer(port);
IConnectionCommunication BTClient = new ConnectionCommunicationBTClient(deviceName, connectMode);
IConnectionCommunication BTServer = new ConnectionCommunicationBTServer(connectMode, timeOut);
```

Figure B.1: Overlay instantiation

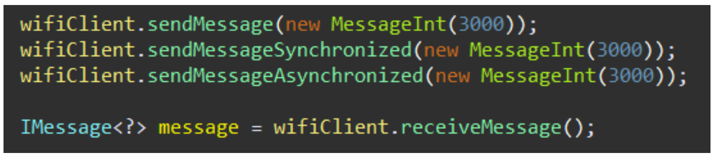
Then before you can send or receive a message, you must first establish a connection, as in the following example:



```
wifiClient.openConnection();
```

Figure B.2: Establishing a connection

Then you can send or receive a message by doing this:

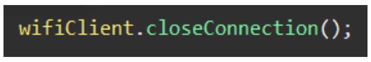


```
wifiClient.sendMessage(new MessageInt(3000));
wifiClient.sendMessageSynchronized(new MessageInt(3000));
wifiClient.sendMessageAsynchronized(new MessageInt(3000));

IMessage<?> message = wifiClient.receiveMessage();
```

Figure B.3: Sending or receiving a message with the overlay

And finally you can close the connection as follows:



```
wifiClient.closeConnection();
```

Figure B.4: Closing a connection