UNIVERSITÉ DE NANTES

TER Report

Refinement of communication protocols by models' transformation

---

Cousseau Axel · Haiti Aya · Juzdzewski Matthieu

Master 1 ALMA
Academic year 2019-2020

Supervisor : Pascal Andre
Team AeLoS, LS2N

Université de Nantes – faculté des Sciences et Techniques (FST)
2 Chemin de la Houssinière
BP 92208, 44322 Nantes Cedex 3

Acknowledgements

We would like to express our sincere gratitude to our Supervisor Dr Pascal André from the LS2N (Laboratory of Digital Sciences) team for providing his valuable guidance, comments and suggestions throughout the course of this project. His assistance both theoretically and practically during our weekly meetings before lock-down allowed us to carry out our project and keep us on the right path during the hard period everyone was going through. We also would like to thank him for constantly motivating us to work harder and for being supportive while we encountered problems and difficulties.

We would also like to acknowledge the valuable scientific discussions with our colleagues from the "Refinement of statecharts in Java by model transformation" research group. Although our projects were complementary, these discussions helped us foster a scientific exchange at an early stage of the research.

Last but not least, we would like to thank Vivian Sicard for taking the time to manage the keys and giving us access to the LS2N facility.

The internship opportunity of working with a mentor, an experienced researcher in our case, was a great chance for us to hone our problem-solving skills. This project introduced us to the overall strategy of scientific work, from identifying the problem to designing the experiments and interpreting the results while planning a strategy to solve every problem that comes our way during the process.

Contents

Chapter 1

Introduction

During the second semester of our master's degree's first year (ALMA), we had to chose a subject among a given set and work as a 3-person team to get a first initiation of the world of research in computer science and to sharpen and test our skills.

We chose the following subject : "Refinement of communication protocols by models' transformation" which is a subpart of the much larger project "Automation in Model Driven Engineering". Supervised by Dr Pascal André, our goal was to study the transformation of models written in UML (mainly statechart and sequence diagrams) to usable Java code. More specifically, we focused on the communication aspect : the remote communication via wifi, bluetooth or USB and the exchange of messages within the system, like method calls for example. We take the UML message as an input and refine that message to the corresponding source code. A second team was tasked with transforming state machines into OOP models. As we need their work to complete our own, we will consider their part as already completed and we will manually code what we would have gotten from them.

So the goal of our team is to find a way to generate Java source code from the communication aspect of a given UML model. To get a physical support on which we can experiment and test our solutions, the LS2N has provided us with a complete LEGO Mindstorm kit and an EV3 controller. The path we have chosen to take, which will represent the three main chapters of this paper, can be resumed as follows : first we will try to understand and frame the problematic by building a simple garage door with the LEGO kit, remotely controlled through the EV3 connected to a computer via USB and wifi. Then we will analyse how the system works by experimenting different approaches and solutions for the refinement process. Finally, we will discuss our solutions and present our findings based on our experience.

Getting started with the project

1.1 Project management

Below is the Gantt diagram presenting the various tasks we achieved throughout the semester. The diagram itself has been made after the facts, however we did have a general idea of what we needed to do. This diagram is just the representation of the informal timeline that we had in mind.

During the first part of the semester, we had team meetings every Friday to work on the project and set goals for the week after. We adapted our schedule during the lock-down with vocal meetings on Discord about twice a week and written communication for planning and details.

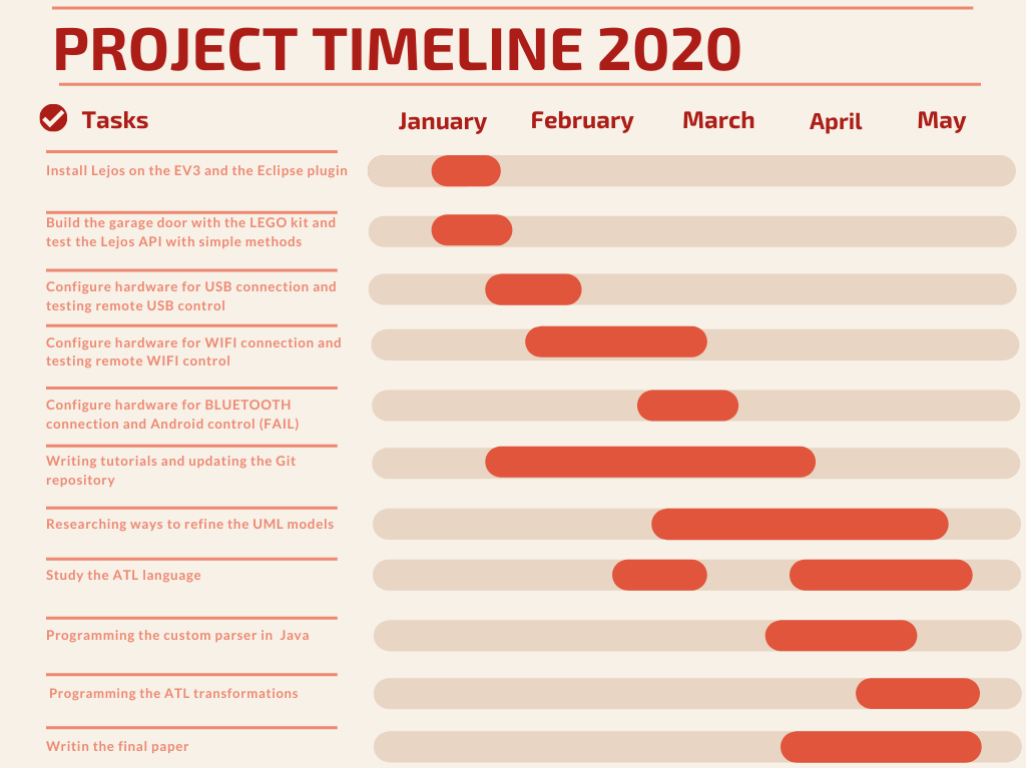


Figure 1.1: Gantt diagram

1.2 Available tools

We have a LEGO EV3 controller and a full LEGO toolkit to start experimenting with a simple case study : a garage door. We will code with Java and Eclipse using the Lejos API. The EV3 needs a 2 to 32 GB SD card to install the Lejos virtual machine.

Figure 1.2: EV3 Controler

1.3 Study case : garage door

In order to better understand the problem of automation and automatic code generation, we based our experiments on the concrete case of a garage door controlled by a remote control or application communicating with various possible means (Wifi, bluetooth, USB cable).

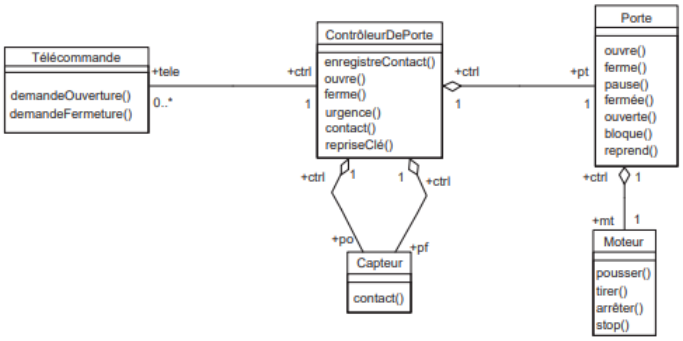


Figure 1.3: Garage door class diagram

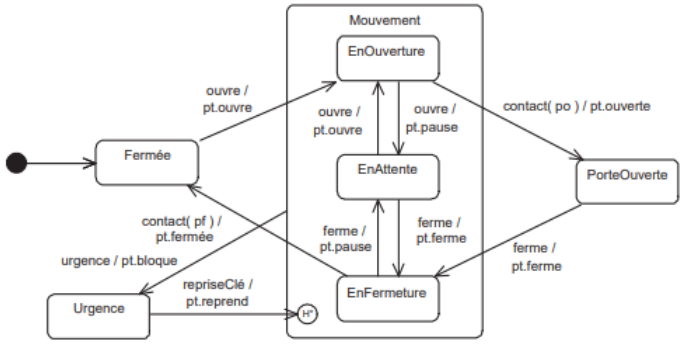


Figure 1.4: Garage door state diagram

This garage door consists of a remote control, a sensor (open/close), a door and a motor. A controller manages these different parts. This example is perfect for our problem, because in addition to being very concrete and familiar, communications have a central place here.

We have chosen to simplify the model to work on the communications between remote control and controller (client and EV3 in our case) which will be remote communications, Wifi or USB, and the communications internal to the controller, i.e. method calls to control the motors or the sensor.

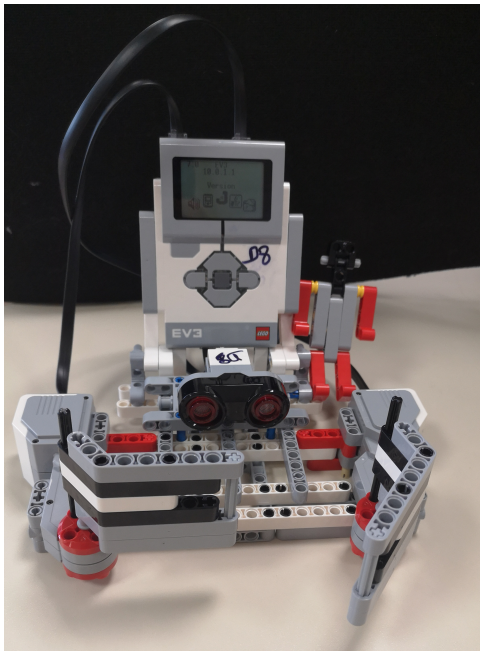
Chapter 2

Experimenting with the EV3 and Lejos

In this section, we will present how we built the garage door and the experiments we conducted using various communication protocols. Tutorials and advice on how to reproduce what we did step by step will be given, either in the next sections or in the annex part at the end of the paper.

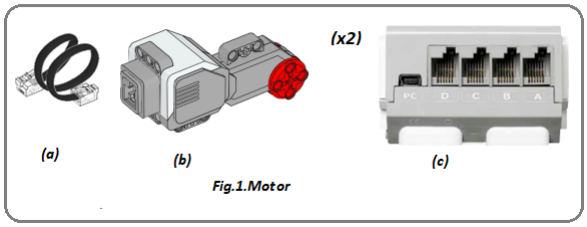
2.1 Building the LEGO robot

As it was the first time we encountered the Lejos framework and the LEGO hardware, we decided it would be best to simplify the study case and get going faster, with a simple but overall complete robot (enough for our purposes at least). Some decisions, specifically the building of the robot, are based solely on cosmetic, arbitrary reasons. This is why we decided to build a garage door with two moving panels instead of just one (as shown in the study case). This means that we have two motors, one for each door. We also decided to use a different kind of sensor to check whether the doors are open or closed. Below is a picture of the final garage door.

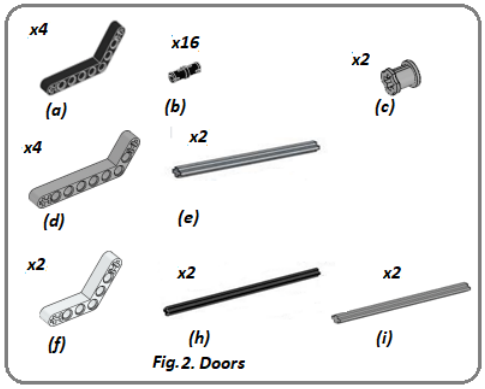


The robot consists of two motors, both of which incorporate a rotation sensor that is accurate to within a degree, optimized to serve as the motor base for the robot. They rotate with a rotational torque of 20 Ncm and a blocking torque of 40 Ncm, slow but powerful enough motors. It must also be taken into account that the action of the two motors is coordinated. The part « Fig.1.Motor (b) » is one of the two motors used. In order to control the forward or reverse rotation of a motor, the motor is connected to one of the four output ports of the controller « Fig.1.Motor (c) » using the flat cable « Fig.1.Motor (a)

». For this robot the motors have been connected to ports 'A' and 'D' as you can see in the code in the following sections.



Each of the doors is formed from two black angled beams with 4x6 modules « Fig.2.Doors (a) » and two grey angled beams with 3x6 modules « Fig.2.Doors (d) », and finally between the two pairs (grey, black), each is connected by two black friction pins « Fig.2.Doors (b) », a white beam with 4x4 modules « Fig.2.Doors (f) » is placed. Each door is fixed at its first end by a 10 black module pin « Fig.2.Doors (h) » and on the other by a 7 grey module pin « Fig.2.Doors (i) ». And finally with a grey 5-module shaft « Fig.2.Doors (e) » and a one-module ring « Fig.2.Doors (c) » each door is fixed to the motor's rotation axis. The final result of the doors is shown in « Fig.2.Doors (j) ».



The support that connects the motors to the controller and ensures the balance of the robot consists of the following parts. Firstly, by means of two red bent beams with 2x4 modules « Fig.3.Robot support (a) » fixed to each motor by means of a beige peg with 3 modules « Fig.3.Robot support (d) », the two motors are connected by means of 2 beams with 15 white modules « Fig.3.Robot support (e) », while the « Fig.3.Robot support (a) » is fixed to the motors by means of a red pin with 2 modules « Fig.3.Robot support (c) ». Thanks to the two 5x7 module frames, the grey « Fig.3.Robot support (f) » is connected to the rear part of the support with four black friction pins to the front part. The front part of the support is also fixed by two beams with 15 white modules using four black pins, two pins for each beam. In order to secure the front part that is closest to the rotation support of the two motors on the floor, two grey 13-module beams « Fig.3.Robot support (g) » are connected to each of the motors with 4 black pins on each side.

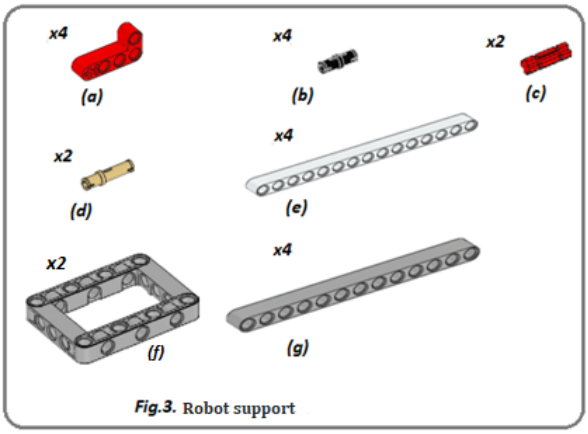


Fig.3. Robot support

In order to detect whether the doors are closed or open, an ultrasonic sensor has been installed to measure the distance between the sensor and the doors using reflected sound waves « Fig.4.Sensor (a) ». Like the engine, the sensor is also connected to the brick by the cable shown in « Fig.4.Sensor (b) » to one of the ports 1, 2, 3 or 4 « Fig.4.Sensor (c) ». The sensor is then attached to the bracket by the rest of the parts « Fig.4.Sensor (d)(e)(f)(g) ».

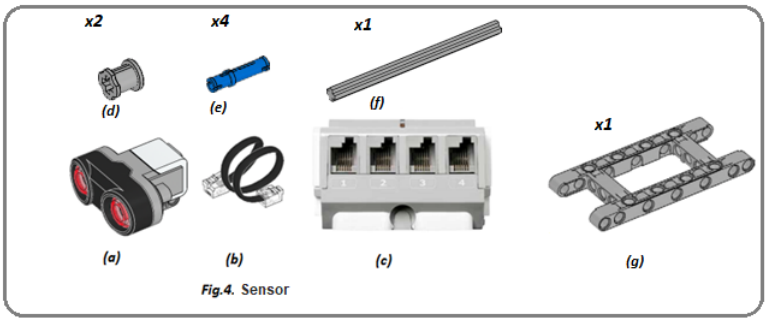
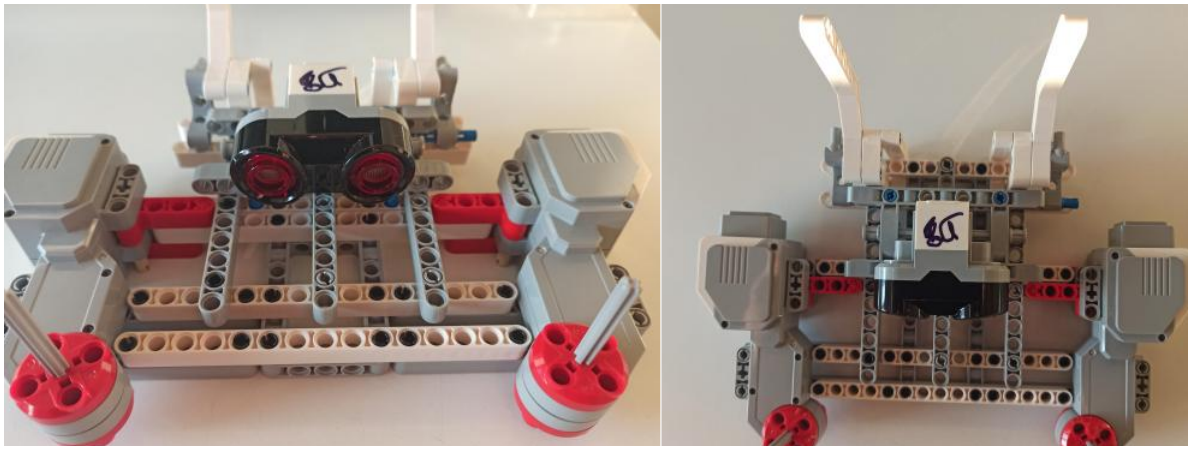


Fig.4. Sensor

Although the robot is functional with these components, we still chose to add a support in order to integrate the controller to the robot.



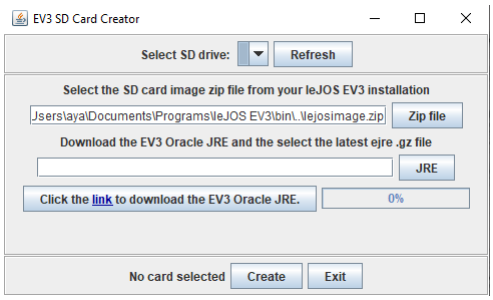
2.2 Installing Lejos and the eclipse plugin

Before you can start experimenting with the robot, you need to install Lejos on the EV3 controller as well as the Lejos plugin on Eclipse.

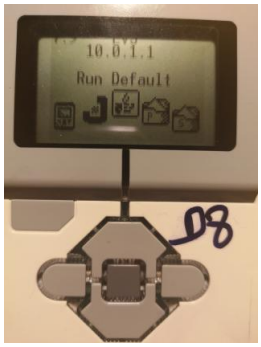
2.2.1 Installation of the LeJOS Framework on the EV3 brick

Originally, the EV3 brick boots on the Mindstorm OS. However, this one doesn't allow to program the behaviour of our robot in Java: so we have to install Lejos (which is a kind of Java virtual machine) on the EV3. This step can be tedious because of a sometimes ambiguous documentation and outdated tutorials. We have therefore described in detail the steps we followed ourselves. You will find this tutorial in « Appendix A » with solutions to possible problems.

First of all, in order to install the LeJOS Framework on the EV3 brick, you will need an SD card between 2 and 32 GB. We worked with a 2 GB card. Preferably, it should be empty and in FAT32 format. Then, you will have to download « leJOS_EV3 _0.9.0 beta » on the official website « lejos.org », more precisely by clicking on leJOS EV3 => Downloads. After launching the application and following the installation, the page on the figure below will be displayed.



You must also prepare your Java development space and the IDE that suits you, IntelliJ or eclipse for example. For the rest we chose eclipse. Once this is done, you will need to insert first a micro SD to the SD card adapter that you will then insert on your computer, and select your card on 'select SD drive'. Then, you just have to click on 'Click the link to download the EV3 Oracle JRE' which will redirect you directly to the oracle site for Java for LEGO Mindstorms EV3. The second download has been chosen: 'Oracle Java SE Embedded version 7'. And finally, before clicking on 'Create', don't forget to indicate the path to this last download for the JRE field. Now you just have to remove the adapter from the SD card and insert your card into the EV3 controller box and turn it on. A new default IP address will now be displayed: 10.0.1.1.



2.2.2 Installing the EV3 plugin on Eclipse

To be able to create a Lejos project on Eclipse, a Lejos project is not a traditional Java project, and manipulate the robot's features, you need to install the Lejos plugin on Eclipse. The tutorial is on the same file mentioned above.

2.2.2.1 Methodology

Generally speaking, the development process goes like this: In order to install the LeJOS plugin on eclipse to be able to run the upcoming code, it's simple, as for any other plugin, you go to the eclipse menu, Help and click on Install New Software. On the window that appears, enter in the first field the following URL: http://lejos.sourceforge.net/tools/eclipse/plugin/ev3. You then select the LeJOS

EV3 plugin and launch its installation, keeping the default settings afterwards. In order to connect to the EV3 brick later, simply go to Window then preferences and on LeJOS EV3, tick connect to named brick and enter the previous default IP address then Apply and OK. You can now create your first LeJOS EV3 Project by going to File, new Project, LeJOS EV3 Project and add a new library of the type « LeJOS Library Container ».

In order to quickly test if our installation is correct, we started by loading a simple program (see Appendix C.2) on the EV3 box. There was no user interaction, just an automatic activation of the motors.

2.3 Communication

In this part, we will discuss the remote communication aspect between the robot (here the garage door) and the controller (here a PC). In particular, we will see the technologies at our disposal: USB, Wifi and Bluetooth.

2.3.1 USB

After having prepared your working environment according to the previous chapter, it is time to get your EV3 controller and the USB cable provided in the Lego box. At first, we experimented with controlling the robot via USB. In order to establish any exchange between your PC and your EV3 controller, you will have to start by connecting the two of them with the USB cable. Once the controller is connected to your machine, unlike Linux where it finds and loads the driver, on Windows, you will have to do it manually. Once this is done, the controller is now connected to your computer and the driver takes care of managing the low level inputs and outputs to this device.

In order to load our program (Appendix D) on the EV3 box, we used Java sockets to implement the Server/Client behavior. A socket is a combination of an IP address and a port number that is used in the Layer 4 transport of the OSI model to establish the connection between the two devices. It is this connection that will allow the exchange of data packets using the USB (Universal serial Bus) communication protocol. When the connection is established, the client can then send *orders* (in the form of String) to the box which will interpret these Strings and call the corresponding methods.

However, it is once again necessary to configure certain parameters before the USB connection is recognized. A tutorial that explains in detail how to set up USB recognition is available in Appendix (see Appendix C).

Finally, it should be noted that the risk of USB communication is at the heart of its operating mode. Any application within the computer that can read the exchanges and damage the integrity of the exchanged data represents a major risk since there is no principle of rights as for applications on Android. There are some mechanisms that may protect from a malware attack and ensure the security of data being transferred outside the system's environment. Encryption is a part of these mechanisms; it is used to encrypt files before transferring them to USB device as a part of the data loss prevention policy.

2.3.2 WiFi

The methodology for establishing a Wifi connection is essentially the same as for the USB part, i.e. a server program is loaded on the EV3 box (see Appendix D). It waits for a client to connect. A program is then executed on the PC (see Appendix G). It connects to the server and can then send commands. The only technical difference between Wifi and USB is the value of the IP address used.

After putting the three devices (Phone, PC and EV3 controller) on the same network by sharing the 4G connection on the phone, the connection between the PC and the box which is in server position on the network is then established, using the "Telnet" protocol. The Telnet protocol is a bidirectional

exchange protocol of the application layer of the OSI model based on the TCP protocol that allows communication with a remote server for an exchange in text format.
Since any communication using the Telnet protocol is plaintext over the network, multiple interceptors on the network can affect the security of network exchanges. It is for this reason that encrypted protocols such as SSH have been developed to replace Telnet to encrypt the data exchange and thus avoid any data breach.

Once again, it is necessary to first configure the hardware before you can claim to use wifi. In Appendix F you will find a tutorial that allows you to follow this configuration step by step. We have worked a lot to make the wifi work, and tried many different methods. Therefore, our tutorial is a mixture of all our discoveries, and it may not be *strictly* accurate. However, it does give a general idea of how to proceed.

2.3.3 Bluetooth

The latest remote communication technology at our disposal is Bluetooth. We started by developing a basic Android application that contains basic commands from the following buttons: connect, open, close and quit. This was done in order to test another control tool but several configurations were necessary. We were not able to get Bluetooth communication working in this time before confinement (February 2020) and the person who currently owns the robot does not have the necessary hardware to move forward on that side. Therefore, we leave this part open for future groups.

2.4 Results

To conclude the experimentation part with the EV3 box and the LEGO robot, we can notice that the biggest difficulty is to install and configure correctly all the necessary parameters and software. We hope that the tutorials we have written will be useful for the groups ofthe following years to get started more quickly. To sum up, we managed to control the robot remotely via a USB and Wifi connection. We also experimented with the basic methods of the Lejos libraries, including the use of motors and the ultrasonic distance sensor.

Our implementation of the garage door differs from the UML model proposed by our supervisor Pascal ANDRE, in the sense that we have a double-leaf door that does not use a pressure sensor but an ultrasonic distance sensor. The rest of this report is based on our implementation of the door, but it is quite possible - we think - to easily migrate to a model closer to the initial UML scheme with pressure sensors.

Chapter 3

Automation

After experimenting with the robot and the Lejos API, we were ready to start thinking about the automation of code generation from UML models. We did try to approach the problem from a purely theoretical perspective, but at the time we were still unclear about how to do it and where to start. So we decided to try an other approach : on one hand, we worked backward from the final Java code to the initial UML model while on the other hand, we still tried to refine the UML models.

3.1 Methodology

Before we begin, here is a reminder of our goal : we want to be able to generate automatically complete or partially complete Java code from one or more UML models. It is important to note that the goal is not necessarily to generate **all** the code but as much as possible, especially the repetitive or common sections we often find from one project to the other. Here is a representation of the work methodology we discussed above.

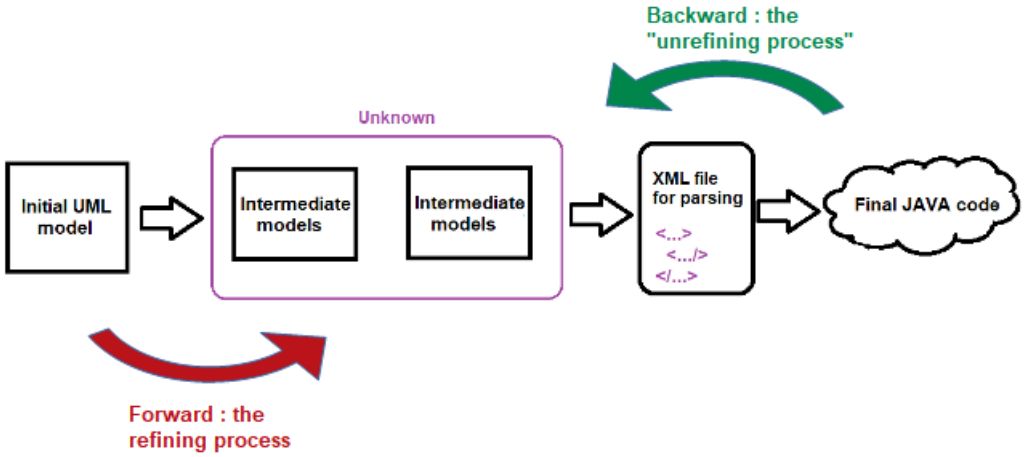


Figure 3.1: Research workflow

The idea behind this methodology is to have a more practical minded approach that going from finish to start allows. This way, we can find out the different constraints and devise solutions as we go. It is also less confusing because we start from a position of strength (we are familiar with the final Java code and we can explain it because we wrote it ourselves) and we progressively get to higher levels of abstraction.

The first idea is to have a link between the Java code and the UML models that we understand so that we can quickly start working towards our goal. We chose to use an XML file : it is flexible, yet powerful, and last but not least it is how UML models are represented internally. So from the backward

perspective, we want to have a representation of the final Java code as an XML file that contains all the data necessary. Then we want to abstract this XML file until we have the initial UML model. Basically, it is a matter of "*unrefining*" : we look at the information that we currently have that the initial model doesn't, and we cut it. By finding out in which order we can do this, we get layers of refinement.

This means that from the forward approach, we should be able to go through these layers, refining and adding information to finally get to the XML link. At this point, a simple parser is enough to generate the Java code by mapping tags and attributes with pre-written code. The process has two main parts :

- refining the initial UML model until we have the XML link
- parsing the XML file to get the Java code

Following is the presentation of what we managed to achieve from a technical point of view. Obviously, there is still a lot more to do and we will discuss our findings in the next and last chapter. We will see what remains to be done and improved, what are the layers of refinement that we identified and other considerations for future development. But first we present here what we currently have which is a simple, but functional, transformation method from an initial UML model to an executable Java code.

3.2 UML to XML

For this part, we decided to use sequence diagrams as source files, believing that it suggested a maximum of communications between the different actors.The goal here is therefore to transform this source file into an XML file created to be simple and composed of the only information needed to generate Java code, based on the information found in a UML diagram. To do this, we used ATLAS Transformation Language (ATL), a model transformation language available as an Eclipse plugin. To carry out the complete transformation process we proceeded in several steps : first the creation of the UML model (sequence diagram) and its XML correspondence, then the creation of the metamodels representing our source and output files, and finally the writing of the ATL transformation rules.

3.2.1 Creation of the UML file

To model our UML diagram we used the Eclipse Papyrus plugin, a graphical editor for UML. Here we have modeled a very simple sequence diagram based on the model used in our experiments. We can see the client's connection message, and the different possible method calls. Once this diagram is finished we retrieve the .uml file generated by Papyrus in the same directory, it is a file describing our diagram in XML format, it will be used as source file during the ATL transformation.

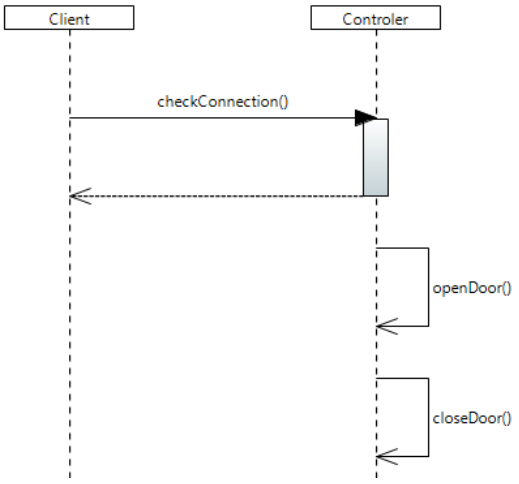


Figure 3.2: Sequence diagram representing the study case

3.2.2 Creation of metamodels

A metamodel is an abstraction to describe the structure of a model. To perform a transformation with ATL we need the metamodels of the input and output models to describe exactly what it will be possible to have as input and how to transform it into an output. Here for example for our .uml file representing the sequence diagram, each tag in the file is represented by a class in the metamodel, and if a tag contained another one, its class will contain an attribute corresponding to the class of this tag in the metamodel. To realize these metamodels we use .ecore files, an EMF model. The metamodel thus made must be able to match any input sequence diagram without modifications.

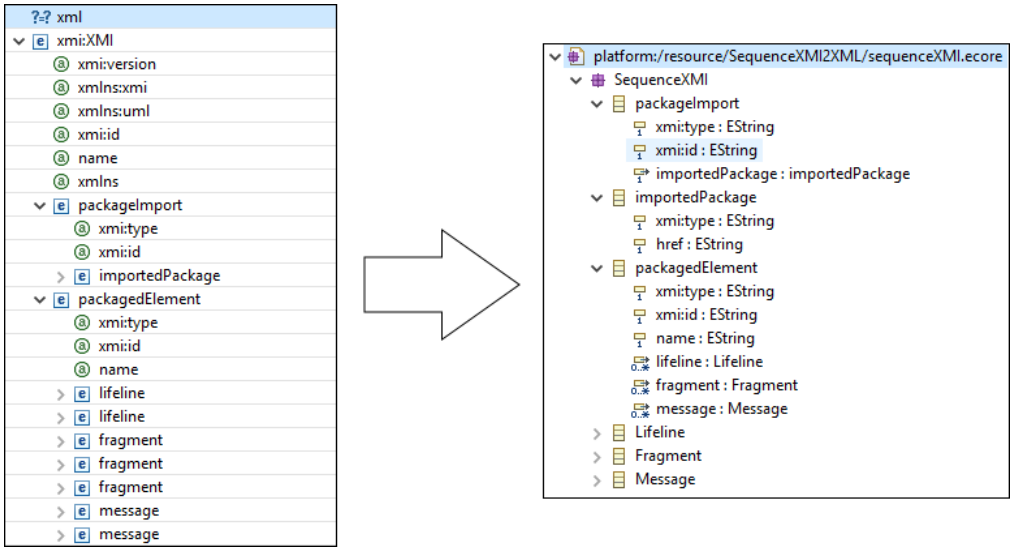


Figure 3.3: xml model to ecore model

In the same way, we produce a metamodel representing the XML file used to generate the code.

3.2.3 Writing transformation rules

The ATL transformation part was the most complicated, as it required a lot of research and documentation concerning the use and syntax of the ATL language. In our ATL file we specify which are the input and output models, using the metamodels created in the previous step to inform ATL about what we have at the beginning and what we want as a result, and then we write the transformation rules. The goal is to link each part of the output model to a part of the source model, and to "explain" how to create them.

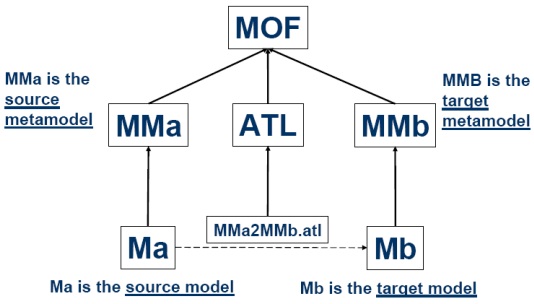


Figure 3.4: ATL Context
[1]

A rule is first composed of a "source" section, with the keyword *from*. This is used to inform the type of the element used and its model, we can also add conditions to restrict the set of elements affected by the rule, and we can declare local variables to the rule.

The rule then consists of the "target" section, with the keyword '*to*, the type of target element and its model are filled in. Note that several targets can be created for the same source. (figure 3.5.

```
lazy rule lifeline2class{
    from
        s : SequenceXMI!Lifeline
    to
        t1 : XML!Class(
            name <- s.name,
            hardware <- '',
            Fonctions <- t2,
            ConnectionInfo <- t3
        ),
        t2 : XML!Fonctions(
            fonction <- SequenceXMI!packagedElement.allInstances().first().message
                ->select(m | m.isContained(s.coveredBy))
                ->collect(m | thisModule.message2fonction(m))
        ),
        t3 : XML!ConnectionInfo(
            type <- '',
            role <- '',
            ip <- '',
            port <- '',
            inputType <- ''
        )
}
```

Figure 3.5: ATL Rule Transforming a Lifeline element into a Class element in the output XML

Here for example, for each Lifeline type element of the SequenceXMI source model, three elements will be created in the target model: Class, Functions, ConnectionInfo. The attributes of these elements are then filled in either by the content of other targets as seen in "Class", or by elements already present in the source model, or by the result of other rules or Helper as in "Functions". A Helper is a method which can be defined and then called in ATL.

Note that in order for ATL to recognize the source file, and for the transformation to work, it was necessary to replace the *<uml:Model>* tag by a tag *<xmi:XMI>*, it was also necessary to add the "xmlns=[MetaModelSourceName]" attribute. These are the only modifications made on the source file after its creation in Papyrus.

3.2.4 Result

We were able to generate an XML file close to the expected result, nevertheless some parts are missing because no information about them is filled in the source UML diagram. This is the case, for example, of the connection information (IP, port, type), which will probably have to be added by hand or using a configuration file, but also of the name of the action assigned to each function, and the content of the Mapper specific to the connection and the function call. To go further, one may wonder if it is possible to add this missing information using other transformations.

Concerning the realized transformation, it works on sequence diagrams, another possible opening is to search if it is possible to generalize the ATL transformation so that it can take as input any UML diagram, or if it is mandatory to realize different transformations for each diagram in order to reach the more global XML model we have defined.

3.3 XML to java

As discussed previously, this is the last step of the code generation process. In the previous step, we refined the UML model to the XML link using the Atlas Transformation Language. This last XML file now contains all the data necessary to generate the code. The only thing left to do is to use the data contained in the XML file to generate the code. To do so, we decided to write our own custom program : we will feed the XML file as an input and the program will generate the code (in the form of a .txt file at the moment). Our program is heavily **component-oriented** : each tag from the XML file is processed by a dedicated specialized class. Each class only cares about its role and nothing else. The result is progressively stored in a List<String> where each element is composed of similar data :

- At the index 0, we find all the import statements. Whenever a class needs an import, it will append it to this String.
- At the index 1, we find the declaration of the class.
- At the index 2, we find the *main* function.
- At the index 3, we find all the necessary functions (called *actions* in the XML file) as well as the declarations of all the global variables.
- At the index 4, we find the closing curly bracket of the class.

Here is an overview of how the program acts :

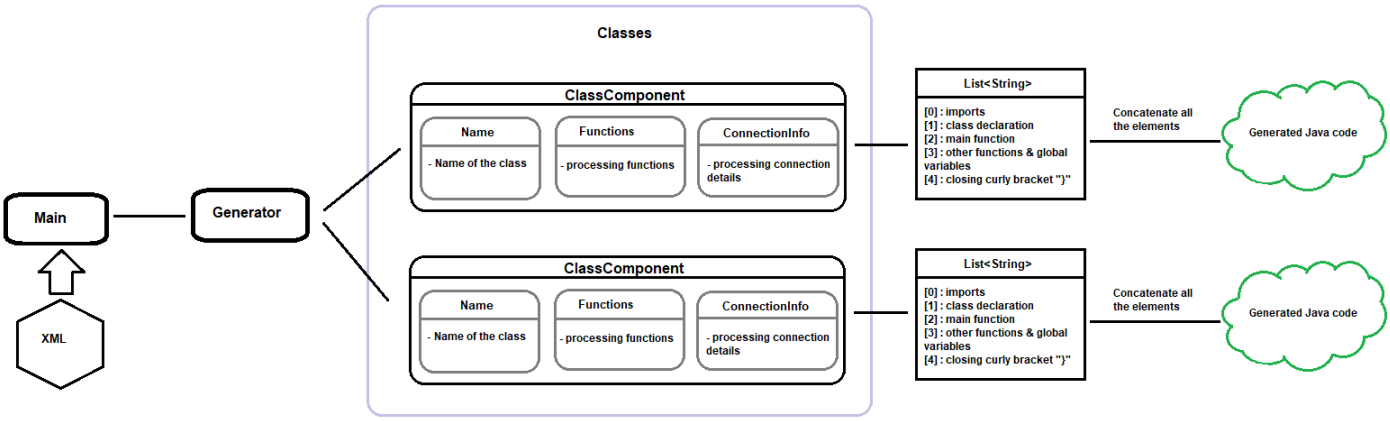


Figure 3.6: Overview of the program's logic

The component-oriented workflow allows for an easier understanding of the program as well as an easier maintenance and perspectives of evolution. Each component contains already written portions of code that will be extracted and imported into the result List when the matching tag or attribute is found. The *FunctionDefinition* class acts as a kind of database containing already written basic functions. Those functions can be accessed using an identifier like A100 or A101 for example (at the moment, those identifiers represent the methods that pull and hire the motors of the garage door). They are stored in a HashMap<String,FunctionDefinition>. Adding new functions or components is just a matter of creating a new class and defining the required behaviour. This means that the program could be considerably extended over time, giving more and more possibilities as we feed it new components.

When all the tags from the XML file have been processed, each element of the List<String> containing the result is concatenated in a single String. That String contains the final code we want. At this point, we are only printing the result in the console, but we could just as well write to a .java file for example.

Chapter 4

Implementation and experimentation

4.1 Architecture of the transformations process

Now that we have presented each part of the process individually, we will present how everything works together to produce the final code. Below is a diagram explaining the entire process, from the creation of the UML model to the result. We will then go through every step one by one.

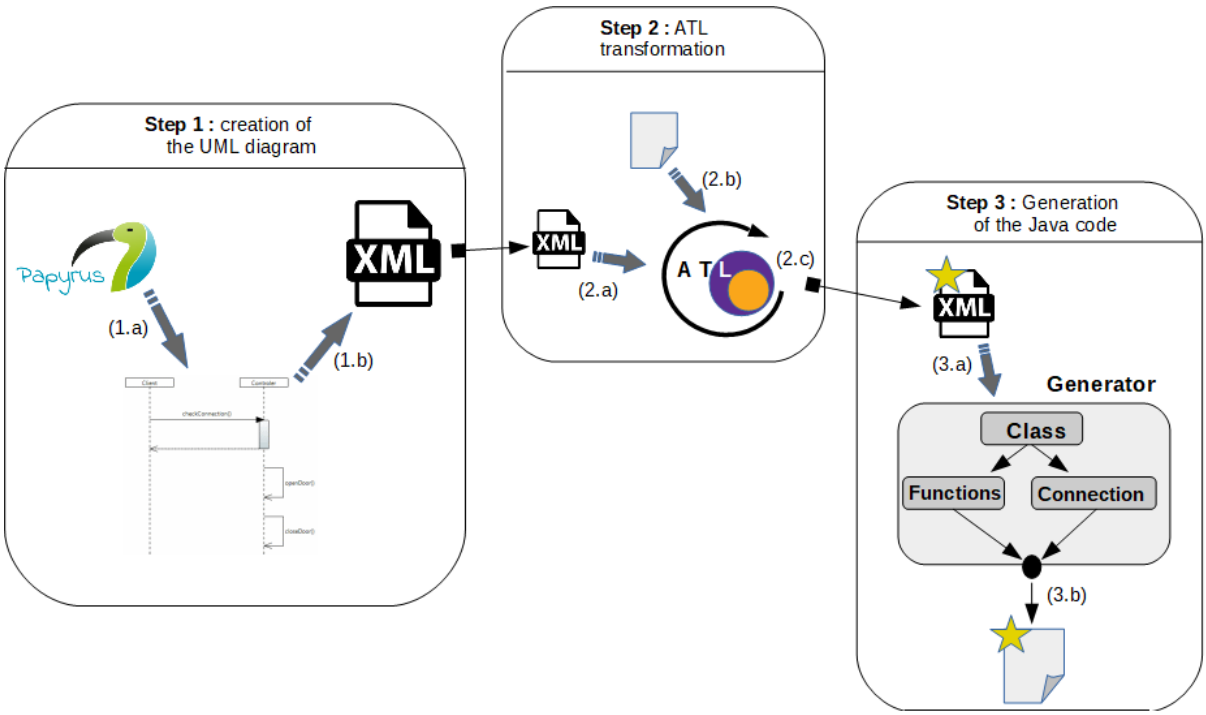


Figure 4.1: Architecture of the process

- **Step 1 : Creation of the UML diagram**
  - **1.a :** The first step is to create the source file for the ATL transformation. For this we use the Eclipse Papyrus plugin, a graphical editor for UML. We create a simple sequence diagram corresponding to our experimentation with the EV3 controller. This diagram represents the possible use cases in our experiment. Here the client checks if the connection is possible with the controller, then the controller can open and close the doors and check if they are open.
  - **1.b :** Once finished, Papyrus generates by itself an .uml file, in the same directory, corresponding to our source file. We recover this file for the next step.

- **Step 2 : ATL Transformation**
  - **2.a :** The source file was generated in the previous part (1.b). In order to use it directly in ATL we replace the root tag *"uml:Model"* by *"xmi:XMI"*, and we add the attribute *"xmlns=SequenceXMI"*. **SequenceXMI** is the name of the metamodel created to describe sequence diagrams, so this attribute allows ATL to understand what we give it as input.
  - **2.b :** We create a metamodel for our source file, and one for our output file. These metamodels are used to describe the models used, they correspond to Ecore files. In this metamodel, each tag in the uml file is seen as a class, each tag nested within it is a reference attribute. As the metamodel describes the model, it will be possible to use any sequence diagram created with Papyrus as source file.
  - **2.c :** Now we have to write the transformation rules. For each element to be created in the output file, we indicate from which element of the source file it is created and how its attributes are filled in.

    Here in our ATL file we create a *<Project>* tag for each *<PackagedElement>* tag in the source file (See Appendix H, rule *"project"*). This tag is unique and contains all the information in the sequence diagram. In the same rule, we call the lazy rule *"lifeline2class"*, which transforms each Lifeline in the sequence diagram into a Class in our output XML. We then fill in all the elements included in each class. For the functions of a class, we use an *"isContained"* helper to find out by which Lifeline the message read in the uml file was sent, and add it to its possible functions. The *<ConnectionInfo>* element of the classes cannot be filled through transformations at this time because the information it contains is not available in the sequence diagram.

    Once the transformation rules have been completed, the various files created in points 2.a and 2.b must be filled in. To do this, right-click on the .atl file, go to **Run As > Run Configuration**, we fill in the location of the input and output metamodels, then the source model. Then just add the name of the output file to be created, and launch the transformation. The resulting XML file is available in Appendix M.

- **Step 3 : Generation of the Java Code**
  - **3.a :** After the previous step, we now have the final XML file that will be fed to the generator. But first, we want to modify the XML file by manually filling some data. Indeed, ATL only yielded the general structure, so we have to provide the missing fields (one possible continuation for this project would be to find a way to eliminate this step or make it as minimal as possible). The complete and detailed information on how to proceed is available below in the section **4.4.3 Manual Configuration** so we will consider that after this step, the XML file is ready (the file is available in Appendix N). At this point, we could consider performing a verification of the syntax of the XML file. One possible way would be to write a *DTD* (Document Type Definition) which we could use to confirm the structure of the file before moving forward with the next step.
  - **3.b :** For this last step, open the main function of the generator in Eclipse (or an other Java IDE). In the declaration of the generator, provide the path to the XML file as the first parameter. The second parameter is not currently used, so it can be left empty (it was supposed to be the path where we want to output the generated code but that functionality has not been implemented). This is an example of how it should look by now : *Generator gen = new Generator("PATH.demo1.xml", "");* The only thing left to do is to run the program.

4.2 Detailed design

In this section, we will detail how the generator works internally. We will describe each class and present a basic API for them (note that the user, meaning the person who wants to generate code, should only have access to the main method and generate the code through the **StartGenerationFromXML();** method of the Generator class). First, here is a diagram representing the overall structure of the program :

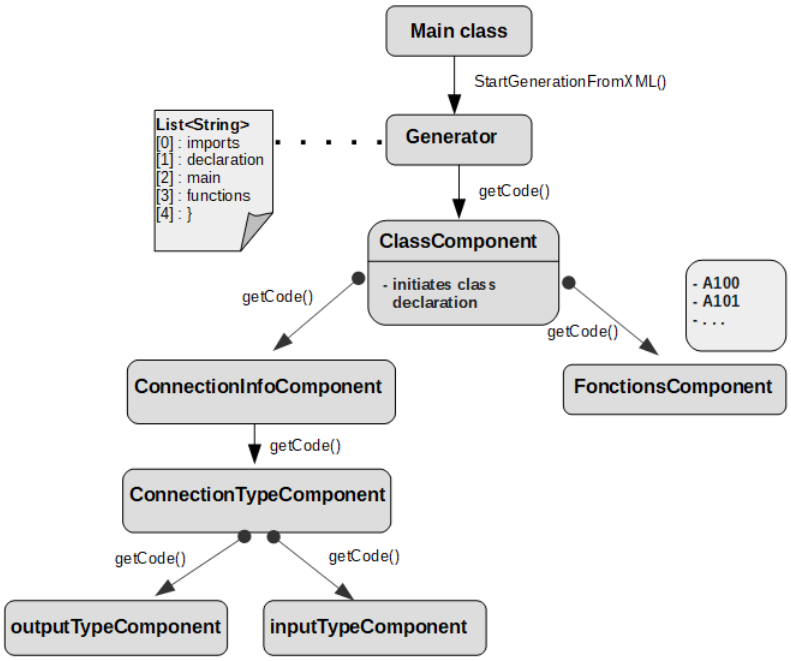


Figure 4.2: Program Design

4.2.1 Main

This is the starting point of the program. The class only contains a main method which declares and initializes the Generator class. As said before, the constructor of the Generator class is waiting for a String representing the path to the XML file. The generation process starts by calling the StartGenerationFromXML() method of the Generator instance created just before. After termination, the result will be printed on the console.

4.2.2 Generator

This is where the actual generation really starts. The class is the last one the end user should be able to use (and only through the aforementioned method). The rest of the generation will unfold on its own through some sort of a *descending* logic : from more general information like the declaration of the class to more detailed aspects as we go. The Generator class is the one storing the List<String> which will be the final result. This list will be passed on to every class along the way so that they can write to it. Using the DOM library of Java, the StartGenerationFromXML() method will parse the XML file obtained through the constructor when the class was instantiated. The first step is to get all the different classes we want to generate. We use are knowledge of the structure of the XML file to get them : they are the children of the root tag <Projet>. The method will then loop through them and process them sequentially as shown in the diagram above (which only shows the process for one class). For each class, we create a ClassComponent, passing it the content of the child *class* tag and calling the getCode() method with the List as a parameter. The getCode() method is the only entry point from one level to the other (see diagram above).

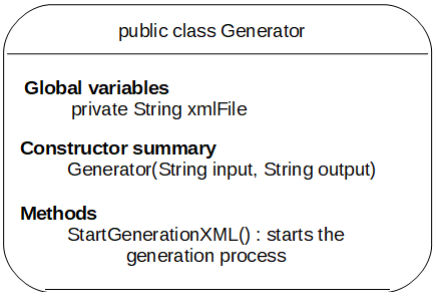


Figure 4.3: Generator API

4.2.3 ClassComponent

This class is the first one to write code in the List : it starts by declaring the class using the *name* attribute of the *class* tag in the XML. It then proceeds to locate two specific tags : ConnectionInfo and Fonctions. If the ConnectionInfo tag is present, a ConnectionInfoComponent is generated and its getCode() method is called. The same principle applies if the Fonctions() tag is present. Note that those tags are not necessarily present in the XML. The generator could work on a standalone application (no remote communication) or a specific class could have absolutely no functions.

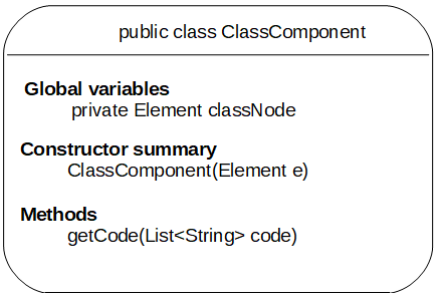


Figure 4.4: ClassComponent API

4.2.4 ConnectionInfoComponent

This class and the ones *descending* from it are the most complicated and probably need the most reworking. As the name suggests, this class manages the remote communication aspect. The first thing we need to do is to find out which type of communication is needed : for now we only manage wifi and USB which are identical from a programming point of view (as we will see in the next sections) but Bluetooth could be added as well. To generate the code for remote communication, we instantiate the ConnectionTypeComponent and call its getCode() method, as always.

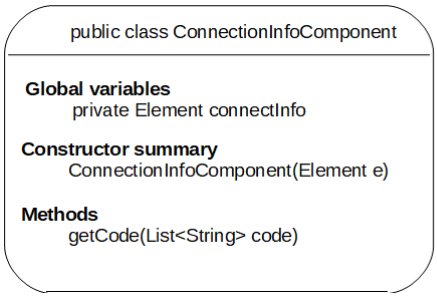


Figure 4.5: ConnectionInfoComponent API

4.2.5 ConnectionTypeComponent

For now, the class only manages wifi and USB connections. Adding Bluetooth is just a matter of extending the *if(type.equals[...])* with an *else* part (and knowing what to put in there obviously). Now that we know the communication type, we need to know the role : client or server. Again, this information is provided within the XML as an attribute of the ConnectionInfo tag. So we use the *getAttribute()* method of the DOM library to recover this data. Depending on the role, we will generate the server side or the client side code (we use Java Socket for TCP/IP communication). The next part is about determining what kind of messages will be exchanged between the server and the client(s). We defined two directions for the messages : from the class, which will be the **output** or to the class, which will be the **input**. The behaviour associated with these two types are managed by the OutputTypeComponent and the InputTypeComponent.

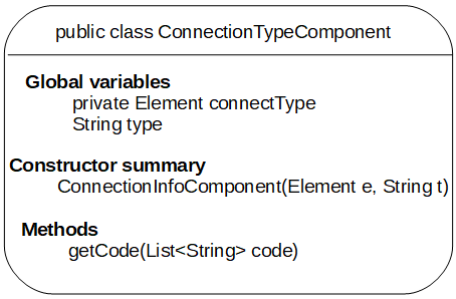


Figure 4.6: ConnectionTypeComponent API

4.2.6 OutputTypeComponent

As said before, this class manages the sending part of remote communication. For now, we only consider text messages (Strings). This class writes the code necessary for listening for user keyboard inputs and stores them in a variable. Adding the possibility for other data types is just a matter of (again) extending the *if* statement.

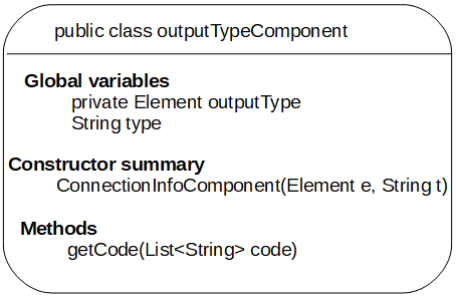


Figure 4.7: OutputTypeComponent API

4.2.7 InputTypeComponent

Again, the class only manages text inputs. It is responsible for receiving the messages sent over the network and knowing what to do with them through the InputMapper tag. The InputMapper is not automatically generated by our ATL transformation, so for the time being (we believe it could be automated), this information has to be written manually, either through the XML file or directly in the final generated code. The InputMapper contains Mappers, which have a guard and an action to perform if the guard is verified. The behaviour is not very flexible for now, but we believe it would be possible to improve on that part. The guard can have a value and a function returning a Boolean. If the guard is verified, it will call the function specified in the actionReturn part of the XML file.

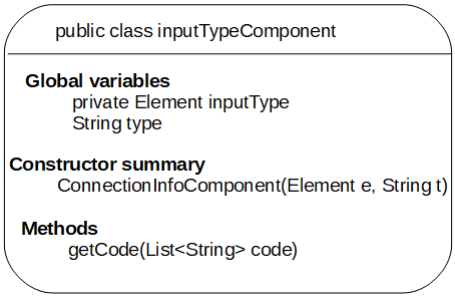


Figure 4.8: InputTypeComponentAPI

4.2.8 FonctionsComponent

This class is responsible for handling function generation. Its constructor initializes all the pre-written functions in a HashMap. The key is the identifier specified in the XML file (A100, A101, etc) and the value is the class containing the code we want to generate. Those classes all inherit from an abstract class so that they can be stored as the same data type : FonctionDefiniton. This allows them to inherit the getCode() method which will be called for generating the code they contain. **Important :** if you want to add a new class (function), you will have to create a Java class that inherits the abstract class FonctionDefinition and most importantly you will have to add that class in the HashMap. Otherwise it will not be recognized. There are ways to automate this (or completely change the way it works), but we did not look into it.

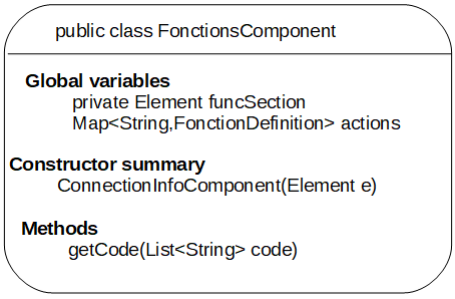


Figure 4.9: FonctionsComponent API

4.3 Running Example

4.3.1 Input data

We will use the Eclipse Papyrus plugin to generate our source file. So the first step is to install it, go to **Help** > **Eclipse Marketplace...** and search for Papyrus.

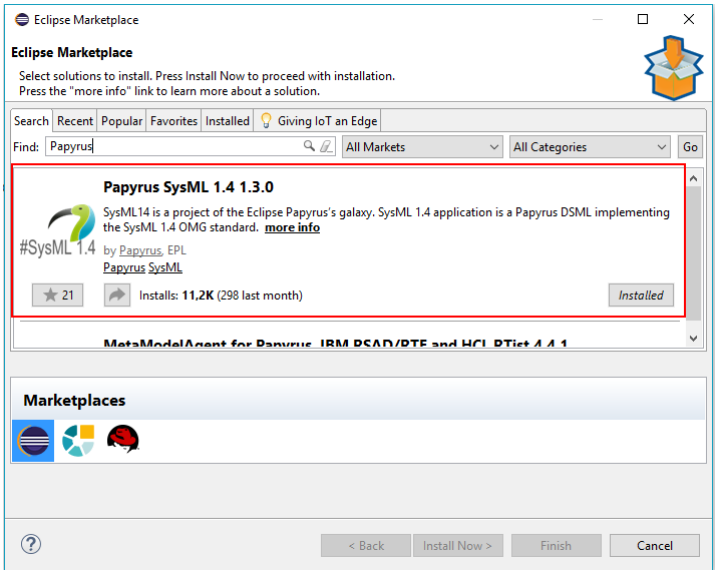


Figure 4.10: Papyrus Installation

To create a new Papyrus project, go to **File** > **New** > **Other...** Then in Papyrus, choose *Papyrus Project* and click Next, choose *UML* and click Next. Enter the name of your project and click Next, then choose *Sequence Diagram* and click Finish.

In the .di file we create a sequence diagram corresponding to our subject of experimentation, here to make it as simple as possible there is only one client and one EV3 controller controlling the doors.

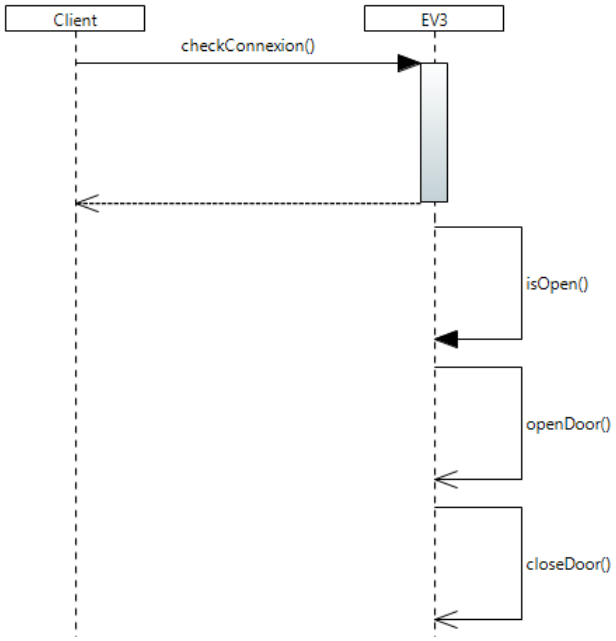


Figure 4.11: Corresponding sequence diagram

Once the diagram is finished, just retrieve the .uml file generated in the same directory by Papyrus, it is available in appendix I.

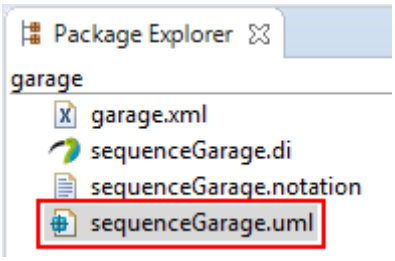


Figure 4.12: .uml file in the project directory

The diagram thus generated is available in appendix I, to be able to use it in ATL we modify the root element *<uml:Model>* and replace it by *<xmi:XMI>*, we also add an attribute *"xlmns=SequenceXMI"*. This file corresponds to the source file used in ATL, it is available in appendix J.

4.3.2 Transformations

To create an ATL project, you must first install ATL on Eclipse. Go to Help > Install New Software, and search for ATL. Once installed, go to File > New > Other... And in ATL select ATL Project.

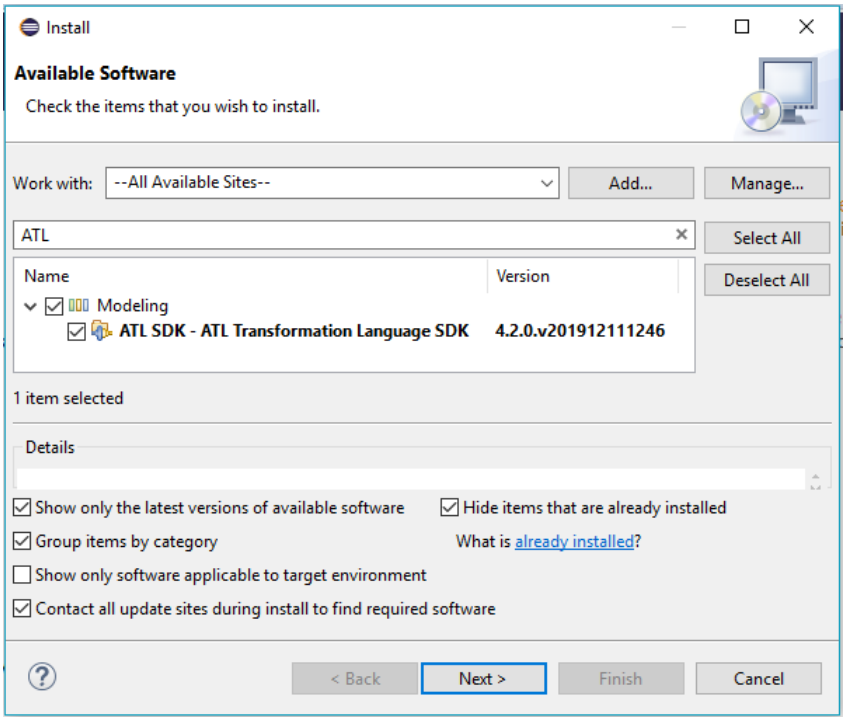


Figure 4.13: ATL installation

To create an ATL file, go to **File>New>Other...**, and choose **ATL File**, this file will contain all the transformation rules, it is available in appendix H. For metamodels, we use ecore files, to create the Ecore file, go to **File > New > Other...**, and then select **Eclipse Modeling Framework > Ecore Model**, they are available in Appendices K and L.

Before starting the ATL transformation, the metamodels used, the source file used and the desired output file must be filled in. To do this, right click on the ATL file, go to **Run As > Run Configuration...**, then fill in the requested information.

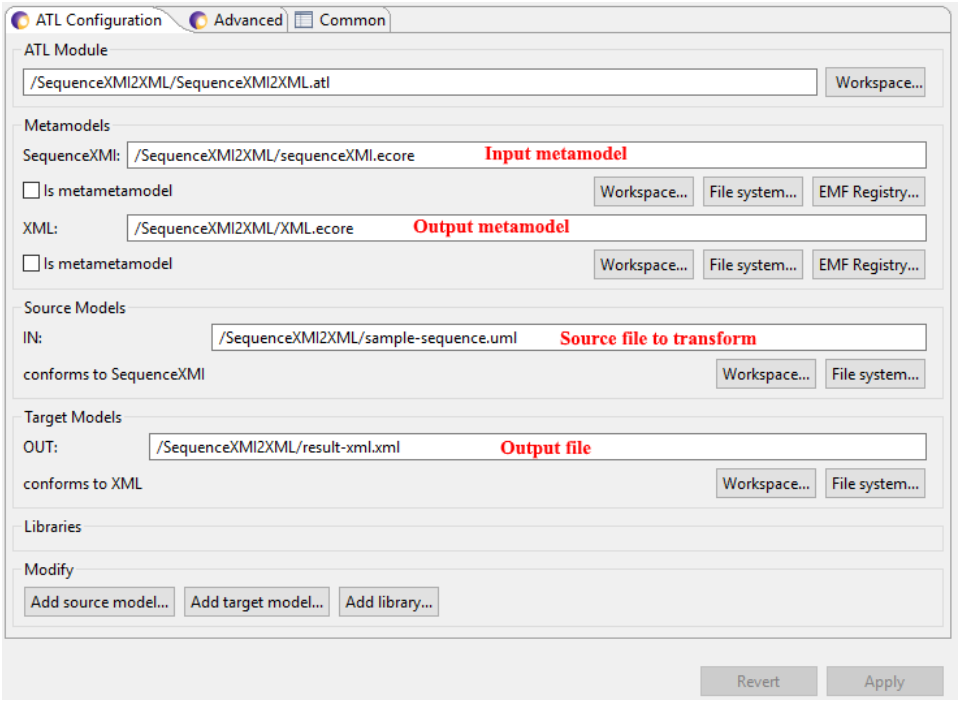


Figure 4.14: ATL run configuration

We can finally launch the transformation, and retrieve the XML file needed to generate java code.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<Projet xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="XML">
  <Class name="Client" hardware="">
    <ConnectionInfo type="" inputType="" role="" ip="" port="" outputType=""/>
    <Fonctions>
      <fonction name="checkConnexion()" actionId=""/>
    </Fonctions>
  </Class>
  <Class name="EV3" hardware="">
    <ConnectionInfo type="" inputType="" role="" ip="" port="" outputType=""/>
    <Fonctions>
      <fonction name="isOpen()" actionId=""/>
      <fonction name="openDoor()" actionId=""/>
      <fonction name="closeDoor()" actionId=""/>
    </Fonctions>
  </Class>
</Projet>
```

Figure 4.15: XML generated file (appendix M)

4.3.3 Manual Configuration

As we said before, the XML file generated by ATL is missing some information that we have to provide manually. To do this in an organized fashion and fully understand the process, we need to be familiar with the structure of the XML file. Here is an extract of the DTD with a focus on the parts that need to be provided. The full DTD is available in Appendix Q.

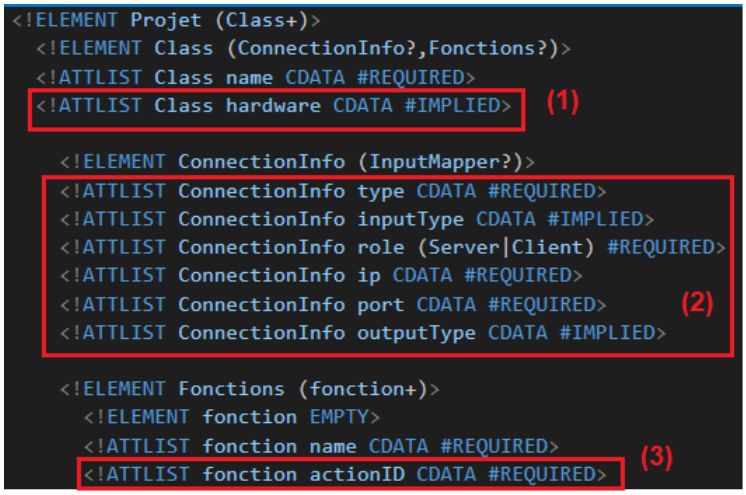


Figure 4.16: Extract of the DTD with a focus on the missing values

Now that we understand the structure, we have to provide the framed part **for each class**. There are three main sections to look at :

- **(1)** : The first part concerns the hardware field. That information is not currently used by the generator, but it might become necessary for future developments. For consistency, enter either PC for the client application (which will run on Eclipse) or EV3 for the server application (which will run on the EV3).
- **(2)** : In this part, we take care of the data required for the remote communication. All the fields are attributes of the ConnectionInfo component. The first is *type* : it wants to know what kind of communication protocol will be used, either Wifi or USB for now as Bluetooth is not supported yet. Note that it is case-sensitive (this inconvenience could be arranged by modifying the generator). The second attribute is *inputType* : if the class needs to receive data, keep the attribute and specify which type of data will be received. For the moment, we only support *text*. Otherwise, simply remove the attribute. The third attribute is *role* : will the class act as the client or the server? Enter Client or Server accordingly. The next attributes are *ip* and *port*. As the name suggests, it is the port used for the connection and the Ip address of the server, which will be used by the client during the initialization of the connection. We typically use port number 5555. The Ip address can be found on the LCD screen of the EV3 controller. Finally, the last attribute is *outputType* : if the class needs to send data, then keep the attribute and provide *text*. Otherwise, simply remove the attribute.
- **(3)** : In this last part, we have to provide the function identifiers corresponding to the functions we want to have generated. For now, we only have A100, A101, A110 and A111. They are used respectively for opening the doors, closing the doors, using the sensor to check distances and checking if the user entered the String *connect*. Of course, they could be used for other things, with minor modifications of the code.

There is one last part to take care of and it is also the more tedious : we have to tell the server class what to do with the input received over the network. This is where the InputMapper tag comes into play : it is composed of Mapper tags, each of which having a *guard* and an *actionReturn* tag. The guard has a value tag specifying what String is expected, as well as an action tag specifying a boolean method (through the *name* attribute) and the expected return value (through the *expect* attribute). As none of this section is generated by ATL, we have to add the whole InputMapper data manually in the XML file. **Important :** the generator expects the InputMapper tag to be a child of the ConnectionInfo tag, so it is necessay to modify that tag accordingly (basically, don't use the shortcut for closing a tag and make a *</ConnectionInfo>*). The XML files before and after manual completion are available in Appendix M and Appendix N.

4.3.4 Result program

The Java Code generator generates two classes, a client and a server, corresponding to the classes indicated in the source XML file. The classes are simply displayed in the console for the experimentation, but we could imagine a .java file generation. The two classes generated are available in appendices O and P.

The client connects to the server using sockets, where information such as IP address or port has been specified in the XML file. Some information remains to be filled in, such as the use of the *"checkConnection()"* function, and the text or keyword corresponding to the connection termination condition.

In the server class, the text or keyword corresponding to the connection termination must also be filled in.

4.3.5 Execution

To test our generation, we import the server code on the EV3 controller, and connect the controller via USB to the computer (a wifi connection could also be possible). Then we run the server program on the EV3, and the Client on Eclipse which will serve as a remote control for our doors.

Once the connection is established, you can open (keyword "1" on the client) and close (keyword "2") the doors, and then close the connection. The code generation is functional. [reprendre ici la fin de la démo utube]

4.4 Summary of the experimentations

After identifying the important automatable and non-automatizable parts of the java code, we set up an XML file containing the important information, while ensuring that the addition of the non-automatizable parts remains simple and understandable. We managed to set up an ATL transformation to arrive at this transition XML file with a UML sequence diagram as a starting file, and we developed a parser to generate the Java code using this XML.

If this suite of transformations is theoretically functional, it may be possible to achieve a better XML model for the java parser, using other transformations and adding more information to reduce manual additions. In addition, the process currently works for UML sequence diagrams, it may be interesting to investigate whether it is possible to make universal transformations that work for all types of UML diagrams.

Chapter 5

Results and perspectives

In the previous chapter, we presented a complete transformation from a simple UML model to the final executable code. However, the work is far from complete. The goal was to identify the different steps of refinement while having a functional system to rely on. It is not meant to be the foundations of the complete project but rather to maybe give an insight on what has to be done. Keeping this fact in mind, we will now present the advantages and problems of our method. Finally, we will try to paint a general picture of the different refinement steps we have identified.

5.1 Discussing our method

As we said in the previous chapter, the program we wrote can be extended to allow for more possibilities. However, adding new components or tags/attributes will imply that the meta-models and rules we defined for ATL will have to be updated. As the complexity grows, so will the management of the ATL transformations. Unless a purely compartmentalized solution is found, this can become a tedious task. That being said, we can think of our method as unstable at the moment, but it would probably reach a point in time when no more heavy changes are necessary, therefore reaching its "mature", stable form. Moreover, we believe that this tedious "*growing process*" is inevitable, whichever solution is used in the future.

There is also a problem of flexibility with our method. The List<String> allows easy text insertion at the end and the beginning of the String, but there might come a time when one will need to insert text in between specific lines. At this point, an other data structure should be considered, like a tree for example. Or the parser could be abandoned entirely. Indeed, the final transformation could be realized through ATL. We used a custom java program to better understand the problematic and start faster, be this is in no way the only solution (and most likely not the best).

5.2 Refinement of UML models

We will now present the refinement steps that we have identified throughout the project. First of all, and we didn't explicitly mentioned it until now, everything can't be fully automated. Or rather it would go against the primary goal of this project which is the simplification of the development and maintenance process of a software using model driven engineering (especially distributed systems). Indeed, some data require user input, like the IP address or the port number in the case of TCP/Ip communication for example. We could feed it somehow into the model itself, but we think it is much faster to just provide that information manually, either in the final model, or in the produced code. However, there is some data that must be provided during the refinement process, decisions must be made, and this brings us to the presentation of the main steps. It is important to note that we only worked with sequence diagrams. The use of other complementary UML models could provide the data needed with minimal user input :

- **Identification of the environment** : the idea is to define which classes work together on the same hardware (meaning they don't communicate remotely, we will call them a *cluster*), which class is the main class, and which class will manage the remote communications (if any), as well

as its role : server or client. This is an important step because we need to know where to create the *main* function, the global variables referencing the other classes as well as the *ConnectionInfo* component.

- **Specification of the environment** : Now that we have defined the overall structure of the clusters, we can add data such as the protocols used for remote communication, the type of messages sent over the network (will it be text, integers, bytes, etc) and what do those messages map to. In terms of code, this step will mostly populate the *main* functions and the *InputMapper* tag of the final XML file.
- **General tasks** : by this point, most of the communication aspects have been dealt with. This step will mainly create the functions themselves.

This is how we conducted the "removing" process from the backward perspective : going from the last point to the first, the information is more and more abstract.

Conclusion

The automation of code generation from UML models is a complex problem. There are many factors to consider and some of them fight the purpose of the others. If the solution is flexible and as complete as possible, the complexity of the refinement process is increased, as well as the number of inputs and decisions the user has to make during the transformation process. Spending that much effort (from the user point of view) into the refinement process might go against the primary goal which is speed and simplicity. On the other hand, if the solution is too general and rigid (say we only generate the skeleton of the classes, methods and some very basic code) then the user will still have a lot of programming to do. The complexity stems from the balancing act we have to do, finding the "sweet spot" between these two extreme cases. The work that we have presented in this paper is far from complete. We have chosen to take a step in a direction that we don't know where it leads to, or if it is even worth pursuing as is. Nevertheless, we have tried to make our solution fully reproducible (through this paper and our Git) so that future teams will be able, we hope, to start working faster and from a higher point of view.

Bibliography

[1] ATL User Guide, The ATL Language
https://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language

[2] ATL User Guide, Tutorial ATL Transformation
https://wiki.eclipse.org/ATL/Tutorials_-_Create_a_simple_ATL_transformation

[3] Document Java - Interface Document https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/Document.html

[4] LeJos EV3 - Documentation
http://www.lejos.org/ev3/docs/

[5] Wikimedia Commons - ATL Context
https://commons.wikimedia.org/w/index.php?curid=4307249

Appendix A

Glossary

- **ATL** : ATLAS Transformation Language, a model transformation language available as an Eclipse plugin
- **LS2N** : Laboratoire des Sciences et du Numérique de Nantes
- **EV3** : the hardware used for controlling the robot
- **EMF** : Eclipse Modeling Framework
- **JRE** : Java Runtime Environment
- **Ncm** : Newton centimeter
- **OOP** : Object-oriented programming
- **SD**: Secure digital
- **UML** : Unified Modeling Language
- **XML** : Extensible Markup Language

Appendix B

Technical appendix

B.1 LEJOS Environment Installation tutorial

The first step is to install the Lejos firmware on the EV3 box. To do this, you will need an SD card between 2 and 32 GB. Preferably it should be empty and in FAT32 format. Then follow the tutorial (really, be careful even if you think you have understood... don't install Eclipse either, there are limits) on this video: https://www.youtube.com/watch?v=ycOA4wnvAyM You should now have the EV3 box that boots on LEJOS instead of mindstorm and the Eclipse plugin installed. If that's the case, bravo, that's the end of this tutorial, go directly to the "END" section. Otherwise, maybe the following will help you:

Problem to install LEJOS on the box:

We also had some problems at first, and we tested with another SD card, and everything worked after that. I can only advise you to follow the video tutorial to the letter (if it still doesn't work, check out the next section on the ECLIPSE PLUGIN).

Problem to install the LEJOS plugin on Eclipse:

If you have problems with this step, again, follow the tutorial to the letter, FROM THE START. When you have installed the plugin via the menu Eclipse Help => Install New SoftWare, you have to go to the menu: Window => Preferences => Lejos EV3 Look at field EV3_HOME. If it is empty, indicate the path to the folder where you installed LEJOS EV3 (following the tutorial).

END

Normally, you should now have the box that boots on the LEJOS firmware and the LEJOS plugin on Eclipse. To create a LEJOS project on Eclipse do File => New => Project => Lejos EV3 => Lejos EV3 Project (If a "Build path problem" error occurs, you have not correctly linked EV3 _HOME as seen previously). Go to Windows => Preference => LEjos EV3 and check Connect To names Brick and enter the address 10.0.1.1 in the field. Validate. That's it, you're ready for the next step!

B.2 Test code

```
package org.testings;
import lejos.hardware.motor.Motor;
import lejos.hardware.port.SensorPort;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.SampleProvider;
```

```
import lejos.utility.Delay;
public class Test1{

        public static void main(String[] args) {
                EV3UltrasonicSensor uSensor = new
                    EV3UltrasonicSensor(SensorPort.S2);
                SampleProvider sampleProvider = uSensor.getDistanceMode();
                float[] sample = new float[sampleProvider.sampleSize()];
                sampleProvider.fetchSample(sample, 0);
                System.out.println(sample[0]);
                while(true){
                        if(sample[0]>8){
                                System.out.println("OUVERT");
                                break;
                        }else{
                                System.out.println("FERME");

                        }
                        sampleProvider = uSensor.getDistanceMode();
                        sample = new float[sampleProvider.sampleSize()];
                        sampleProvider.fetchSample(sample, 0);
                        Delay.msDelay(1000);
                }
                Delay.msDelay(20000);
                Motor.A.setSpeed(90);
                Motor.D.setSpeed(90);
                Motor.A.forward();
                Motor.D.backward();
                Motor.A.rotateTo(-90);
                Motor.D.stop();
                Motor.A.stop();

        }

}
```

B.3 Tutorial for connecting the EV3 controller to the PC via USB

This tutorial assumes that you have correctly installed the Eclipse plugin, that you have created an Eclipse LEJOS project, and that you have created an Eclipse LEJOS project. and that your controller boots on the LEJOS firmware. If it is not the case, see tutorial "Installation LEJOS firmware + EClipse plugin".

Take your Ev3 box and the USB cable provided in the LEGO box. **CONNECT YOUR EV3 to your PC during this tutorial**.

This tutorial also assumes that you operate under Windows 10, if this is not the case, too bad!

First of all, you have to know that when you connect your Ev3 box to your PC, Windows 10 seems to recognize it and install drivers, but it's not true... Completely wrong! You will have to install a particular driver by yourself : the RNDIS !!. To do this, first read the following "To read" lines, you can download the DRIVER from the DRIVER folder on the git ter-ir-2020/Transfo-protocoles/DRIVER if you have access to it.

To read: Create a new folder "InsertRelevantNameForU" at the root of your hard drive. Extract the files from the DRIVER folder in the Git to this folder. Right-click the folder. on the file with the .inf extension and select "Install". Be aware of possible popups. Is it done? Okay, so now follow this:

- Windows + R
- devmgmt.msc
- Expand "Network card"
  - you should see something like "USB Ethernet/RNDIS Gadget."
  - Right click on it
  - Properties => Advanced
  - Select "Value" then enter in the field "10.0.1.1" and validate all that.

Bravo, it's over! Now it's time to test the connection:

- Windows + R
- cmd
- ping 10.0.1.1

You should have an answer with 0% loss.

If this is the case, you have correctly connected your box to your PC via USB. Now you just have to upload a program on this Ev3 box.

I assumed at the beginning of this article that you already had a properly setup Eclipse project in Lejos project mode. If this is not the case, see the tutorial "LEJOS firmware installation + EClipse plugin". Got it? Now create a class by checking the "public static void main" checkbox. Write something simple as LCD.drawsString("HELLO EV3",0,0) (Import what ecplise tells you, i.e. import lejos.hardware.lcd.LCD; in this case). Now we are ready to import this nice program on the box. To do this, right click on your class => Run As ==> lejos EV3 Program.

Your controller should ask you to wait a second. Then you should see "HELLO EV3" appear on the screen of the box.

Don't panic if you don't!It didn't work for me the first time either...

Your controller makes a nice noise with a nice mistake that doesn't make sense to you? Let's dive into the wonderful world of JRE! Because there lies the problem. (Press the back button on the box to exit the error screen)

I hope for your sake that you have version 1.7 of JRE java. Otherwise you will have to install it. Right click on your LEJOS project. Select Properties => java compiler

Then change the JRE to version 1.7, and validate. Now try to upload the program again.

If it works, bravo, you have correctly linked your case to your PC! If not, follow the steps in order and good luck!

B.4 Source code to run on the EV3 controler for USB and WIFI connection

```
package org.client;

import java.io.BufferedReader;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

import lejos.hardware.lcd.LCD;
import lejos.hardware.motor.Motor;
import lejos.hardware.port.SensorPort;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.SampleProvider;


public class TestUSBcontrol {
        private static EV3UltrasonicSensor uSensor;

        public static boolean estOuvert(){
                SampleProvider sampleProvider = uSensor.getDistanceMode();
                float[] sample = new float[sampleProvider.sampleSize()];
                sampleProvider.fetchSample(sample, 0);
                if(sample[0]>8){
                        return true;
                }
                return false;
        }

        public static void main(String[] args) {
                String clientRequest;
                String serverResponse;
                uSensor = new EV3UltrasonicSensor(SensorPort.S2);
                try(ServerSocket socket = new ServerSocket(5555))
                {
                        while(true)
                        {
                                LCD.drawString("Waiting", 0, 0);
                                Socket connectionSocket = socket.accept();
                                LCD.clear();
                                LCD.drawString("Connected client", 0, 0);
                                BufferedReader reader = new
                                   BufferedReader(new
                                   InputStreamReader(connectionSocket.getInputStream())

                                OutputStream output =
                                   connectionSocket.getOutputStream();
                                PrintWriter writer = new
                                   PrintWriter(output,true);
                                String text;

                                do {
                                        text = reader.readLine();
                                        if(text.equals("1") &&
                                           !estOuvert())
                                        {
                                                // commenter ces deux
                                                   lignes
                                                LCD.clear();
```

```
                                        OpenDoor();
                                }
                                else if(text.equals("2") &&
                                   estOuvert())
                                {
                                        //commenter ces deux lignes
                                        LCD.clear();
                                        CloseDoor();
                                }

                        }while(!text.equals("leave"));

                        socket.close();
                }
        }
        catch(IOException e1)
        {
                e1.printStackTrace();
        }
    }
    public static void OpenDoor()
    {
            Motor.A.setSpeed(90);
            Motor.D.setSpeed(90);
            Motor.A.forward();
            Motor.D.backward();
            Motor.A.rotateTo(-90);
            Motor.D.stop();
            Motor.A.stop();
    }

    public static void CloseDoor()
    {
            Motor.A.setSpeed(90);
            Motor.D.setSpeed(90);
            Motor.A.forward();
            Motor.D.forward();
            Motor.A.rotateTo(0);
            Motor.D.stop();
            Motor.A.stop();
    }
}
```

B.5 Control source code to be launched on the PC for a USB connection

```
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;
import java.io.OutputStream;
import java.io.PrintWriter;

public class ClientUSB {
```

```
        public static void main(String[] args) {
                try(Socket socket = new Socket("10.0.1.1",5555))
                {
                        OutputStream output = socket.getOutputStream();
                        PrintWriter writer = new PrintWriter(output,true);
                        Scanner sc = new Scanner(System.in);
                        String text;

                        do
                        {
                                text = sc.nextLine();
                                writer.println(text);
                        }while(!text.equals("leave"));

                        socket.close();
                }
                catch(UnknownHostException e)
                {
                        System.out.println("Server not found");
                }
                catch(IOException e)
                {
                        System.out.println("I/O error");
                }
        }
}
```

B.6 Tutorial for connecting the EV3 controller to the PC via WIFI

This tutorial also assumes that you have correctly installed the Lejos EV3 plugin on Eclipse, and that you have created a LEJOS Project and that your box boots on the LEJOS firmware. If this is not the case, see tutorial "LEJOS firmware installation + EClipse plugin".

First of all, you only need your Ev3 controller and your PC and please follow the following steps:

STEP 1) Plug the Wifi adapter into the USB Host port of the EV3 brick.

STEP 2) Turn on the EV3 brick.

STEP 3) - Pull down the menu "tools"

- select "Wifi" ;
- tick the "Wifi" box ;
- select "Connections" ;
- select the Wifi access point from the list, then wait ;
- Enter the WEP/WPA key if necessary;
- select "Connect" ;
- scroll down the "tools" menu;
- select "Brick info" ;

- keep the IP address of the EV3 brick (e.g. 192.168.1.2) for later use.

STEP 4) We need to establish a 4G connection sharing between our Android, PC and the controller so that all three belong to the same network. A new IP address will appear on the screen of the Lejos brick controller.

STEP 5) Define the Lejos controller as a server rather than a client by scrolling and clicking on the Wifi icon then on Access point pt+ and enter the IP address of your brick and validate.

STEP 6) In the command line of your PC run the instruction: telnet followed by the new IP address displayed on the brick and enter as login 'root' and without password.

STEP 6bis) If the Telnet command is not recognized: C:>telnet google.com 80

'Telnet' is not recognized as an internal or external command, operable program or batch file.

Cause: The Telnet client in a Windows Operating System is disabled by default.

Command-Line Solution : Start the following command with authorizations for Administrator level:

dism /online /Enable-Feature/FeatureName:TelnetClient

Manual configuration solution : Start by searching and click on 'Turn Windows features on or off' from the search area near the Start button. Then on the page that appears search and select Telnet Client and finally click Ok.

You will then have the 'Apply Changes' loading page in the title. Once the changes are done, you can finally click on 'Close'.

STEP 7)root@EV3: # dropbear

root@EV3: # hcitool dev

Devices:

hci0 11:22:33:44:55:

STEP 8) Run the following class: 'TestUSBControl'

STEP 9) Wait for 'Waiting' to be displayed on the controller screen, which indicates waiting for a connection.

STEP 10) Run the following Class: 'Client WIFI'

STEP 11) To Open the portal enter 1 on your Console and to close it enter 2 and to end the connection enter Leave.

B.7 Control source code to be launched on the PC for a WIFI connection

```
package org.client;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;
import java.io.OutputStream;
```

```
import java.io.PrintWriter;

public class ClientWIFI {

        private static Scanner sc;

        public static void checkConnection(){
                String text ="";
                sc = new Scanner(System.in);
                do
                {
                        text = sc.nextLine();
                }while(!text.equals("connect"));

        }

        public static void main(String[] args) {
                checkConnection();
                try(Socket socket = new Socket("192.168.43.37",5555))
                {
                        OutputStream output = socket.getOutputStream();
                        PrintWriter writer = new PrintWriter(output,true);
                        String text;

                        do
                        {
                                text = sc.nextLine();
                                writer.println(text);
                        }while(!text.equals("leave"));

                        socket.close();
                }
                catch(UnknownHostException e)
                {
                        System.out.println("Server not found");
                }
                catch(IOException e)
                {
                        System.out.println("I/O error");
                }
        }
}
```

Appendix C

Technical appendix (in french)

C.1 Annexe A

Tutoriel d'installation du LEJOS firmware et de LeJOS plugin sur eclipse

Le premier pas consiste à installer le Lejos firmware sur le boitier EV3. Pour cela, munissez-vous d'une carte SD entre 2 et 32 Go. De préférence, elle doit être vide et au format .FAT32. Suivez ensuite le tutoriel (vraiment, soyez attentif même si vous pensez avoir compris.. ne résinstallez pas Eclipse non plus, ya des limites) sur cette vidéo : https://www.youtube.com/watch?v=ycOA4wnvAyM Vous devriez maintenant avoir le boitier EV3 qui boot sur LEJOS au lieu de mindstorm et le plugin Eclipse installé. Si c'est le cas, bravo, c'est la fin de ce tutoriel, passer directement à la section "FIN". Sinon, peut-être la suite pourra vous aider:

Problème pour installer LEJOS sur le boitier:

Nous avons aussi eu des problème au début, et nous avons testé avec une autre carte SD, et tout fonctionnait après ça. Je ne peux que vous conseiller de suivre le tutoriel vidéo à la lettre (si ça ne fonctionne toujours pas, regardez la section suivante sur le PLUGIN ECLIPSE).

Problème pour installer le plugin LEJOS sur Eclipse:

Si vous avez des problèmes avec cette étape, encore une fois, suivez le tutoriel à la lettre, DEPUIS LE DÉBUT. Lorsque vous avez installé le plugin via le menu Eclipse Help => Install New SoftWare, il faut absolument aller voir dans le menu : Window => Preferences => Lejos EV3 Regardez le champ EV3_HOME. S'il est vide, indiquez le chemin du dossier où vous avez installer LEJOS EV3 (en suivant le tutoriel).

FIN

Normalement, vous devriez maintenant avoir le boitier qui boot sur le firmware LEJOS et le plugin LEJOS sur Eclipse. Pour créer un projet LEJOS sur Eclipse, faites File => New => Projet => Lejos EV3 => Lejos EV3 Project (Si une erreur du style "Build path problem" arrive, vous n'avez pas correctement lié EV3_HOME comme vu précédemment.) Allez dans Windows => Preference => LEjos EV3 et cochez Connect To names Brick et entrez dans le champ l'addresse 10.0.1.1. Validez. C'est terminé, vous êtes prêt pour la suite!

C.2 Annexe B

Code de Test

```
package org.testings;
import lejos.hardware.motor.Motor;
import lejos.hardware.port.SensorPort;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.SampleProvider;
import lejos.utility.Delay;
public class Test1{

        public static void main(String[] args) {
                EV3UltrasonicSensor uSensor = new
                   EV3UltrasonicSensor(SensorPort.S2);
                SampleProvider sampleProvider = uSensor.getDistanceMode();
                float[] sample = new float[sampleProvider.sampleSize()];
                sampleProvider.fetchSample(sample, 0);
                System.out.println(sample[0]);
                while(true){
                        if(sample[0]>8){
                                System.out.println("OUVERT");
                                break;
                        }else{
                                System.out.println("FERME");

                        }
                        sampleProvider = uSensor.getDistanceMode();
                        sample = new float[sampleProvider.sampleSize()];
                        sampleProvider.fetchSample(sample, 0);
                        Delay.msDelay(1000);
                }
                Delay.msDelay(20000);
                Motor.A.setSpeed(90);
                Motor.D.setSpeed(90);
                Motor.A.forward();
                Motor.D.backward();
                Motor.A.rotateTo(-90);
                Motor.D.stop();
                Motor.A.stop();

        }

}
```

C.3 Annexe C

Tutoriel de connexion du boitier au PC via USB

Ce tuto suppose que vous avez correctement installé le plugin Eclipse, que vous avez créer un Projet Eclipse LEJOS et que votre boitier boot sur le LEJOS firmware. Si ça n'est pas le cas, voir tuto "Installation LEJOS firmware + plugin EClipse".

Munissez vous de votre boitier Ev3 et du cable USB fourni dans la boite LEGO. **BRANCHEZ VOTRE EV3 à votre PC pendant tout ce tuto.**

Ce tuto suppose également que vous opérez sous Windows 10, si ça n'est pas le cas, dommage !

Tout d'abord, il faut savoir que lorsque vous connectez votre boitier Ev3 à votre PC, Windows 10 semble le reconnaitre et installer des drivers, mais c'est faux... Complétement faux! Il va vous falloir installer un driver particulier par vous-même : le RNDIS !!. Pour ce faire, lisez d'abord les lignes "A lire" qui suivent , vous pouvez téléchareger le DRIVER à partir du dossier DRIVER sur le git ter-ir-2020 Transfo-protocolesTransfo-protocoles si vous y avez accès.

A lire : Créez un nouveau dossier "InsertRelevantNameForU" à la racine de votre disque dur. Extrayez-y les fichiers se trouvant dans le dossier DRIVER du Git. Cliquez droit sur le fichier d'extension .inf et selectionnez "Installer". Soyez consentant aux éventuelles popup. C'est fait? Okay, alors suivez maintenant ceci :

- Windows + R
- devmgmt.msc
- Etendez "Carte réseau"
  - vous devriez voir quelque chose du genre "USB Ethernet/RNDIS Gadget"
  - Clique droit dessus
  - Propriétés => Avancé
  - Sélectionnez "Valeur" puis entrez dans le champ "10.0.1.1" et validez moi tout ça

Bravo, c'est terminé ! Maintenant il est temps de tester la connexion :

- Windows + R
- cmd
- ping 10.0.1.1

Vous devriez avoir une réponse avec 0

Si c'est le cas, vous avez correctement connecté votre boitier à votre PC via USB. Reste maintenant à upload un programme sur ce boitier Ev3.

J'ai suppososé au début de cet article que vous aviez déjà un projet Eclipse correctement Setup en mode Lejos project. Si ça n'est pas le cas, voir le tutoriel "Installation LEJOS firmware + plugin EClipse". C'est bon? Créez maintenant une classe en cochant la case "public static void main" . Écrivez quelque chose de simple comme LCD.drawsString("BONJOUR EV3",0,0) (Importez ce qu'ecplise vous indique, soit import lejos.hardware.lcd.LCD; dans ce cas). Maintenant nous sommes fin prêt pour importer ce beau programme sur le boitier. Pour ce faire, cliquez droit sur votre classe => Run As ==> lejos EV3 Program.

Votre boitier devrait vous implorer d'attendre 1 seconde. Puis vous devriez voir apparaitre "BONJOUR EV3" sur l'écran du boitier.

Pas de panique si ça n'est pas le cas ! Moi non plus ça n'a pas fonctionné du premier coup...

Votre boitier fait un joli bruit avec une sympathique erreur qui ne fait pas sens pour vous? Plongeons dans le magnifique monde des JRE ! Car là se trouve le problème. (Appuyez sur le bouton retour sur le boitier pour quitter l'écran d'erreur)

J'espère pour vous que avez la version 1.7 du JRE java. Sinon il va falloir l'installer. Cliquez droit sur votre projet LEJOS. Selectionnez Properties => java compiler

Puis Changer le JRE pour la version 1.7, et validez. Maintenant réessayez d'upload le programme.

Si ça marche, bravo, vous avez correctement lié votre boitier à votre PC ! Sinon, suivez bien les étapes dans l'ordre et bonne chance

C.4 Annexe D

Code source à exécuter sur le boîtier pour une connexion USB et WIFI

```
package org.client;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

import lejos.hardware.lcd.LCD;
import lejos.hardware.motor.Motor;
import lejos.hardware.port.SensorPort;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.SampleProvider;


public class TestUSBcontrol {
        private static EV3UltrasonicSensor uSensor;

        public static boolean estOuvert(){
                SampleProvider sampleProvider = uSensor.getDistanceMode();
                float[] sample = new float[sampleProvider.sampleSize()];
                sampleProvider.fetchSample(sample, 0);
                if(sample[0]>8){
                        return true;
                }
                return false;
        }

        public static void main(String[] args) {
                String clientRequest;
                String serverResponse;
                uSensor = new EV3UltrasonicSensor(SensorPort.S2);
                try(ServerSocket socket = new ServerSocket(5555))
                {
                        while(true)
                        {
                                LCD.drawString("Waiting", 0, 0);
                                Socket connectionSocket = socket.accept();
                                LCD.clear();
                                LCD.drawString("Connected client", 0, 0);
                                BufferedReader reader = new
                                    BufferedReader(new
                                    InputStreamReader(connectionSocket.getInputStream())

                                OutputStream output =
                                    connectionSocket.getOutputStream();
                                PrintWriter writer = new
                                    PrintWriter(output,true);
                                String text;
```

```
                                do {
                                        text = reader.readLine();
                                        if(text.equals("1") &&
                                            !estOuvert())
                                        {
                                                // commenter ces deux
                                                    lignes
                                                LCD.clear();
                                                OpenDoor();
                                        }
                                        else if(text.equals("2") &&
                                            estOuvert())
                                        {
                                                //commenter ces deux lignes
                                                LCD.clear();
                                                CloseDoor();
                                        }

                                }while(!text.equals("leave"));

                                socket.close();
                        }
                }
                catch(IOException e1)
                {
                        e1.printStackTrace();
                }
        }
        public static void OpenDoor()
        {
                Motor.A.setSpeed(90);
                Motor.D.setSpeed(90);
                Motor.A.forward();
                Motor.D.backward();
                Motor.A.rotateTo(-90);
                Motor.D.stop();
                Motor.A.stop();
        }

        public static void CloseDoor()
        {
                Motor.A.setSpeed(90);
                Motor.D.setSpeed(90);
                Motor.A.forward();
                Motor.D.forward();
                Motor.A.rotateTo(0);
                Motor.D.stop();
                Motor.A.stop();
        }
}
```

C.5 Annexe E

Code source de contrôle à lancer sur le PC pour une connexion USB

```
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;
import java.io.OutputStream;
import java.io.PrintWriter;

public class ClientUSB {

        public static void main(String[] args) {
                try(Socket socket = new Socket("10.0.1.1",5555))
                {
                        OutputStream output = socket.getOutputStream();
                        PrintWriter writer = new PrintWriter(output,true);
                        Scanner sc = new Scanner(System.in);
                        String text;

                        do
                        {
                                text = sc.nextLine();
                                writer.println(text);
                        }while(!text.equals("leave"));

                        socket.close();
                }
                catch(UnknownHostException e)
                {
                        System.out.println("Server not found");
                }
                catch(IOException e)
                {
                        System.out.println("I/O error");
                }
        }
}
```

C.6 Annexe F

Tutoriel de connexion du boitier au PC via WIFI

Ce tuto aussi suppose que vous avez correctement installé le plugin Lejos EV3 sur Eclipse, et que vous avez créé un Projet LEJOS et que votre boitier boot sur le LEJOS firmware. Si ça n'est pas le cas, voir tuto "Installation LEJOS firmware + plugin EClipse".

Dans un premier temps, munissez-vous uniquement de votre boitier Ev3 et de votre PC et veuillez suivre les étapes suivantes:

ETAPE 1) Brancher l'adaptateur Wifi dans le port USB Host de la brique EV3.

ETAPE 2) Allumer la brique EV3.

ETAPE 3) - dérouler le menu "outils"

- sélectionner "Wifi" ;
- cocher la case "Wifi" ;

- sélectionner "Connections" ;
- sélectionner le point d'accès Wifi dans la liste, puis attendre ;
- saisir la clé WEP/WPA si nécessaire ;
- sélectionner "Connect" ;
- dérouler le menu "outils" ;
- sélectionner "Brick info" ;
- conserver l'adresse IP de la brique EV3 (par exemple 192.168.1.2) pour un usage ultérieur.

ETAPE 4) Il faut établir un partage de connexion 4G entre notre Android, PC et le contrôleur afin que ces trois appartiennent au même réseau. Une nouvelle adresse IP s'affichera à l'écran du contrôleur de la brick Lejos.

ETAPE 5) Définir le contrôleur Lejos comme serveur plutôt que client en déroulant et cliquant sur l'icône Wifi ensuite sur Access point pt+ et entrez l'adresse IP de votre brick et validez.

ETAPE 6) Dans la ligne de commande de votre PC lancez l'instruction: telnet suivie de la nouvelle adresse IP affichée sur la brick et rentrez comme login 'root' et sans mot de passe.

ETAPE 6bis) Si la commande Telnet n'est pas reconnu: C:>telnet google.com 80

'Telnet' is not recognized as an internal or external command, operable program or batch file.

Cause: Le client Telnet dans un Système d'exploitation Windows est désactivé par défaut.

Solution-ligne-de-commande : Lancez la commande suivante en disposant d'autorisations de niveau Administrateur:

dism /online /Enable-Feature/FeatureName:TelnetClient

Solution-configuration-manuelle : commencez par rechercher et cliquer sur 'Turn Windows features on or off' à partir du domaine de recherche près du Start button. Ensuite, sur la page qui s'affiche recherchez et sélectionnez Telnet Client et pour finir cliquez sur Ok.

Vous aurez par la suite la page de chargement des changements en titre : 'Apply Changes'. Une fois que les changements sont réalisés, vous pouvez enfin cliquer sur 'Close'.

ETAPE 7)root@EV3: # dropbear

root@EV3: # hcitool dev

Devices:

hci0 11:22:33:44:55:

ETAPE 8) Lancer la Classe 'TestUSBControl', voir l'Annexe D

ETAPE 9) Attendre l'affichage de 'Waiting' sur l'écran du contrôleur qui marque l'attente d'une connexion.

ETAPE 10) Lancer la Classe 'Client WIFI', voir l'Annexe G

ETAPE 11) Pour Ouvrir le portail, entrez 1 sur votre Consol et pour le fermer entrez 2 et afin terminer la connexion entrez Leave.

C.7 Annexe G

Code source de contrôle à lancer sur le PC pour une connexion WIFI

```
package org.client;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;
import java.io.OutputStream;
import java.io.PrintWriter;

public class ClientWIFI {

        private static Scanner sc;

        public static void checkConnection(){
                String text ="";
                sc = new Scanner(System.in);
                do
                {
                        text = sc.nextLine();
                }while(!text.equals("connect"));

        }

        public static void main(String[] args) {
                checkConnection();
                try(Socket socket = new Socket("192.168.43.37",5555))
                {
                        OutputStream output = socket.getOutputStream();
                        PrintWriter writer = new PrintWriter(output,true);
                        String text;

                        do
                        {
                                text = sc.nextLine();
                                writer.println(text);
                        }while(!text.equals("leave"));

                        socket.close();
                }
                catch(UnknownHostException e)
                {
                        System.out.println("Server not found");
                }
                catch(IOException e)
                {
                        System.out.println("I/O error");
                }
        }
}
```

C.8 Annexe H

ATL Transformation Rules, sequence diagram to xml

```
-- @path SequenceXMI=/SequenceXMI2XML/sequenceXMI.ecore
-- @path XML=/SequenceXMI2XML/XML.ecore

module SequenceXMI2XML;

create OUT: XML from IN: SequenceXMI;

helper context SequenceXMI!Message def: isContained(s : String): Boolean=
        if s.split(' ')->select(m | m=self.sendEvent).notEmpty() then
                true
        else
                false
        endif;


rule project{
        from
                s : SequenceXMI!packagedElement
        to
                t : XML!Projet(
                        Class <- s.lifeline
                        ->select(l | l.oclIsTypeOf(SequenceXMI!Lifeline))
                        ->collect(l | thisModule.lifeline2class(l))
                )
}

lazy rule lifeline2class{
        from
                s : SequenceXMI!Lifeline
        to
                t1 : XML!Class(
                        name <- s.name,
                        hardware <- '',
                        Fonctions <- t2,
                        ConnectionInfo <- t3
                ),
                t2 : XML!Fonctions(
                        fonction <-
                           SequenceXMI!packagedElement.allInstances().first().message
                                ->select(m | m.isContained(s.coveredBy))
                                ->collect(m |
                                   thisModule.message2fonction(m))
                ),
                t3 : XML!ConnectionInfo(
                        type <- '',
                        role <- '',
                        ip <- '',
                        port <- '',
                        inputType <- '',
                        outputType <- ''
                )
```

```
}

lazy rule message2fonction{
        from
                s : SequenceXMI!Message
        to
                t : XML!Fonction(
                        name <- s.name,
                        actionId <- ''
                )
}
```

C.9 Annexe I

Sequence diagramme UML file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uml:Model xmi:version="20131001"
    xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
    xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
    xmi:id="_ap9FQIicEeqlhcbsJUKvLg" name="garage">
  <packageImport xmi:type="uml:PackageImport"
      xmi:id="_WtKCkJOfEeqeCZPXz7N5rw">
    <importedPackage xmi:type="uml:Model"
        href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#_0"/>
  </packageImport>
  <packagedElement xmi:type="uml:Interaction"
      xmi:id="_LpWq8JRwEeqMFZR3MbZmcg" name="Interaction1">
    <lifeline xmi:type="uml:Lifeline" xmi:id="_NDpesJRwEeqMFZR3MbZmcg"
        name="Client" coveredBy="_Th2RMJRwEeqMFZR3MbZmcg
        _TicuIJRwEeqMFZR3MbZmcg"/>
    <lifeline xmi:type="uml:Lifeline" xmi:id="_QGpVoJRwEeqMFZR3MbZmcg"
        name="EV3" coveredBy="_Th2RMZRwEeqMFZR3MbZmcg
            _TicHEZRwEeqMFZR3MbZmcg _U1SPAJRwEeqMFZR3MbZmcg
                _U1S2EJRwEeqMFZR3MbZmcg
            _ceuFkJRwEeqMFZR3MbZmcg _ceuFkZRwEeqMFZR3MbZmcg
                _gENlkJRwEeqMFZR3MbZmcg
            _gEOMoJRwEeqMFZR3MbZmcg _TiX1oJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_Th2RMJRwEeqMFZR3MbZmcg" name="Message3SendEvent"
        covered="_NDpesJRwEeqMFZR3MbZmcg"
        message="_Th1qIJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_Th2RMZRwEeqMFZR3MbZmcg" name="Message3ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_Th1qIJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:BehaviorExecutionSpecification"
        xmi:id="_TiX1oJRwEeqMFZR3MbZmcg"
        name="BehaviorExecutionSpecification6"
        covered="_QGpVoJRwEeqMFZR3MbZmcg" finish="_TicHEZRwEeqMFZR3MbZmcg"
            start="_Th2RMZRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_TicHEZRwEeqMFZR3MbZmcg" name="Message8SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_TicHEJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_TicuIJRwEeqMFZR3MbZmcg" name="Message8ReceiveEvent"
        covered="_NDpesJRwEeqMFZR3MbZmcg"
        message="_TicHEJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_U1SPAJRwEeqMFZR3MbZmcg" name="Message10SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_U1RA4JRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_U1S2EJRwEeqMFZR3MbZmcg" name="Message10ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_U1RA4JRwEeqMFZR3MbZmcg"/>
```

```
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_ceuFkJRwEeqMFZR3MbZmcg" name="Message13SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_cetegJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_ceuFkZRwEeqMFZR3MbZmcg" name="Message13ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_cetegJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_gENlkJRwEeqMFZR3MbZmcg" name="Message16SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_gEM-gJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_gEOMoJRwEeqMFZR3MbZmcg" name="Message16ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_gEM-gJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_Th1qIJRwEeqMFZR3MbZmcg"
        name="checkConnexion()" receiveEvent="_Th2RMZRwEeqMFZR3MbZmcg"
        sendEvent="_Th2RMJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_TicHEJRwEeqMFZR3MbZmcg"
        name="" messageSort="reply" receiveEvent="_TicuIJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_U1RA4JRwEeqMFZR3MbZmcg"
        name="isOpen()" receiveEvent="_U1S2EJRwEeqMFZR3MbZmcg"
        sendEvent="_U1SPAJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_cetegJRwEeqMFZR3MbZmcg"
        name="openDoor()" messageSort="asynchCall"
        receiveEvent="_ceuFkZRwEeqMFZR3MbZmcg"
        sendEvent="_ceuFkJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_gEM-gJRwEeqMFZR3MbZmcg"
        name="closeDoor()" messageSort="asynchCall"
        receiveEvent="_gEOMoJRwEeqMFZR3MbZmcg"
        sendEvent="_gENlkJRwEeqMFZR3MbZmcg"/>
  </packagedElement>
</uml:Model>
```

C.10 Annexe J

Sequence diagramme source file for ATL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="20131001"
    xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
    xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
    xmi:id="_ap9FQIicEeqlhcbsJUKvLg" name="garage" xmlns="SequenceXMI">
  <packageImport xmi:type="uml:PackageImport"
      xmi:id="_WtKCkJOfEeqeCZPXz7N5rw">
    <importedPackage xmi:type="uml:Model"
        href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#_0"/>
  </packageImport>
  <packagedElement xmi:type="uml:Interaction"
      xmi:id="_LpWq8JRwEeqMFZR3MbZmcg" name="Interaction1">
    <lifeline xmi:type="uml:Lifeline" xmi:id="_NDpesJRwEeqMFZR3MbZmcg"
        name="Client" coveredBy="_Th2RMJRwEeqMFZR3MbZmcg
        _TicuIJRwEeqMFZR3MbZmcg"/>
    <lifeline xmi:type="uml:Lifeline" xmi:id="_QGpVoJRwEeqMFZR3MbZmcg"
        name="EV3" coveredBy="_Th2RMZRwEeqMFZR3MbZmcg
            _TicHEZRwEeqMFZR3MbZmcg _U1SPAJRwEeqMFZR3MbZmcg
                _U1S2EJRwEeqMFZR3MbZmcg
            _ceuFkJRwEeqMFZR3MbZmcg _ceuFkZRwEeqMFZR3MbZmcg
                _gENlkJRwEeqMFZR3MbZmcg
            _gEOMoJRwEeqMFZR3MbZmcg _TiX1oJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_Th2RMJRwEeqMFZR3MbZmcg" name="Message3SendEvent"
        covered="_NDpesJRwEeqMFZR3MbZmcg"
        message="_Th1qIJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_Th2RMZRwEeqMFZR3MbZmcg" name="Message3ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_Th1qIJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:BehaviorExecutionSpecification"
        xmi:id="_TiX1oJRwEeqMFZR3MbZmcg"
        name="BehaviorExecutionSpecification6"
        covered="_QGpVoJRwEeqMFZR3MbZmcg" finish="_TicHEZRwEeqMFZR3MbZmcg"
            start="_Th2RMZRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_TicHEZRwEeqMFZR3MbZmcg" name="Message8SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_TicHEJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_TicuIJRwEeqMFZR3MbZmcg" name="Message8ReceiveEvent"
        covered="_NDpesJRwEeqMFZR3MbZmcg"
        message="_TicHEJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_U1SPAJRwEeqMFZR3MbZmcg" name="Message10SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_U1RA4JRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_U1S2EJRwEeqMFZR3MbZmcg" name="Message10ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_U1RA4JRwEeqMFZR3MbZmcg"/>
```

```
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_ceuFkJRwEeqMFZR3MbZmcg" name="Message13SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_cetegJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_ceuFkZRwEeqMFZR3MbZmcg" name="Message13ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_cetegJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_gENlkJRwEeqMFZR3MbZmcg" name="Message16SendEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_gEM-gJRwEeqMFZR3MbZmcg"/>
    <fragment xmi:type="uml:MessageOccurrenceSpecification"
        xmi:id="_gEOMoJRwEeqMFZR3MbZmcg" name="Message16ReceiveEvent"
        covered="_QGpVoJRwEeqMFZR3MbZmcg"
        message="_gEM-gJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_Th1qIJRwEeqMFZR3MbZmcg"
        name="checkConnexion()" receiveEvent="_Th2RMZRwEeqMFZR3MbZmcg"
        sendEvent="_Th2RMJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_TicHEJRwEeqMFZR3MbZmcg"
        name="" messageSort="reply" receiveEvent="_TicuIJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_U1RA4JRwEeqMFZR3MbZmcg"
        name="isOpen()" receiveEvent="_U1S2EJRwEeqMFZR3MbZmcg"
        sendEvent="_U1SPAJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_cetegJRwEeqMFZR3MbZmcg"
        name="openDoor()" messageSort="asynchCall"
        receiveEvent="_ceuFkZRwEeqMFZR3MbZmcg"
        sendEvent="_ceuFkJRwEeqMFZR3MbZmcg"/>
    <message xmi:type="uml:Message" xmi:id="_gEM-gJRwEeqMFZR3MbZmcg"
        name="closeDoor()" messageSort="asynchCall"
        receiveEvent="_gEOMoJRwEeqMFZR3MbZmcg"
        sendEvent="_gENlkJRwEeqMFZR3MbZmcg"/>
  </packagedElement>
</xmi:XMI>
```

C.11 Annexe K

Sequence Diagram Metamodel

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    name="SequenceXMI">
  <eClassifiers xsi:type="ecore:EClass" name="packageImport">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:type"
        lowerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:id"
        lowerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="importedPackage"
        lowerBound="1" eType="#//importedPackage"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="importedPackage">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:type"
        lowerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="href"
        lowerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="packagedElement">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:type"
        owerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:id"
        lowerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures  xsi:type="ecore:EAttribute" name="name"
        lowerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="lifeline"
        upperBound="-1"
        eType="#//Lifeline"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="fragment"
        upperBound="-1"
        eType="#//Fragment"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="message"
        upperBound="-1"
        eType="#//Message"/>
```

```
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Lifeline">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:type"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:id"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="coveredBy"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Fragment">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:type"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:id"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="covered"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="message"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="start"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Message">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:type"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="xmi:id"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="messageSort"
      lowerBound="1"
       eType="ecore:EDataType
          http://www.eclipse.org/emf/2002/Ecore#//EString"/>
```

```
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="receiveEvent"
        lowerBound="1"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="sendEvent"
        eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

C.12 Annexe L

XML output file metamodel

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="XML">
  <eClassifiers xsi:type="ecore:EClass" name="Projet">
    <eStructuralFeatures xsi:type="ecore:EReference" name="Class"
        lowerBound="1" upperBound="-1"
         eType="#//Class" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Class">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="hardware"
        lowerBound="1"
         eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ConnectionInfo"
        lowerBound="1"
         eType="#//ConnectionInfo" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Fonctions"
        lowerBound="1"
         eType="#//Fonctions" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Fonctions">
    <eStructuralFeatures xsi:type="ecore:EReference" name="fonction"
        upperBound="-1"
         eType="#//Fonction" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Fonction">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="actionId"
        lowerBound="1"
         eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ConnectionInfo">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="inputType"
        lowerBound="1"
         eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="role"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ip"
        lowerBound="1" eType="ecore:EDataType
```

```
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="port"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="inputMapper"
        lowerBound="1"
         eType="#//InputMapper"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="outputType"
        lowerBound="1"
         eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="InputMapper">
    <eStructuralFeatures xsi:type="ecore:EReference" name="exitCondition"
        lowerBound="1"
         eType="#//ExitCondition"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ExitCondition">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="condition"
        lowerBound="1"
         eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Mapper">
    <eStructuralFeatures xsi:type="ecore:EReference" name="guard"
        lowerBound="1" eType="#//Guard"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="actionReturn"
        lowerBound="1"
         eType="#//ActionReturn"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ActionReturn">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Guard">
    <eStructuralFeatures xsi:type="ecore:EReference" name="value"
        lowerBound="1" eType="#//Value"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Value">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="value"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Action">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
        lowerBound="1" eType="ecore:EDataType
        http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="expect"
        lowerBound="1"
         eType="ecore:EDataType
            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

C.13 Annexe M

final XML file (before manual completion)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Projet xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="XML">
  <Class name="Client" hardware="">
    <ConnectionInfo type="" inputType="" role="" ip="" port=""
       outputType=""/>
    <Fonctions>
      <fonction name="checkConnexion()" actionId=""/>
    </Fonctions>
  </Class>
  <Class name="EV3" hardware="">
    <ConnectionInfo type="" inputType="" role="" ip="" port=""
       outputType=""/>
    <Fonctions>
      <fonction name="isOpen()" actionId=""/>
      <fonction name="openDoor()" actionId=""/>
      <fonction name="closeDoor()" actionId=""/>
    </Fonctions>
  </Class>
</Projet>
```

C.14 Annexe N

final XML file (after manual completion)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Projet xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="XML">
  <Class name="Client" hardware="PC">
    <ConnectionInfo type="USB" role="Client" ip="192.168.43.208"
       port="5555" outputType="text"/>
    <Fonctions>
      <fonction name="checkConnexion" actionId="A111"/>
    </Fonctions>
  </Class>
  <Class name="EV3" hardware="EV3">
    <ConnectionInfo type="USB" inputType="text" role="Server"
       ip="192.168.43.208" port="5555">
    <InputMapper>

                <Mapper>
                        <guard>
                                <value>1</value>
                                <action name="isOpen" expect="false"/>
                        </guard>
                        <actionReturn name="openDoor"/>
                </Mapper>

                <Mapper>
                        <guard>
                           <value>2</value>
                           <action name="isOpen" expect="true"/>
                        </guard>
                        <actionReturn name="closeDoor"/>
                </Mapper>

</InputMapper>
    </ConnectionInfo>
    <Fonctions>
      <fonction name="isOpen" actionId="A110"/>
      <fonction name="openDoor" actionId="A100"/>
      <fonction name="closeDoor" actionId="A101"/>
    </Fonctions>
  </Class>
</Projet>
```

C.15 Annexe O

Generated Java Code (Client)

```
import java.net.Socket;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Scanner;
import java.util.Scanner;

public class Client{

        public static void main(String[] args){
                try(Socket socket = new Socket("192.168.43.208",5555)){
                        OutputStream output = socket.getOutputStream();
                        PrintWriter writer = new PrintWriter(output,true);
                        Scanner sc = new Scanner(System.in);
                        String text;
                        do{
                                text = sc.nextLine();
                                writer.println(text);
                        }while(!text.equals("leave"));
                        socket.close();
                }
                catch(IOException e){
                        System.out.println("Error");
                }
        }

        private static Scanner sc;
        public static void checkConnexion(){
                String text = "";
                sc = new Scanner(System.in);
                do{
                        text = sc.nextLine();
                }while(!text.equals("connect"));
        }

}
```

C.16 Annexe P

Generated Java code (Server)

```
import java.net.Socket;
import java.io.IOException;
import java.net.ServerSocket;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import lejos.hardware.port.SensorPort;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.robotics.SampleProvider;
import lejos.hardware.motor.Motor;
import lejos.hardware.motor.Motor;

public class EV3{

        public static void main(String[] args){
                try(ServerSocket socket = new ServerSocket(5555)){
                        Socket connectionSocket = socket.accept();
                        BufferedReader reader = new BufferedReader(new
                           InputStreamReader(connectionSocket.getInputStream()));
                        String text;
                        do{
                                text = reader.readLine();
                                if(text.equals("1") /*&& !isOpen()*/){
                                        openDoor();
                                }
                                if(text.equals("2") /*&& isOpen()*/){
                                        closeDoor();
                                }
                        }while(!text.equals("leave"));
                        socket.close();
                }
                catch(IOException e){
                        e.printStackTrace();
                }
        }

        //private static EV3UltrasonicSensor uSensor;
        /*public static boolean isOpen(){
                SampleProvider sampleProvider = uSensor.getDistanceMode();
                float[] sample = new float[sampleProvider.sampleSize()];
                sampleProvider.fetchSample(sample, 0);
                if(sample[0]>8){
                        return true;
                }
                return false;
        }*/
        public static void openDoor(){
                Motor.A.setSpeed(90);
                Motor.D.setSpeed(90);
                Motor.A.forward();
                Motor.D.backward();
                Motor.A.rotateTo(-90);
```

```
                Motor.D.stop();
                Motor.A.stop();
        }
        public static void closeDoor(){
                Motor.A.setSpeed(90);
                Motor.D.setSpeed(90);
                Motor.A.forward();
                Motor.D.forward();
                Motor.A.rotateTo(0);
                Motor.D.stop();
                Motor.A.stop();
        }

}
```

C.17 Annexe Q

Document Type Definition (DTD) for final XML

```
<!ELEMENT Projet (Class+)>
        <!ELEMENT Class (ConnectionInfo?,Fonctions?)>
        <!ATTLIST Class name CDATA #REQUIRED>
        <!ATTLIST Class hardware CDATA #IMPLIED>

                <!ELEMENT ConnectionInfo (InputMapper?)>
                <!ATTLIST ConnectionInfo type CDATA #REQUIRED>
                <!ATTLIST ConnectionInfo inputType CDATA #IMPLIED>
                <!ATTLIST ConnectionInfo role (Server|Client) #REQUIRED>
                <!ATTLIST ConnectionInfo ip CDATA #REQUIRED>
                <!ATTLIST ConnectionInfo port CDATA #REQUIRED>
                <!ATTLIST ConnectionInfo outputType CDATA #IMPLIED>

                <!ELEMENT Fonctions (fonction+)>
                        <!ELEMENT fonction EMPTY>
                        <!ATTLIST fonction name CDATA #REQUIRED>
                        <!ATTLIST fonction actionID CDATA #REQUIRED>
                <!ELEMENT InputMapper (Mapper+)>
                <!ELEMENT Mapper (guard,actionReturn)>
                <!ELEMENT guard (value,action)>
                <!ELEMENT value (#PCDATA)>
                <!ELEMENT action EMPTY>
                <!ATTLIST action name CDATA #REQUIRED>
                <!ATTLIST action expect (true|false) #REQUIRED>
                <!ELEMENT actionReturn EMPTY>
                <!ATTLIST actionReturn name CDATA #REQUIRED>
```