Multipart asynchronous communication, application to mobile robots

GICQUEL Alexandre, GUERIN Antoine, ROZEN Anthony, ROCHETEAU Nathan
Supervised by ANDRÉ Pascal and CARDIN Olivier

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 implement a new communication protocol in order to increases the communication panel available to be able to carry out more transformation of models containing communications. 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, Multi-part communications

1 Introduction

As part of our Master, we have to carry out a technical project supervised by a researcher or a teacher-researcher, called Capstone. This project was to be carried out throughout the first semester of the year 2021/2022, in parallel with the lessons.

The subject « Multipart asynchronous communication, application to mobile robots », proposed by Mr. Pascal ANDRÉ, was a subject which followed our TER « Refinement of Communication Protocols by Model Transformation ». This TER research subject focused on the transformation on the transformation models for software where remote communications takes place, for example between a vehicle and a portal. In addition, it was necessary to analyze and detail communication protocols between two computer systems in order to automate a code generation from a model representing this communication. For the Capstone project, the primary objective is to abstract a new communications protocol for a future model transformation program containing communications. This new type of protocol will be added to the existing overlay created during the TER which already abstracts WiFi and Bluetooth communications.[1]

To carry out this project, we have set up a case study, representing a vehicle race. [2] Each vehicle is connected to a remote control via Bluetooth. A race controller is present in order to manage the race, such as the start signal. Finally, a central server using MQTT protocol is set up for communication between the race controller and the vehicles.

2 Mqtt

MQTT, "Message Queuing Telemetry Transport", is an open source messaging protocol that ensures non-permanent communications between devices by transporting their messages. It is also a standardized protocol based on TCP/IP. To communicate with MQTT, connected objects use a server, called a broker, that is to say a program in charge of receiving the published information in order to transmit it to the subscribed clients. The broker has a relay role, which means that it is the broker who transmits the information to the machine wishing by the one who sends the information (Fig. 1, in the appendix). There are several types of brokers: ActiveMQ, JoramMQ, Mosquitto or RabbitMQ. Mosquitto is the broker that will be used throughout the Capstone project. This broker is an open source mqtt server.[3]

Internal communication to the MQTT protocol is done via channels called topic. These greatly simplify communications because if one machine wants to read a message or transmit a message to another machine, it must send it on one channel listened to by the other machine. For example if a machine A wants to send a message to a machine B, A must send the message on the channel "iot-data" and B must be listening on the same channel. To send a message via MQTT, the publish() function is used, and to receive a message, the subscribe() function is used to listen on the channel and the MqttCallBack interface is used to perform an action when the message arrives.

In the project, the paho framework was used to program mqtt communications.

With MQTT, you can set the QoS (Quality Of Service), meaning that for each message sent you can choose how the broker should process it.

- QoS 0: The message sent is not stored by the broker. There is no acknowledgement of receipt. The message will be lost if the server or client is shut down. This is the default mode.
- QoS 1: The message will be delivered at least once. The client returns the message until the broker sends an acknowledgement of receipt.
- QoS 2: The broker saves the message and transmits it until it is received by all connected subscribers.

In this project, the QoS was left on 0.

The advantages of this protocol are its lightness, it requires only minimal resources and can therefore be used on small micro-controllers. In addition to simplifying communication, MQTT has been designed to maximize battery savings for mobile devices. MQTT uses 11 times less energy to send messages and 170 times less energy to receive messages than HTTP. MQTT is also 93 times faster than HTTP. This is why it is widely used in industry. [4] But one of the biggest advantages of this protocol is that several different machines such as cars, computers, etc, can communicate among themselves even if those are not programming on the same languages.

3 Race Controller

3.1 First approach

The race controller class aims to behave like a control tower. Every action that will happen on the race will first pass through the race controller. The race controller will manage to start the race when every vehicle is ready, to redirect bonuses and penalties from a vehicle to another, to receive all the timers from the vehicles and to store them before sorting them to obtain a final rank.

3.2 Technical approach

First of all, the race controller class contains a main static method. This class contains a Mqtt Client which is a new connection on "localhost" port 1883. Then this connection is opened. A Hashmap object is created to store the whole scores of the race so as to treat this data easily.

The next step is for the client to set itself listening on the vehicle channels that will participate to the race in order to be able to communicate with them. At the launch of the application, a Swing window opens and when all the vehicles are ready, the Race Master can click a button to launch the race (Fig. 2, in the appendix). Launching the game consist in sending a Command.START message to a channel containing all of the vehicles in the race. This Command.START message is just a security convention to be sure that all start messages will be the same. It aims to simplify the treatment of the information on vehicle side. After sending the messages, a copy of the System.currentTimeMillis() method's return value is done.

The following step is a looping section of code which purpose is to wait for messages to arrive from the other vehicles through the mqtt topics. If the message is a Command.Bonus then the bonus will be randomly chose by the race controller. If it's a red shell then all of the players except the sender can be shot by the shell. If it's a green shell then all of the player without exception can be shot. The choice is done randomly over the players according to the shell characteristics. Once the random vehicle has been picked up, the race controller sends him a penalty message. If the message is a Command.Finish one then a timer corresponding to the subtraction of the copy from the current value of System.currentTimeMillis() is stored in the box of the Hashmap corresponding to the vehicle sending the message. Once every Command.Finish is received from the whole players, the race is finished and the looping section is over.

Finally, the method sortByValue of the race controller is called on the Hashmap in order to sort all of the timers from the shortest one to the longest one. The final rank is then displayed to all of the players and the main method ends.

4 Vehicle and Remote control

4.1 Vehicle

The vehicle of this project is simply a model of construction of an EV3 robot. We used both motors as well as the color sensor for this case study.

In the simplified class diagram of the vehicle (Fig. 2, in the appendix), we can see the components of the EV3, the two motors and the color sensor. Motors are the main elements of the vehicle allowing it to move. The color sensor is used to read the different colors on the floor of the race:

- Red : the colour of the finish line.
- Yellow : the color of the bonuses/penalty

Then we have the communication components that connect the vehicle with the remote control and with the server.

Operation of communication with the remote control :

- Connection : A vehicle is controlled remotely by a remote control. In order to initialize the connection, the vehicle simply manages the Bluetooth reception of the data. That is to say, as long as the remote control does not ask to connect, the vehicle waits. When the Android app launches, the login test is started.
- Use : The remote sends different data when the buttons are pressed. The vehicle, according to the data received, performs an action. Example: Bit Received = 1 - the vehicle moves forward.

Operation of communication with the server :

- Connection : The vehicle automatically connects during initialization to the MQTT server. You just need the vehicle to have the IP address of the server and the port number. But you also need the server to be active.
- Use : a first communication is made at the start of the race. The MQTT server sends a message to the vehicle to warn it that the race has started. So the vehicle can start moving. The second communication is made at the finish of the race. The vehicle detects the red finish line thanks to its sensor and sends to the server that it has arrived. Then there may be other incoming or outgoing communication depending on the different types of bonuses or malus.

4.2 Remote control

The remote control that will control the vehicle is a mobile application, and they communicate together via Bluetooth, whether to send a message from the remote control to the vehicle, or to receive a message from the vehicle. Thereby, it is possible that a delay exists, because a Bluetooth transmission is not instantaneous.

The mobile application contains 2 activities, which are the screen for the Bluetooth connection to the vehicle, and the screen that allow the user to control the vehicle and to see some details about the race.

- Connection to the vehicle : In order to connect the user's mobile to the vehicle, the address mac of the Bluetooth of the vehicle must be entered by the user. (Fig. 3, in the appendix)

- Control of the vehicle : The user have 5 buttons available to control the vehicle : move forward, backward, turn right, turn left or break. When one button is pressed, a message containing a byte is sent to the vehicle, corresponding to the action the user wanted to do (Press *Backward* send the byte *2* to the vehicle). (Fig. 4, in the appendix)
- Details about the race : Some details about what happened in the race are shown to the user. For example, a countdown before the start of the race or, when the race is finished, the leaderboard. These informations are provided by the vehicle via the Bluetooth transmission, but the vehicle got these informations by the MQTT Server. (Fig. 4, in the appendix)

5 Communication Overlay

A communication overlay is a layer that lies above all communications. It consists of integrating the different types of communication in a single layer by generalizing the common actions carried out during a communication. This overlay will therefore be used in the same way, regardless of the type of communication. And she won't have to worry about the type of message sent to stay as generic as possible.

To add this new communication protocol, the MQTT protocol, it was necessary to modify the old overlay at the level of the sends and received acknowledgements of receipt because it is easier to give the message as parameter of these functions than the identifier of the message if it is not known to the user (although these functions are not often used by the user). But it was also necessary to add new communication primitive or modify some because when sending or receiving a message via MQTT, it is necessary to indicate on which channel to send or watch it received. So for sending a message via MQTT, just the message sending function (used by message sending primitive) has been rewritten because for sending Wifi/Bluetooth messages it is the write() function that is used while for MQTT, it is the publish() function that is used. To avoid major modifications and almost identical rewriting of the old message sending primitive, the message sending channel indication has been put into the generic message type by adding a new constructor for each message type and modifying the compression and message information due to that new constructor.

For the received of a message, the old primitive receiveMessage() has been rewritten to look on a default channel and also because for MQTT, it is more the read() function which is used to read a message but a function create by our care that is in the MqttCallback interface implementation, SimpleMqttCallback. Because for received messages via MQTT, it is necessary to create an implementation of MqttCallback that specifies what to do when the client receives a message. However it is not advisable to use this rewrite of this primitive because it is better to indicate on which channel you want to read a message. So another primitive of the same name, but with the addition of the channel and a keyword as a parameter was created. (the keyword refers to the standard of sending MQTT message create, explain afterwards).

In SimpleMqttCallback, when we receive a message, we decided to add a couple containing the message and the channel in a message list until it is processed by the user. So when the user processes a message, he must remove it from the list using the created function, removeMsg(). However, to facilitate the search for messages in the list of unprocessed messages and to avoid returning a message not expected by the user, a standard for writing MQTT messages was created. A MQTT message must be in the form of a keyWord:bodyOfMessage. The keyword corresponds to a word designating the subject of the message and it must be unique. The only problem with this solution is that the user who receives this message must know the keyword. And to avoid user error, a verification of the use of the standard was added when sending or receiving a message.

Finally, the rewriting of the Send/Receive acknowledgements was done for the same reason as the primitives.

6 Appendix

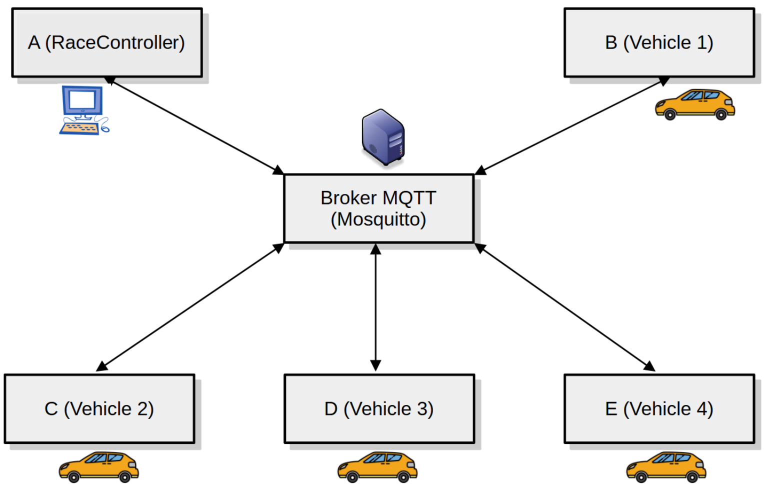


Fig. 1. Mqtt description

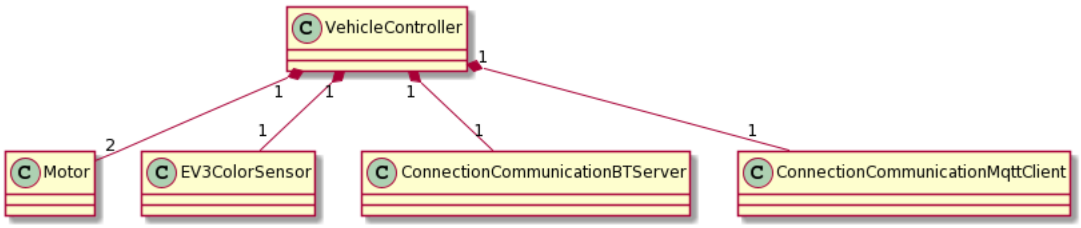

VehicleController
Motor
EV3ColorSensor
ConnectionCommunicationBTServer
ConnectionCommunicationMqttClient
1
1
1
1
1
2
1
1
1

Fig. 2. Class diagram - Vehicle part

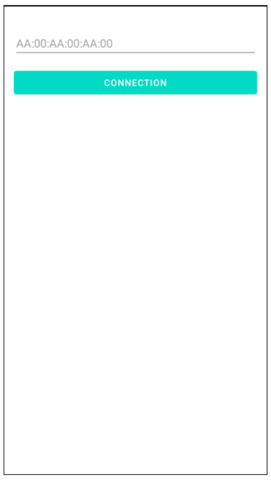


Fig. 3. Vehicle connection screen

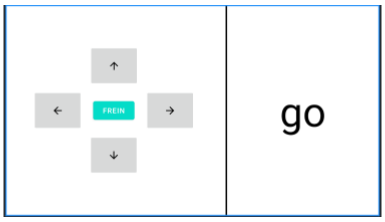
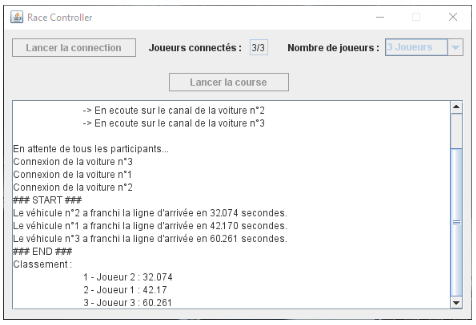


Fig. 4. Vehicle control screen



```
              -> En ecoute sur le canal de la voiture n°2
              -> En ecoute sur le canal de la voiture n°3

En attente de tous les participants...
Connexion de la voiture n°3
Connexion de la voiture n°1
Connexion de la voiture n°2
### START ###
Le véhicule n°2 a franchi la ligne d'arrivée en 32.074 secondes.
Le véhicule n°1 a franchi la ligne d'arrivée en 42.170 secondes.
Le véhicule n°3 a franchi la ligne d'arrivée en 60.261 secondes.
### END ###
Classement :
              1 - Joueur 2 : 32.074
              2 - Joueur 1 : 42.17
              3 - Joueur 3 : 60.261
```

Fig. 5. Swing Window Race Controller

References

1. : Ter report 2021-2022. {https://gitlab.univ-nantes.fr/ter-ir-2020/transfo-protocoles/-/blob/master/TER_S2_M1_informatique-2020-2021/TER-compte_rendus/reportTER21.pdf}
2. : Gitlab capstone 2021-2022. {https://gitlab.univ-nantes.fr/ter-ir-2020/transfo-protocoles/-/tree/master/Capstone2021-2022}

3. : Mqtt presentation. `{https://www.journaldunet.fr/web-tech/dictionnaire-de-l-iot/1440686-mqtt-comment-fonctionne-ce-protocole/}`
4. : Mqtt advantages. `{https://www.planete-domotique.com/blog/2021/03/17/protocole-mqtt-iot-domotique/}`