Refinement of Communication Protocols by Model Transformation

a TER 2021 Article

GUERIN Antoine, ROZEN Anthony, GICQUEL Alexandre
Supervised by ANDRÉ Pascal

Master Informatique ALMA - Université of Nantes - LS2N Laboratory - AELOS Team, France
firstname.lastname@etu.univ-nantes.fr

Abstract. Nowadays in computer systems, distant communications are more and more important, especially thanks to new technologies such as 5G which allows to extend the communication perimeter compared to old technologies. Model engineering aims to shorten the development cycle by focusing on abstractions and partially automating code generation. In this article, we will observe the results of the first case study implemented. And we will deduce a first layer of abstraction for the communications that will allow to form a first basis for a model transformation software. To carry out the experiments, the programs will be deployed on the EV3 mindstorm robot and on Android. Thus we will be able to have access to several types of communications.

Keywords : Model engineering, Model transformation, Communication, Refinement, Code generation.

1 Introduction

Nowadays, software is playing an increasing role in automated production, whether in industry, mechanization or service with the assistance of robots and artificial intelligence. The launch of programs such as "Industry of the Future" have made it possible to converge efforts in cybernetics. We speak of Industry 4.0 for these innovations relating to the Internet of Things, robotics, artificial intelligence or big data. The project to be carried out consists in setting up a model-based approach (Model Engineering)[1] to build software that allows the generation of computer code from models with multiple communications, different or not, through model transformation.

UML model transformation into computer code aims to help developers maximise their time by reducing the time between analysis and design of software. This will allow developers to focus on more complex and project-specific tasks, rather than on communications that are general to any project.

At first glance, one might think that this project is simple to implement. However, if you look a little closer, it is quite the opposite because the implementation of a communication or a connection can differ according to the protocols used. Indeed, these protocols do not work in the same way and do not have the same constraints, for example in a WiFi protocol, a server can accept several clients, whereas in a Bluetooth protocol, a client and a server can only accept one connection. Furthermore, the implementation of a protocol is not the same from one system to another, especially for the Bluetooth protocol which does not have the same implementation of a connection if one uses a system programmed under the LeJos, or a system programmed under Android.

The objective of this project is to implement the process of transforming UML models into Java programs with an assisted and parameterized generation of code. However, in order to realize the model transformation software, we will start by implementing a concrete operational solution of the automatism to progressively go up in abstraction. This year we had to set up the case study comprising several and to study it to draw the first abstraction of the communications.

The article is organized as follows. We begin by introducing the background in Section 2, followed by one of the case study experiments. We illustrate the manual experiments of the case study in Section 3. The analysis and introspection of these experiments is unpacked in Section 4 and the primitive part in Section 5.Finally, in Section 6, we make a comparison with the different works of the previous years before concluding.

2 Background

The objective is to create a software production chain from models for connected systems. In particular, it is about devices communicating with each other with a "real" execution environment that takes into account the constraints of operation, security, operational reliability and performance, for example.[2,3]

Some features are general, others depend on the environment or the system itself. We can use one or several modeling languages (UML [4], SysML [5]), associated verification tools, simulation tools, etc.

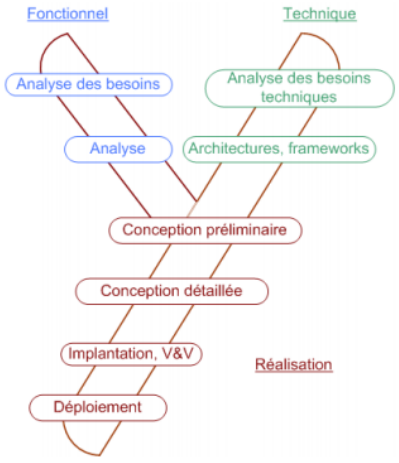

Fig. 1. Unified process 2TUP

The analysis model immersed in a technical environment will be called a design model, as illustrated by the Y-shaped development model in Fig. 1 inspired by [6]. For this purpose, communication tools based on the PLCs of automata, robots, sensors and actuators are used. Intermediate levels can be implemented to facilitate the

implementation of the code production chain, inspired by the model-driven approach and software production lines .

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

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

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

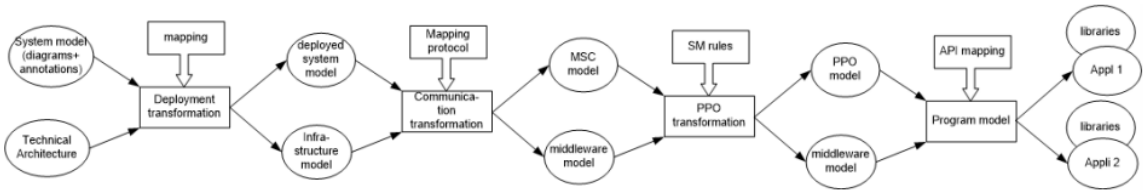

Fig. 2. General transformation process

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

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

The process in Fig. 2 is abstract but generic.

Case study The purpose of setting up a case study is to visualize how a communication between two computer systems works. This visualization will make it possible

to identify the identical protocols of communication compared to other software where a communication between systems intervenes. All this to arrive perhaps at a future automation of these protocols. 3 following case studies are possible:

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

Here, the case study involving a communication between a portal and a vehicle will be developed because nowadays, this is the closest communication to our daily life because automatic cars are not ubiquitous yet and are not yet 100% reliable to avoid collisions. And regarding the control of a vehicle from a tablet, this happens for remote controlled cars, either for children or during stunts for the cinema, so it is not a communication that can happen daily.

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

3 Manual experimentation

The source code that we have is recovered from the students having realized the same subject of research as us. Thus, each year, the group working on the research topic can go further than the previous group. The code we obtained only study one type of communication. So our goal was at least to make two types of communications, Bluetooth and WiFi, and to create a communication overlay in order to facilitate and optimize the work of programmers.

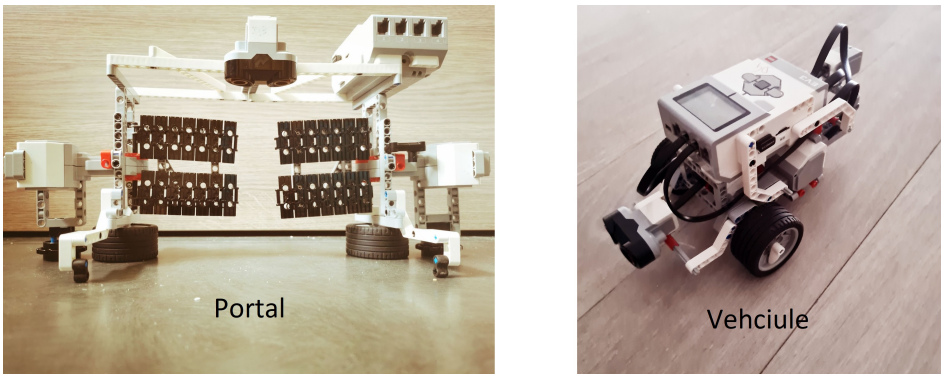

Fig. 3. Model of the portal and the vehicle

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

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

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

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

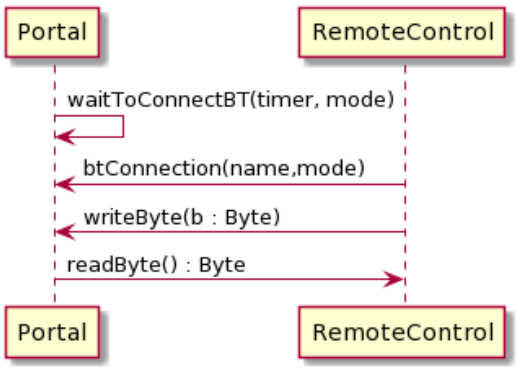

Fig. 4. The connection/communication between the portal and the remote control

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

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

activity to be launched in the list of possible activities (relative to the button on which the user has pressed).

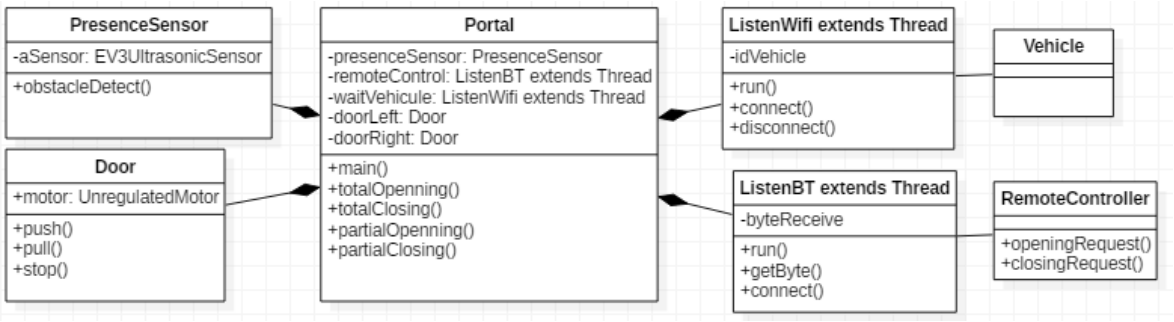

Fig. 5. Class diagram - Portal part

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

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

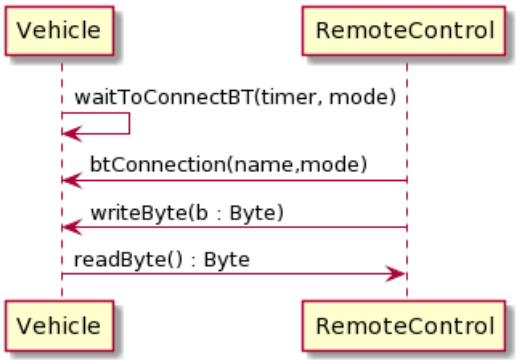

Fig. 6. The connection/communication between the vehicle and the remote control

The code of the vehicle remote control is modelled on the code of the gate remote control. This makes the vehicle less controllable but prevents the problems of overloading the data flow of the EV3 brick

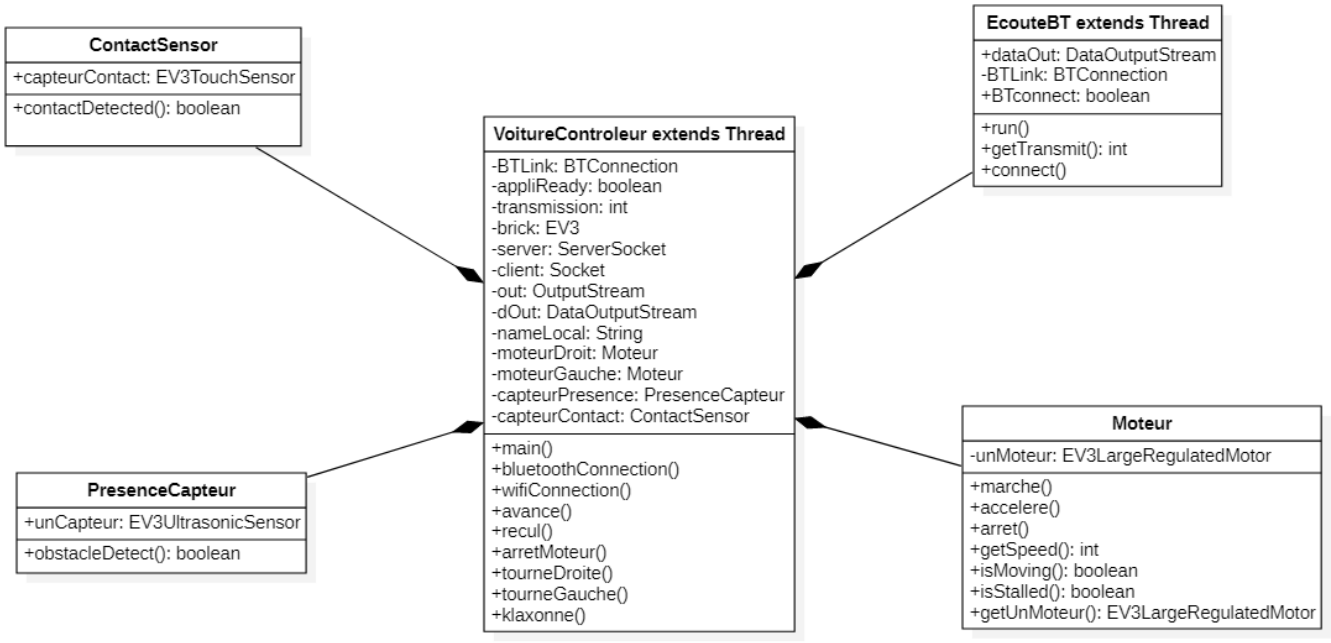

Fig. 7. Class diagram - Vehicle part

3.3 The connection/communication between the vehicle and the portal

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

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

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

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

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

To make this connection we have used the Sockets (Socket and SocketServer) of the Java framework. However, the WiFi connection must be made in parallel because otherwise while the vehicle or portal is establishing a WiFi connection with the other system, it will not be able to perform any task until the connection is established. To achieve this parallelization, the WiFi connection must be implemented in a thread. Furthermore, we must also put the communications in threads for the same reasons.

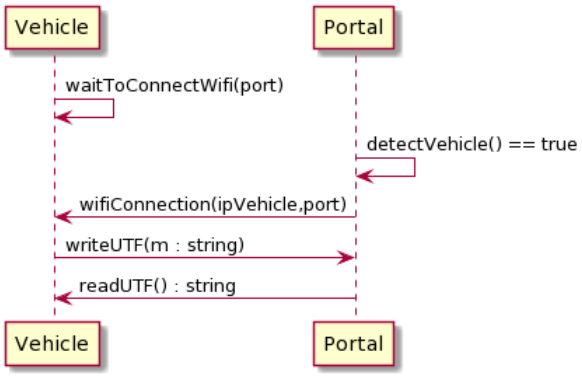

Fig. 8. The connection/communication between the vehicle and the portal

4 Code introspection and Connections/Communications

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

4.1 WIFI protocol

To capture the packets sent by a WiFi connection and communication between 2 Mindstorm EV3 robots, one corresponding to the vehicle and the other to the portal, we can use the Wireshark software. However, this software is only usable on a computer. So to be able to retrieve the sent packets, we first thought of using a USB cable to connect the computer to one of the two EV3 bricks. However, after some research on the internet and some tests, we didn't succeed in recovering the transmitted packets.

Fig. 9. Simplification of WiFi transmission

Then we thought of putting the computer between the two EV3 bricks when sending packets. This means that the computer will be the intermediary of the two bricks to recover the packets. (Fig. 9)

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

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

Fig. 10. Capture of packets WiFi

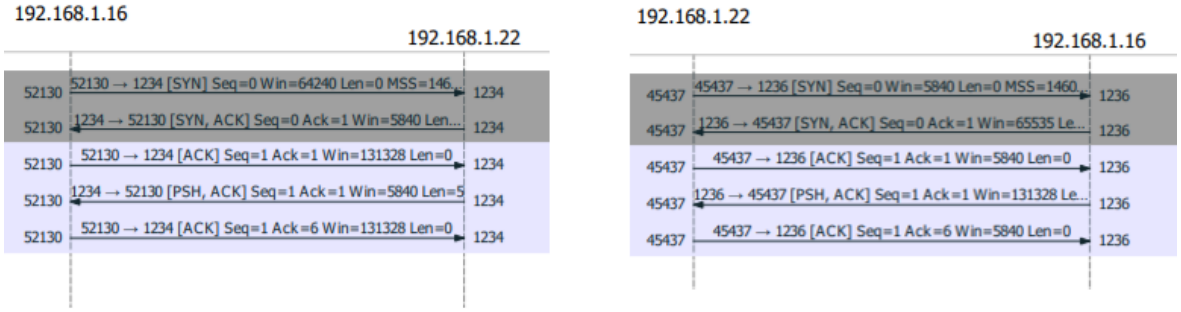

Fig. 11. Packet capture TCP flow graph

As we can see on the Fig. 10, the procedure of sending messages is the same as in the situation "vehicle -> portal (computer)". However, regarding the establishment of a connection, there are some changes, the following options have been added:

- TSval,
- Tsecr

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

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

To conclude, we have noticed that 3 packets were needed to establish a connection (the first three packets) and that only 2 packets were needed to send a communication (the last two packets). We also noticed that there were different options for a TCP connection depending on which system was the server or the client.(Fig. 10, Fig. 11)

4.2 Bluetooth protocol

Just like capturing the different packets sent between two Mindstorm EV3 robots, we are going to use the Wireshark software to observe the different packets sent between an EV3 and a remote control (which can actually be a phone or a computer emulator). However, the Wireshark software does not automatically integrate a solution to analyze Bluetooth protocols under Windows.[7] You have to think, when installing Wireshark, to choose to install the optional USBPcap package, allowing to capture USB traffic, because on most computers, Bluetooth is used through the USB bus. Then, to capture packets, you have to launch the capture on the USBPcap 1 interface.

The next problem is that, as we are trying to intercept packets on the computer, the phone cannot be the remote control, so the emulator on Android Studio must be the remote control. However, the Android Studio emulator does not include virtual software for Bluetooth, which means that we cannot use Bluetooth via the emulator.[8] This implies that we cannot use the computer as a remote control, only the phone can, but we cannot use the phone to pick up packets, only the computer can. A possible solution would be to capture Bluetooth packets via an application on the phone, but the rendering is unreadable and unusable.

5 Communication Primitives

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

5.1 Primitives analysis

In order to realize the code of this overlay, we must analyze the common actions carried out by the various types of communication (called primitives), analyze the parameters of the various types of connection and the services which will have to be integrated. This analysis is based in the first instance on the case study carried out, but can later be supplemented with new types of communication.

Thus the case study allowed us to note that between the Bluetooth and WiFi protocols, there is no difference for a message sending. However, concerning the establishment of the connection, each protocol has a different implementation. Moreover, depending on whether the system is a client or a server of the connection, this changes the internal implementation of the protocol. Therefore the communication overlay will be done in order to avoid code redundancy, so create an abstract class implementing an interface, and which will be declined in several subclasses each representing a protocol and a role (the role is either server or client).[9] To remain as generic as possible, a message sending must not depend on the type of the message sent. For this reason a Message type has been created allowing this generalization of messages.

After a more detailed study of the possible primitives we could perform, we realized several types of message sending. A synchronous message sending, which means that the system waits for an acknowledgement of receipt to be able to continue. And 2 asynchronous message sending, which means that the system can continue to perform

tasks even if it has not received an acknowledgement of receipt. Therefore, the primitives corresponding to these message sending and acknowledgement sending have been added. [9]

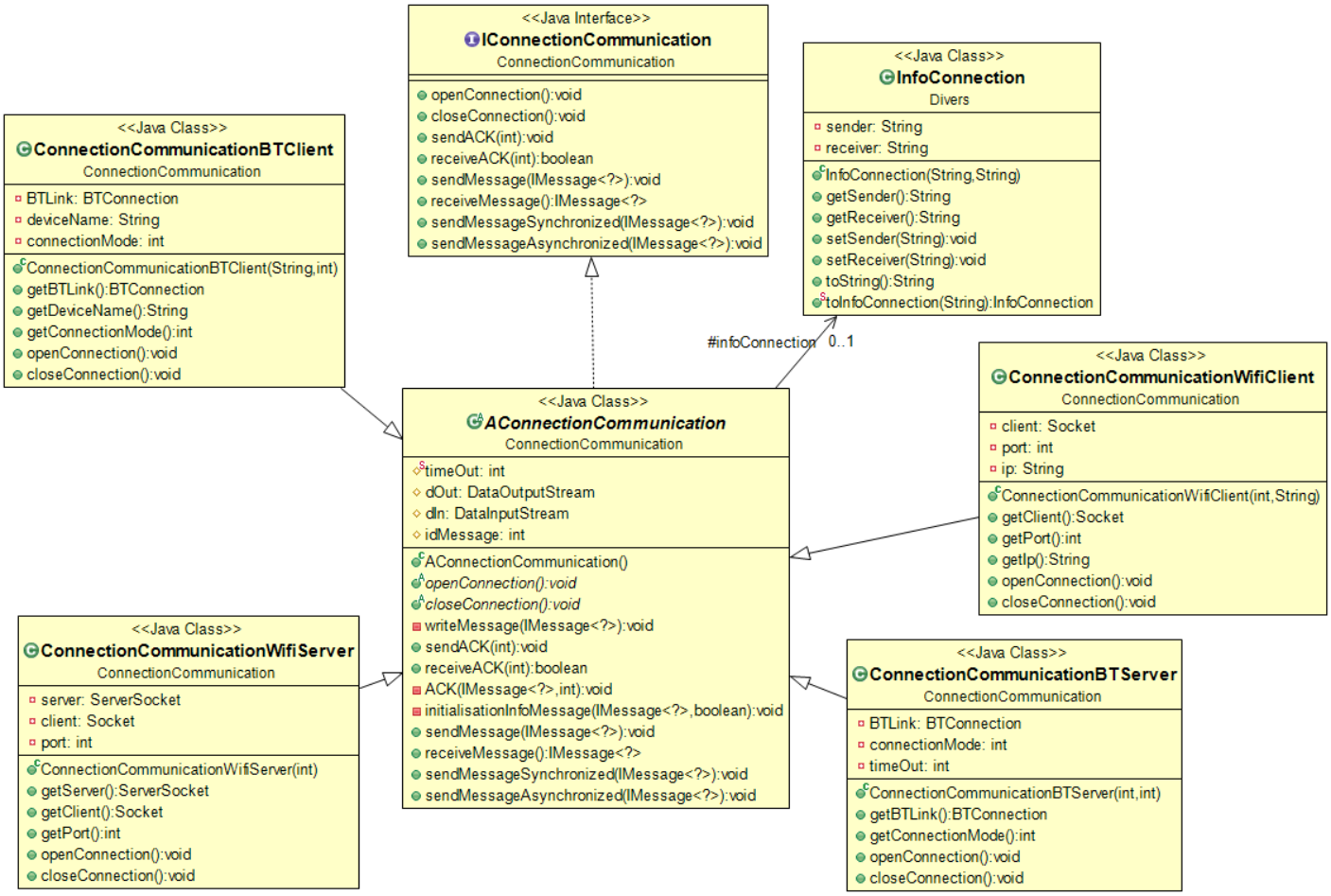

Fig. 12. Communication overlay class diagram

5.2 The conception of message sending

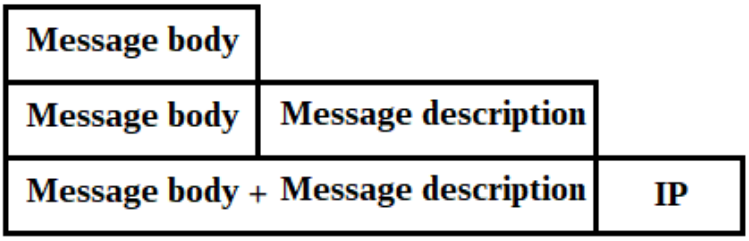

Fig. 13. Layers of the message sent

When a message is sent through the communication overlay, it fills the different layers composing the message. However the layer corresponding to the body of the message is initialized beforehand by the user of the overlay. Then the layer "Message description" is filled during the initialization of the type Message containing the body of the message. This one is composed of the identifier of the message, the type of the message and if the message must receive an acknowledgement of receipt or not. Nevertheless this one is modified during the use of the overlay because this one modifies the parameter "acknowledgement of receipt" according to the type of sending used (synchronous thus sending of acknowledgement of receipt, or asynchronous thus sending or not of an acknowledgement of receipt according to which asynchronous sending used). And finally the IP layer of the message corresponding to the IP of the sender and the IP of the receiver is filled by the overlay just before sending the message.
So sending a message between two systems using the communication overlay looks like this :

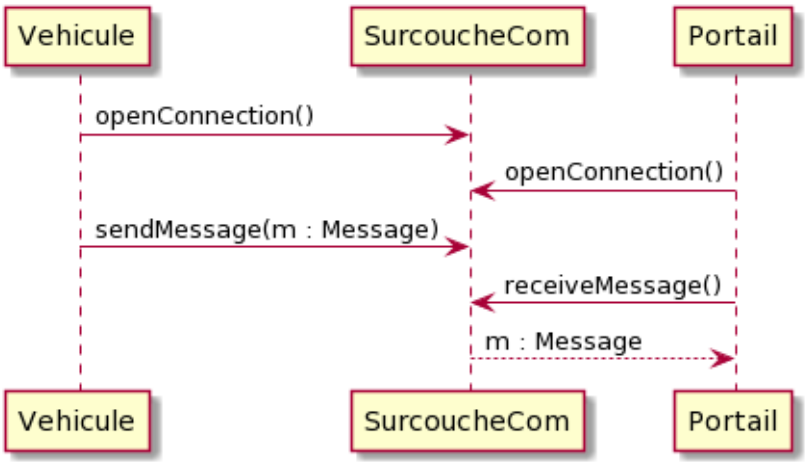

Fig. 14. Communication of the case study using the overlay

6 Related work

Ciccozzi et al [10] show that the transformation of UML models remains a difficult problem and is more suited to simulation than to software development, and has been implemented, for example, in model checking, execution via C++ [11] or MoKa/Papyrus [12,13]

On the work of the former groups that worked on this subject, we found that the code did not respect the requirements or the initial model, which although detailed, did not guarantee the consistency or completeness of the system specification.

Also the different wireless communications in the case study remained abstracted as sending messages in the model. Thanks to our evolution of the project, our model still remains a little abstract in the sense that the parameters provided remain consequent but it seems to us nevertheless essential in the sense that it will be able to adapt to the needs of each project via the parameterization or by replacing transformations by other transformations.

The human intervention in the transformations remains preponderant when there are alternatives, like the one above, communication protocols for sending messages.

7 Conclusion

Because of the health crisis, we were delayed. And because of this delay, we were not able to test several things like the new remote control that allows the portal to be the WiFi server, like the fact that the information about a Bluetooth connection is properly filled in, but also the testing of the modified case study with the communication overlay. These parts of the project still need to be tested. The implemented communication overlay allows a WiFi server to accept one and only one client. But a WiFi server can accept several clients at the same time. Now in the prototype file of the code-java, of the WiFi of this year is several track of implementation of this new overlay of which a server which accepts several client. So the next group working on this research topic will have to choose the best implementation of this new overlay. Now that the case study and the first abstraction have been completed, the next group will have to study how to implement the communication overlay in different projects where communications are involved (how to implement the overlay in a UML model). Then, they will have to derive a first prototype of model transformation from the realized UML model generalization.

The implementation of the communication overlay was the most important step for the progress of this project. After having grouped and completed the study of the cases, we were able to analyze the different protocols of the types of communication. We were able to draw this communication overlay in order to facilitate the use of exchanges between systems.

Enriching the models, formalizing the refinement processes, making modular and customizing the development to rely on transformation tools are the tracks followed. From a theoretical point of view, the transformation processes remain little explored. One perspective is to design an algebra of transformations to combine them. From a practical point of view, it is necessary to rationalize the software engineering process as

a combination of decisions and to experiment with a typology of transformations. From a tooling point of view, it is necessary to be able to combine transformations written with different languages and which are interactive so that the designer influences the design choices.

References

1. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. (2012)
2. André, P.: Case studies in model-driven reverse engineering. In: Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2019, Prague, Czech Republic, February 20-22, 2019. (2019) 256–263
3. André, P., Tebib, M.E.A.: Refining automation system control with MDE. In Hammoudi, S., Pires, L.F., Selic, B., eds.: Proceedings of MODELSWARD 2020, Valletta, Malta, February 25-27, 2020, SCITEPRESS (2020) 425–432
4. André, P., Vailly, A.: GÉNIE LOGICIEL - Développement de logiciels avec UML 2 et OCL - Cours, études de cas et exercices corrigés. Edition ellipses edn. Volume 6. Collection Technosup (2013)
5. Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML: Systems Modeling Language. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008)
6. Vallé, F., Roques, P.: UML 2 en action: De l'analyse des besoins à la conception. Collectio Architect Logiciel (2011)
7. : Analyse bluetooth protocols on windows using wireshark. {https://tewarid.github.io/2020/08/20/analyze-bluetooth-protocols-on-windows-using-wireshark.html}
8. : Android studio documentation. {https://developer.android.com/studio/run/emulator#wifi}
9. Gicquel, A., Guerin, A., Anthony, R.: Refinement of communication protocols by model transformation (May 2021)
10. Ciccozzi, F., Malavolta, I., Selic, B.: Execution of uml models: a systematic review of research and practice. Software & Systems Modeling **18**(3) (Jun 2019) 2313–2360
11. Ciccozzi, F.: On the automated translational execution of the action language for foundational uml. Software & Systems Modeling **17**(4) (Oct 2018) 1311–1337
12. Guermazi, S., Tatibouet, J., Cuccuru, A., Seidewitz, E., Dhouib, S., Gérard, S.: Executable modeling with fuml and alf in papyrus: Tooling and experiments. In: Proc. of the 1st International Workshop on Executable Modeling in (MODELS 2015)., Ottawa, Canada (September 2015) 3–8
13. Planas, E., Cabot, J., Gómez, C.: Lightweight and static verification of uml executable models. Comput. Lang. Syst. Struct. **46**(C) (November 2016) 66–90