INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

GEI Department

Innovation Project

Wireless IO Tools

Authors:
Louis Chauvet
Alexandre Cros
Michael Ejigu
Luan Truong
Andy Xu
Thomas Zennaro

Mentors:
Olivier Boizot
François Malou
Stéphane Motta

January 18, 2021

Contents

I Introduction

This project is realized in the context of the Internet of Things project, in the fifth year of Innovative Smart Systems major at the National Institute of Applied Sciences of Toulouse (INSA). Our team is comprised of six engineering students, three Computer Science and Network engineering students and three Electronics and Automation students. The Smart Cabin project was chosen among a list of available projects for its industrial relevancy and the opportunity to work in cooperation with STERELA and Airbus.

1. Context

Our client is STERELA, an Airbus subcontractor. They are working on a proof of concept to demonstrate to Airbus the feasibility of a project in order to obtain funds from Airbus to advance to the production stage.

Airbus currently tests each plane on the ground before their initial flight. To do this, they need to connect different wires from a computer to different sensors. This way, the computer can send simulated data in order to retrieve sensor responses which allow to check the plane subsystems function correctly. The issue is that a cabin can be 70m long and the tests can have thousands of different configurations which require immense cable lengths, workforce and time to set up. The purpose of this project is to implement a wireless interface (gateway) between the plane and the computer (called the ESAO workstation, acronym for "Essais au Sol Assistés par Ordinateur") to manage tests for them to communicate. The gateway has to detect the plane's configuration, send test signals to the plane through a radio module and receive the returned values. In addition to this "translation work", the gateway must be an autonomous system in order to limit the time operators have to spend on it - for instance, it needs to update automatically.

STERELA is implementing an automated testing tool for Airbus which is comprised of two parts: a test box located on the plane which implements the tests, and a gateway to mediate wireless communication between the test box and a workstation. Making this communication wireless has many advantages such as making the testing easier, cheaper and faster as no cables are needed.

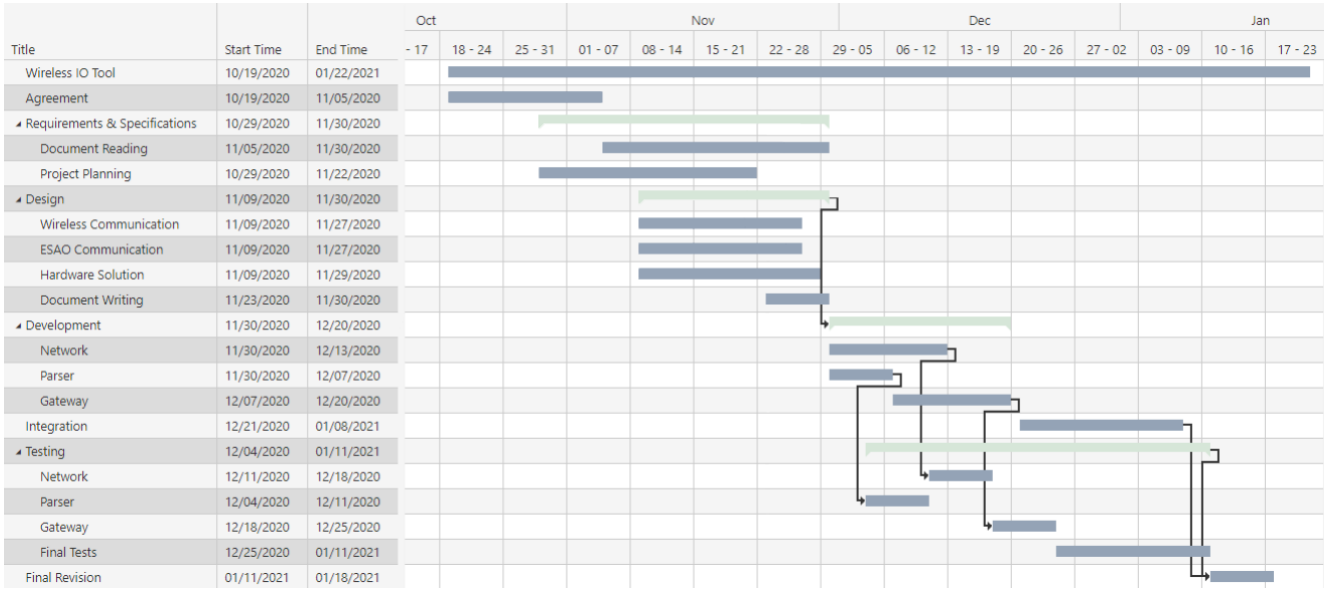
2. Specifications

The objective is to conceive and implement a gateway machine (PC or micro computer such as a Raspberry Pi) that serves the following functions:

- Communicate with the test box through a radio module (XBee)
- Communicate with the ESAO workstation through the IDMK3 protocol (Airbus Ethernet protocol based on UDP)
- Translate the orders from the ESAO workstation to the the test box, and vice versa (data received from the test box to the workstation).
- Capable of automatically updating
- Manage the timeouts (if a command fails to send, our machine has to send it again)

3. Timeline

To establish a first temporal organization we wrote a Gantt diagram (Figure I.1).



Title	Start Time	End Time
Wireless IO Tool	10/19/2020	01/22/2021
Agreement	10/19/2020	11/05/2020
Requirements & Specifications	10/29/2020	11/30/2020
Document Reading	11/05/2020	11/30/2020
Project Planning	10/29/2020	11/22/2020
Design	11/09/2020	11/30/2020
Wireless Communication	11/09/2020	11/27/2020
ESAO Communication	11/09/2020	11/27/2020
Hardware Solution	11/09/2020	11/29/2020
Document Writing	11/23/2020	11/30/2020
Development	11/30/2020	12/20/2020
Network	11/30/2020	12/13/2020
Parser	11/30/2020	12/07/2020
Gateway	12/07/2020	12/20/2020
Integration	12/21/2020	01/08/2021
Testing	12/04/2020	01/11/2021
Network	12/11/2020	12/18/2020
Parser	12/04/2020	12/11/2020
Gateway	12/18/2020	12/25/2020
Final Tests	12/25/2020	01/11/2021
Final Revision	01/11/2021	01/18/2021

Figure I.1: Gantt diagram of the project

The first step was the signature of the agreement. Because of the pandemic this took longer than expected. Following signature the client was able to send us all the useful documents to start working on the project. From that moment we divided our work into sub-teams in order to analyze and extract information from those documents. Next we started the conception part and did different reviews and validations with the client. Then we began the development on virtual machines set up specifically to give every team member the same development environment despite the distance.

As this Gantt diagram was an estimation we sometimes had to delay some milestones.

4. Financial aspects

As we are creating a proof of concept, the goal is not to have a very detailed version of the system. The architecture and scalability of our project is much more important to the client than the specific treatment of every single possible command. We are working on a simplified system that uses few commands and only one end-point for each side. The process does not require much power and our only requirements is for our machine to be able to interact with the radio module and the ESAO. Therefore only a USB and a Ethernet port are needed so that costs will be reduced.

Firstly this report will present all the project conception we developed around the hardware and the software. The second part present all the software specifications.

II

Conception

1. Hardware Conception

In this part we present the hardware conception we realized considering all the specifications and the clients requests.
There are several requirements on the hardware conception. The gateway has to communicate with an IP Ethernet network on one side and the XBee protocol on the other side. IP Ethernet is imposed by the existing test system which uses this protocol to work. The XBee protocol has been suggested by the client as tests are performed inside warehouses where there are different negative environmental effects for waves such as metal, posts and planes. XBee appeared as the best solution.

1.1. General overview of the complete hardware system

The diagram below (II.1) summarises all the components we use for the hardware of the interface between the gateway, the aircraft (XBee Module) and the ESAO (Ethernet). The gateway is composed of an XBee module (XB24CDMWIT-001) which needs an adapter (WRL-11812) to interact through the USB port with a machine that will run the code and communicate with the ESAO through Ethernet. All the communication protocols have been defined by the client, STERELA.

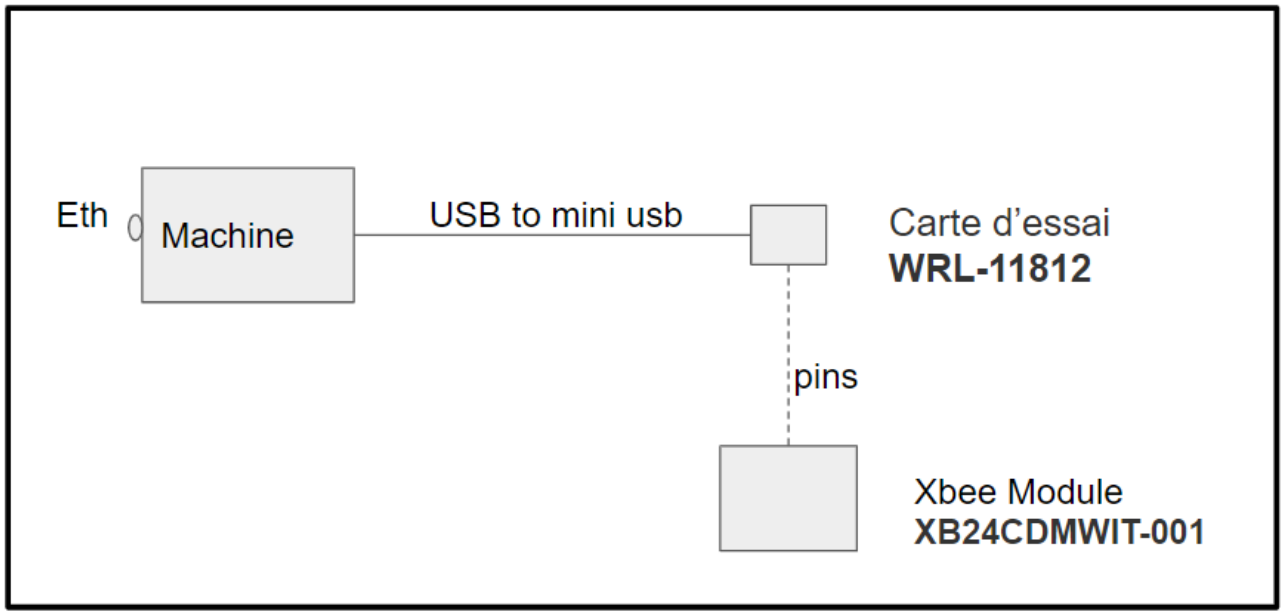


Figure II.1: Hardware scheme of our Gateway

1.2. XBee module features

The *Digi S2C DigiMesh 2.4* module (II.2) was recommended by the client. It is able to support an XBee mesh network architecture in case STERELA wants to connect several test boxes to one module.

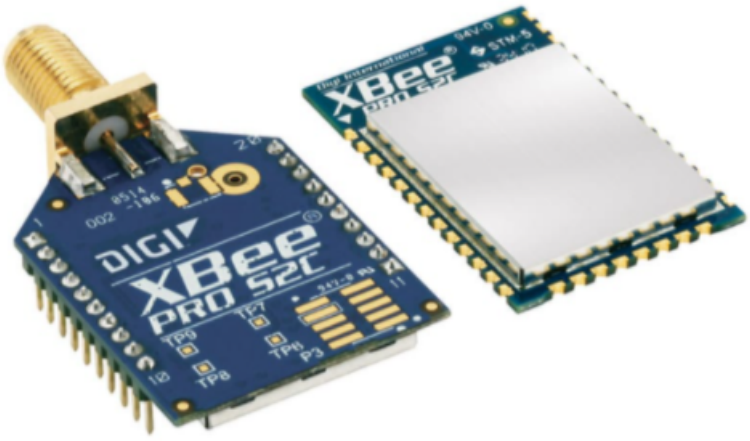


Figure II.2: XBee S2C DigiMesh 2.4

The XBee module needs an adapter to be connected to a machine using a USB connection. STERELA suggested us the *SparkFun XBee Explorer USB* adapter (II.3). It is possible to directly plug the module into this adapter in order to communicate by serial link with the XBee module (II.4).

Figure II.3: SparkFun XBee Explorer USB

Figure II.4: USB link of the XBee module

1.3. Electrical diagram of the XBee module, adapter and USB link package

Figure II.5 represents the complete electrical diagram including the adapter, the module and the USB pins.

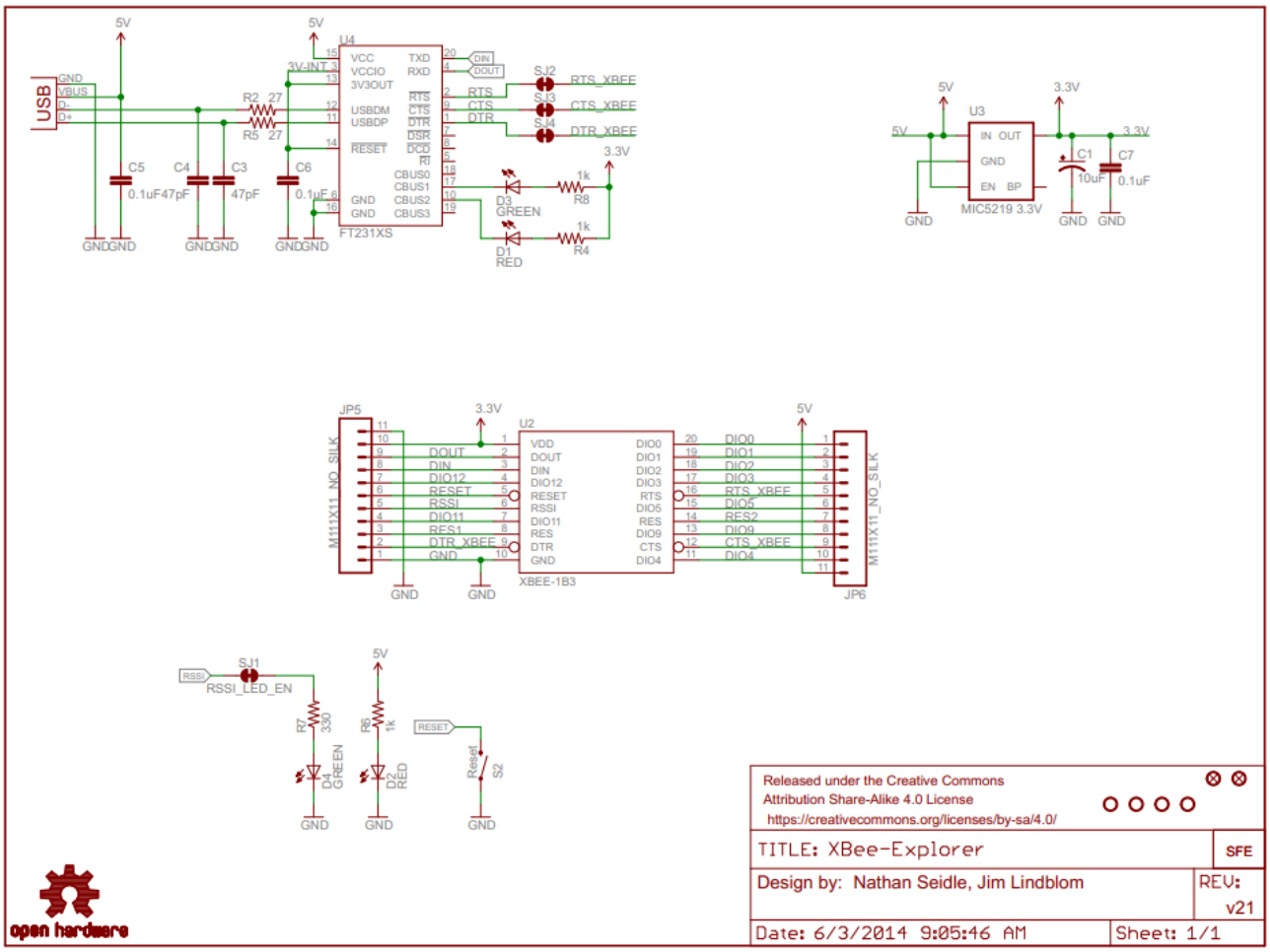


Figure II.5: Complete electrical diagram of the XBee adapter proposed by the constructor

We decided to simplify the schematic II.5 on our own and reduce it to only what we needed in order to have a clear documentation for the client. Figure II.6 represents all pins used to connect the XBee module to its adapter card and to the Gateway USB port. There is a USB connection between the Gateway and the adapter and a serial connection between the adapter and the XBee module.

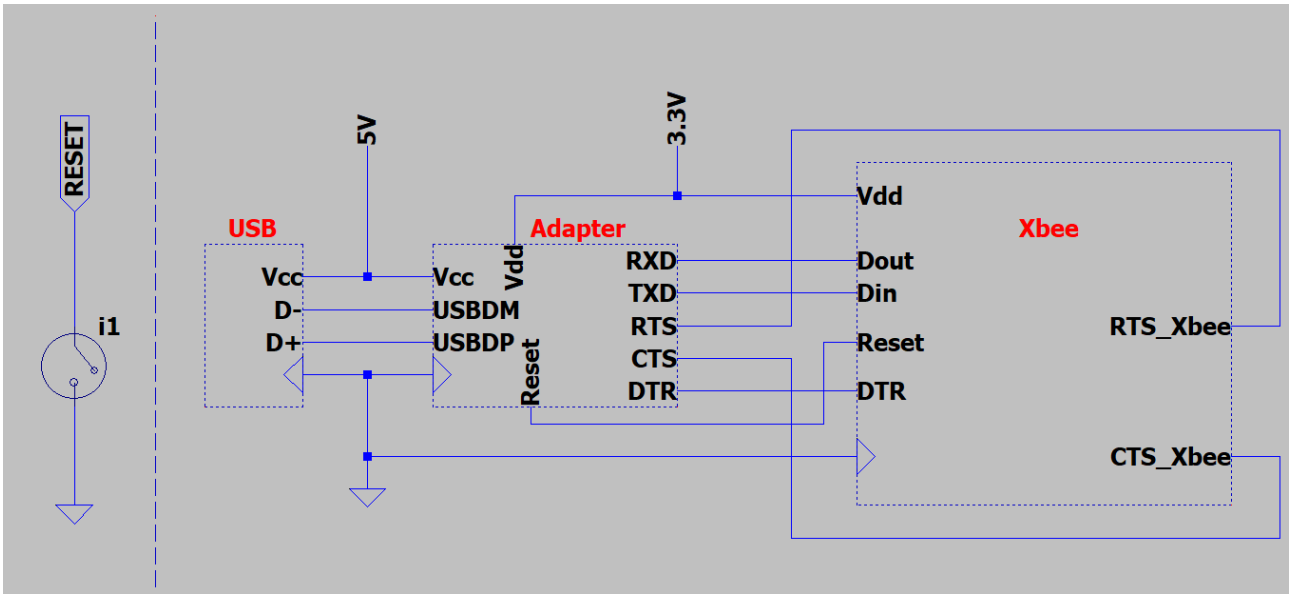


Figure II.6: Simplified electrical diagram of the XBee module, adapter and USB link package

2. Software Conception

This part illustrates all the conception work we have done for the software. Unlike the hardware part, we were free to design the software as we thought best.
We decided to use object-oriented conception. Oriented-object conception consists of defining interactions between software layers called "objects". Each object corresponds to a physical entity. The objective is to define objects' structures, behaviours and interactions.
We found two main advantages to using this method. First, we wanted to develop the software around the Agile method. As we are a team of six people, we agreed that it would be more efficient. Moreover as we are developing a proof of concept, we also took the scalability of our prototype into account, and this method was the most suitable for potential evolutions.

2.1. Use case diagram for the Gateway

The diagram below shows the different communication possibilities between each hardware element of the test station. The principal element is the ESAO workstation. It can send commands through the gateway which responds with either a message or an error. The gateway also provides a service to the test box element. If the gateway receives a command the test box is able to retrieve it and provide an answer. The third user (actor) is the gateway administrator who needs to access the system logs and update the database for parsing.

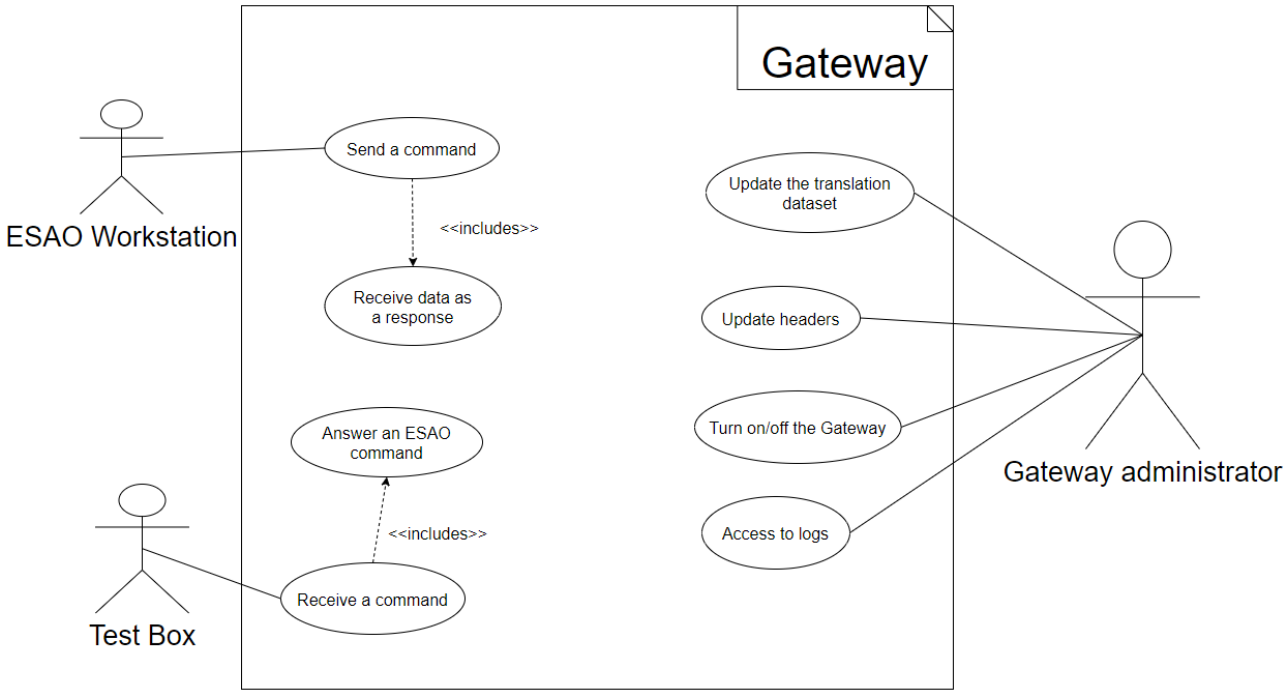


Figure II.7: Use case diagram

2.2. Interaction diagram for the Gateway system

The following figure shows the sequence of interactions between the test box and the ESAO through a software control entity of the gateway. First we have the reception step of an ESAO instruction and emission to the test box. We then have the reception of a response frame from the test box and emission to the ESAO.

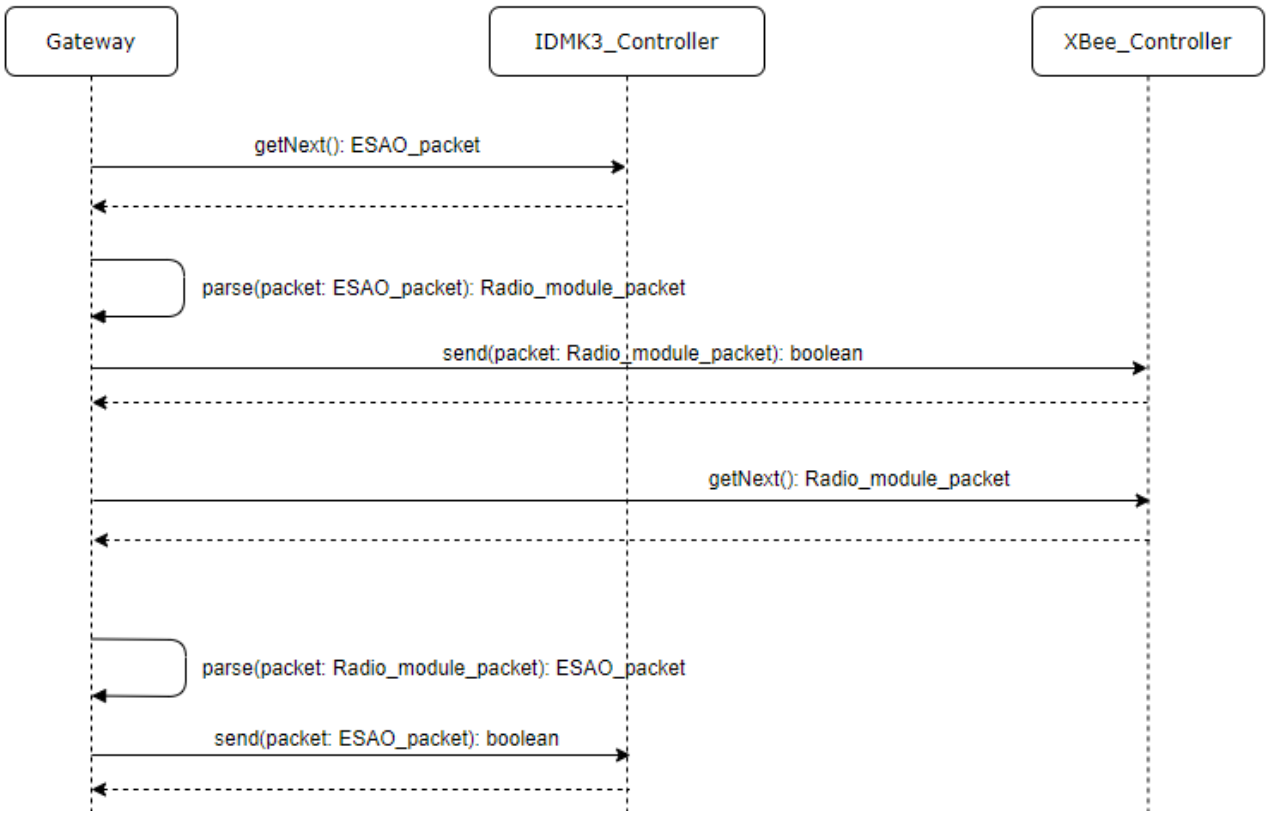


Figure II.8: Interactions diagram for the Gateway with the ESAO and the test box

The Gateway retrieves a packet in the **IDMK3_controller**'s buffer with the **getNext()** method. The **IDMK3_controller** returns an **ESAO_packet**. The Gateway translates the ESAO packet into a **Radio_module_packet** thanks to the **parse()** method. The Gateway sends the packet to the **XBee_controller** with the **sendPacket()** method which returns "false" if an error occurs.

The Gateway retrieves a packet in the **XBee_controller**'s buffer with the **getNext()** method. The **XBee_controller** returns a **Radio_module_packet**. The Gateway translates the XBee packet into an **ESAO_packet** thanks to the **parse()** method. The Gateway sends the packet to the **IDMK3_controller** with the **sendPacket()** method which returns "false" if an error occurs.

2.3. Class diagram of the Gateway system

The following figure shows all the classes of the gateway software application that are implemented, with any dependencies between them and their interactions.

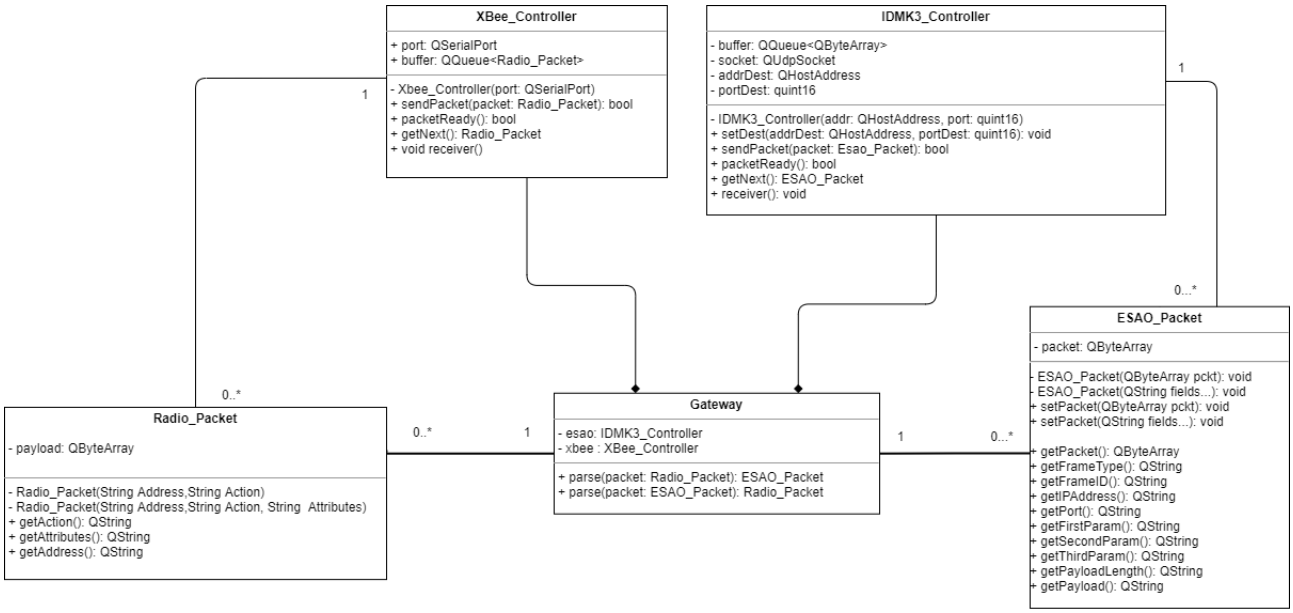


Figure II.9: Class diagram for the system

Conception hypothesis:

- The ESAO workstation communicates with only one test box which means that the message parsing and sending does not need an address system.
- Parsing rules are stored in an external database in order to avoid having to change the gateway source code any time instructions are added.

Communication interfaces:

We decided to separate the communication methods of the two interfaces as they are different. This also allows for better system scalability. If a communication interface is changed, modifications will be contained within a class and won't affect the rest of the code.

Classes **Radio_Packet** and **ESAO_Packet** describe packets coming from the corresponding interfaces that are described by the classes **XBee_Controller** and **IDMK3_Controller**.

XBee_Controller methods:

- **sendPacket(packet: Radio_Packet): bool**: This method sends an XBee packet over the XBee interface and returns "false" if there is an error
- **getNext(): Radio_Packet**: This method retrieves the first XBee packet in the controller's packet buffer
- **receiver(): void**: This method receives sent packets from the test box side's Xbee radio interface

IDMK3_Controller methods:

- **sendPacket(packet: ESAO_Packet): boolean**: This method sends an IDMK3 packet over the IDMK3 interface and returns "false" if there is an error

- **getNext(): ESAO_Packet**: This method retrieves the first IDMK3 packet of the controller's packet buffer
- **receiver(): void**: This method receives sent packet from the ESAO side's IDMK3 interface

The main class is **Gateway**. It is comprised of two controller instances **XBee_controller** and **IDMK3_controller**.

Gateway methods:

- **parse(packet: Radio_Packet): ESAO_Packet**: This method retrieves the received XBee packet, parses it, translates it to the corresponding ESAO packet and then re-encapsulates it in order to send it to the ESAO.
- **parse(packet: ESAO_Packet): Radio_Packet**: This method retrieves the received IDMK3 packet, parses it, translates it to the corresponding XBee packet and then re-encapsulates it in order to send it to the test box.

2.4. Database for command translation

In order to retrieve the corresponding command codes between the ESAO and the test box, we need a table to save all the equivalences. We chose to implement an SQL database because of its ease of use and modification. To this day, we are still working out details with Sterela so the database is not implemented, but the architecture is set. We will have to implement C++ methods to retrieve all the information we need. Moreover, other methods will allow easy updating of the database by passing arguments which have to be added or deleted from it. The QtSQL module will allow us to implement this database and its interactions.

The following figure (II.10) is a class diagram that represents link between data in the SQL database.

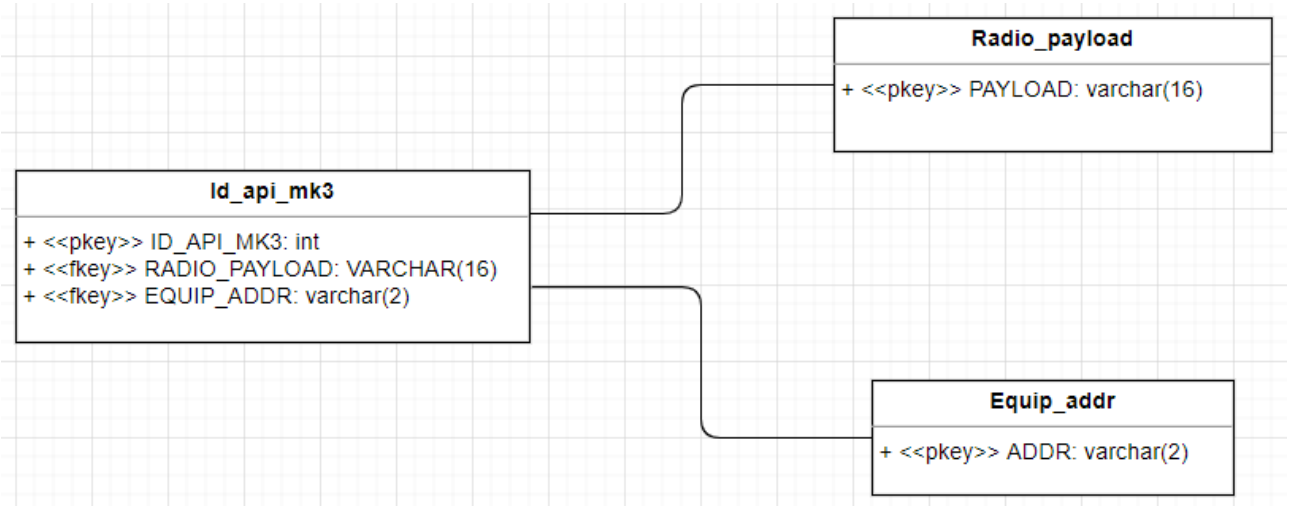


Figure II.10: Database schematic

However the current system does not allow a one-to-one correspondence between ESAO and Test Box commands. In some case, multiple ESAO commands can request information

that correspond to multiple fields of a single Test Box command. In this way it's impossible to develop a scalable method in order to bypass this constraint without treating all of these uneven correspondences individually. This problem is currently in the hand of STERELA. We are currently working with Sterela on reworking Test Box commands to keep a one-to-one correspondence with ESAO commands.

III

Development

This section develops all the technical parts of the project. It mainly contains the software methods analysis and functionalities regarding the conception specifications and the clients restrictions.

First of all and as we mentioned in the conception part, for this proof of concept STERELA has decided that the gateway will communicate with only one test box. Therefore in our case, the destination address is always the same. That is why we configure the module once using the XCTU program. We configure the network address, network role, packets static destination address and mode. We have chosen the Transparent mode for its simplicity. It is adapted to communication between two nodes. If STERELA decides to keep the XBee technology on the production stage, the **XBee_mode** must be changed to **API** mode. In this mode the module can be configured dynamically while the program runs and will be able to communicate to different end-points. However, **API** mode is more complex to use.

1. Qt Framework

It is important to define the development method. As we were working in sub-teams, we defined a set of standards in order to limit problems and gain time on the integration part of the project. In this way, STERELA suggested **Qt** to us. This platform includes an IDE and implements several frameworks which aim to simplify variable manipulations, and also provide us with useful libraries for serial and UDP communication. Moreover STERELA is already using this platform for their projects so using it would avoid potential difficulties like understanding the code and software maintenance in the future.

2. Radio_Packet class

The following figure (III.1) shows the packet structure of the communication between our gateway and the test box. This communication protocol was designed by STERELA. The information we found relevant to a radio module packet are the address, action and attributes. We can recompute all the other fields if necessary. That is why our **Radio_Packet** class has only 2 attributes (an address and a payload field which group action and attribute values).
The constructor for this class also only needs these fields.
We also prepare the getters and setters for these variables.

PREFIXE				PAYLOAD		SUFFIXE	
START	INFO TRAME			TRAME		STOP	CONTRÔLE
8 bit	*3 bit*	*13 bit*	*8 bit*	*8 bit*	*N octets*	*8 bit*	*16 bit*
STX	**VERSION**	**TAILLE**	**ADRESSE**	**ACTION**	**ATTRIBUT**	**ETX**	**CRC16**
02h	XXXX		XX	XX	XX	03h	XXXX

TAILLE

CRC16

Description des champs :

STX :	Octet 02h de début de trame.
VERSION :	Numéro de version du protocole de 0 à 7.
TAILLE :	Nombre d'octet contenu dans la trame (Champ 'action' + n 'attribut').
ADRESSE :	Adresse du boitier de test.
ACTION :	Indique la commande à exécuter.
ATTRIBUT :	Octet facultatif (Voir descriptif ci-après).
ETX :	Octet 03h de fin de trame.
CRC16 :	CRC16 calculé sur tous les octets compris entre STX et ETX. La table appliquée est issue du CRC-CCITT (kermit) disponible en annexe.

Figure III.1: Packet structure of the communication between the gateway and the test box

3. XBee_Controller class

XBee_Controller is responsible for the XBee module communication. This class only has two attributes. The serial port for the communication with the XBee module and the queue where we store received packets.

It has a constructor which requires an instantiated **QSerialPort** object from the Qt module. The constructor configures and opens the serial port.

The **sendPacket** method takes a **Radio_Packet** as input. To begin it checks if the port is open. It then takes the address and payload information of the given **Radio_Packet**, formats an array of bytes according to the structure defined by STERELA (see figure in **Radio_Packet** class part), and sends it to the text box. If in future, if STERELA decides the gateway will communicate with many test boxes, this method must be adapted to take that into account.

The **receiver** method is the body for our receiver thread. This thread has two roles. Its first role is to retrieve all packets from the XBee serial port. To do so, this thread takes each received byte one by one and checks for the start bits. Once it gets the start bits (defined in STERELA's communication protocol), it retrieves each of the given fields of the packet (version length, address, payload using length, end bits, and CRC), checks if the CRC is valid and adds it to the received packets queue. A CRC is a series of bits which defines the content of the packet. Given a packet, we get its content, compute its CRC (or Checksum), and compare it to the received CRC. If the two values are the same, the packet has been received correctly, otherwise it means it has been altered. The second role of this thread is to add correct packets to the class' queue.
This thread can without a doubt be optimized. We went for functionality over optimality.

As mentioned previously, after this proof of concept, this thread must be modified to be able to retrieve packets from multiple users as the structure of the frame won't be the same.

The **getNext** method retrieves the next packet from the queue only after having verified that such a packet exist and was well received using the method **packetReady** meaning the queue is not empty.

4. Radio_Packet and XBee_Controller tests

To test those two classes, we send packets to our own module from our own module (source address = destination address). Given an action and attributes, we are able to reconstruct the whole packet correctly. This packet is unpacked well by our receiver thread. We also have a meeting with the client to test out the communication with their test box.

5. ESAO_Packet class

The figures below (III.2 and III.3) show the packet structure of the communication between the gateway and the ESAO workstation. We use the IDMK3 communication protocol designed by Airbus.

The information we found relevant to the ESAO packet is actually every field as their content let us identify the command being sent so that we can respond with the correct answer. That is why our **ESAO_Packet** class has only a packet attribute as it is a array of bytes and has one method to retrieve each field. The constructor of the class helps recreating the packet to be sent from a already completed array of bytes or by giving it each field making up the packet. We also implement the getters and setters for these variables.

DS1

Field 1	ASCII	1 byte	command
Field 2	ASCII	3 bytes	sub command

DS2

Field 1	Integer	4 bytes	Home @IP
Field 2	Integer	2 bytes	Home UDP port

DS3

Field 1	Integer	4 bytes	Discrete, Analog, A429 bus, CAN bus, IP@ of receiver, begin memory @ for dump, specific test function
Field 2	Integer	4 bytes	A429 label, CAN identifier, SAP or AFDX port identifier, end memory @ for dump, specific test sub function
Field 3	Boolean	2 bytes	Options : Echo mode* for T100, T150, T200, T250, and T300 commands, loopback generic for analog, discrete, A429, CAN or all inputs

*:The echo mode is used to retrieve by the test system (ESAO workstation in previous chart) the **current value** of an LRU output, for commands T100, T150, T200, T250, and T300, **without modification of this value**.

DS4

Field 1,2	Variable length opaque data	Password, hardware or software P/N, boolean or analogue value, A429 word, CAN or AFDX Payload, memory data for dump, parameter list for specific tests… These data shall not be converted by an algorithm. They correspond to a bit string.

Figure III.2: Information of the different fields of an IDMK3 frame

H (Hello)

	DS1		DS2		DS3			DS4	
type of data	ASCII		Integer	Integer	Integer	Integer	Boolean	Variable length opaque data	
Length (in bytes)	1	3	4	2	4	4	2	2	4
Hello (command to LRU)	H	H00	Home IP@	Home UDP port (50300)	0	0	0	Length	<password>
OK for Hello (response from LRU)	O	H00	Home IP@	Home UDP port (50300)	0	0	0	Length	0
Error for Hello (response from LRU)	E	H00	Home IP@	Home UDP port (50300)	0	0	0	Length	0

Figure III.3: Packet structure of the communication between the gateway and the ESAO

Regarding the setters:

- The **setPacket** method allows us to build the packet using all the fields from **DS1** to **DS4**

or from one already formed ESAO packet. They take into account headers parameters of the frame.

Regarding the getters:

- The **getPacket** method returns the current packet
- Getters which return each specific field of the packet:
  - The **getFrameType** returns the command type
  - The **getFrameID** returns the sub-command id
  - The **getIPAddress** returns the IP address of the sender
  - The **getPort** returns the port used by the sender
  - The **getFirstParam** returns the field 1 of DS3
  - The **getSecondParam** returns the field 2 of DS3
  - The **getThirdParam** returns the field 3 of DS3
  - The **getPayloadLength** returns the length of the payload
  - The **getPayload** returns the payload

6. IDMK3_Controller class

IDMK3_Controller is responsible for the IDMK3 communication. This class has four attributes. The socket which receives the incoming traffic, the address and port destination of our messages and the queue where we store the received packets.

It has a constructor which requires an instantiated **QObject** from the Qt module. The constructor takes the IP address and the port, configures and opens the UDP socket to listen on the given port. We also implement the getters and setters for these variables.

The **sendPacket** method takes an ESAO_Packet as input then sends the already formatted **QByteArray** attribute of the given ESAO_Packet (headers + payload). It is more of a response method as it uses the IP address and the port of the sender of the last received packet for destination. This implementation works well in our case as the one to initiate the communication with our gateway is always the ESAO and one gateway is only linked to one ESAO.

The **receiver** method retrieves the incoming UDP packet from the ESAO to our socket defined by the IP address and the defined port. As Airbus uses a fixed routing table, our gateway is 172.30.44.1:8888. Each reception retrieves all pending packets. Thus the packets are stored in a queue with the first packet being processed at one time each

The **getNext** method returns the current packet to be processed taking into account the FIFO mode only after having verified that such a packet exist and was well received using the method **packetReady** meaning the queue is not empty.

7. ESAO_Packet and IDMK3_Controller tests

To test those two classes, we send packets to our own module from our own module (source address = destination address) then send an answer if, given each field, we are able to retrieve the corresponding and wanted field in the correct format. The reconstruction of the whole packet is therefore also correct in our own system. We also have a meeting with the client to test out the communication with the ESAO to insure that the format is correct and that both systems understand each other.

8. Gateway class

As defined in the conception part, the **Gateway** class is the real core of our project. It is the one responsible for translating one packet coming from the ESAO in the format defined by Airbus using the IDKM3 protocol into a packet to be sent to the test box by Xbee in the format defined by STERELA then doing it the other way to provide the answer of the test box to the ESAO. This is done by the methods named **parse** which return the corresponding packet to be sent depending on the given packet type.

One of the requirements of our project (even though it is only a proof of concept) is to have it be scalable. Therefore, the ability to increase the number of commands received from the ESAO and then sending the correct command to the test box to return the correct answer is critical in our implementation. Furthermore, the ability to only change the data part and not the source code of our gateway for each upgrade and modification is also heavily valued. This is why we chose to use an external database which is more user friendly to implement the different commands.

This database is designed to be easy to use such that every command is written in a string style regardless of its actual format in their respective protocol. Therefore, even though both communication protocols use byte arras, the abstraction task is given to the communication modules which are in our case the **Xbee_Controller** and **IDMK3_Controller**. So the packet translation methods find the correct values for each field in both format following the implemented database.

However, due to the many differences between the commands and responses format of Airbus and STERELA, we could not implement a usable database to make the correspondence from Airbus to STERELA at the time of our first prototype test.

9. Integration tests

Approaching the project deadline, we had the opportunity to visit Airbus campus at Saint-Martin-du-Touch near Toulouse in order to realize different tests of the first version of the system.

9.1. Material used to conduct tests

- 1 PC running on Linux which was simulating one part of gateway to send commands. The same program also permitted to display the payload of the test box answer.
- 1 PC running on Linux which was simulating one part of gateway to receive commands from the ESAO and then answering it following the given command. The same program also permitted to display the received and sent packets.
- 1 modified A33EPSE board (part of the test box) brought by STERELA which permitted to receive command, process it and send the corresponding answer.
- 2 radio modules Digi XB24CDM

9.2. First test

Purpose

The purpose of this test was to test the communication between the gateway and the test box.

Procedure

The communication was set up using the XB24CDM modules. Commands sent by the PC were well received by the test box (III.4) which gave the expected answers. Moreover, answers were well received by the PC and displayed.

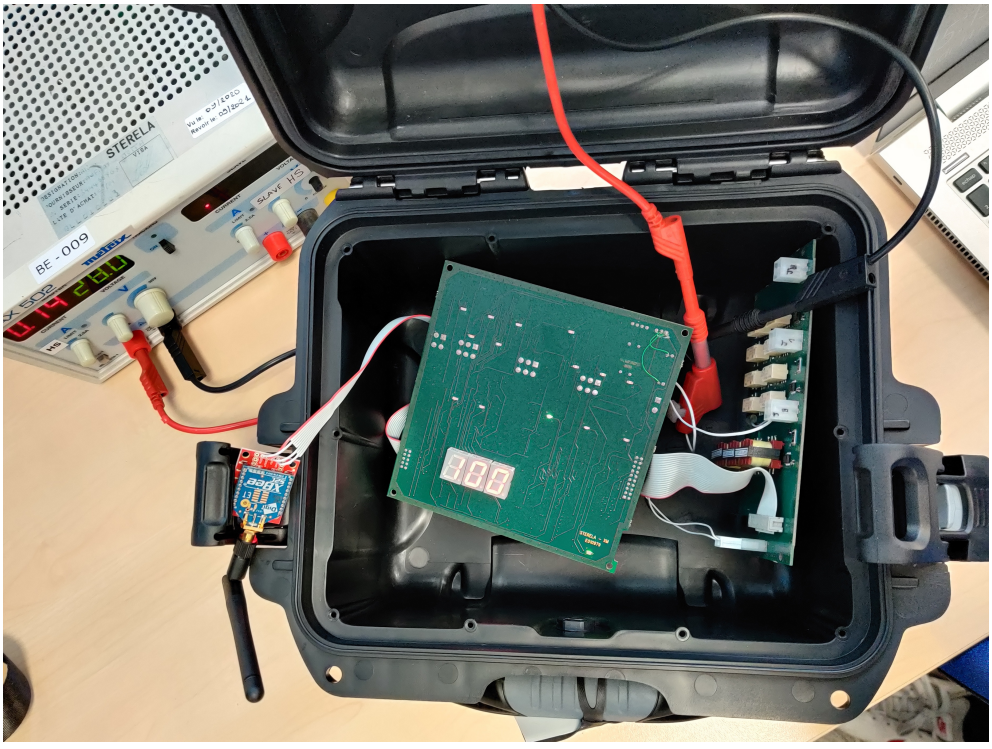


Figure III.4: Picture of the test box during the tests session

Conclusion

We can conclude that the first test was very convincing and the code's first version was conclusive.

9.3. Second test

Purpose

The purpose of this test was to test the communication between the gateway and the ESAO.

Procedure

We sent commands to the PC from the ESAO (III.5) using a graphical interface. Depending on the received commands, the PC was able to send back the expected answer in a comprehensible format for the ESAO which was also able to display it.

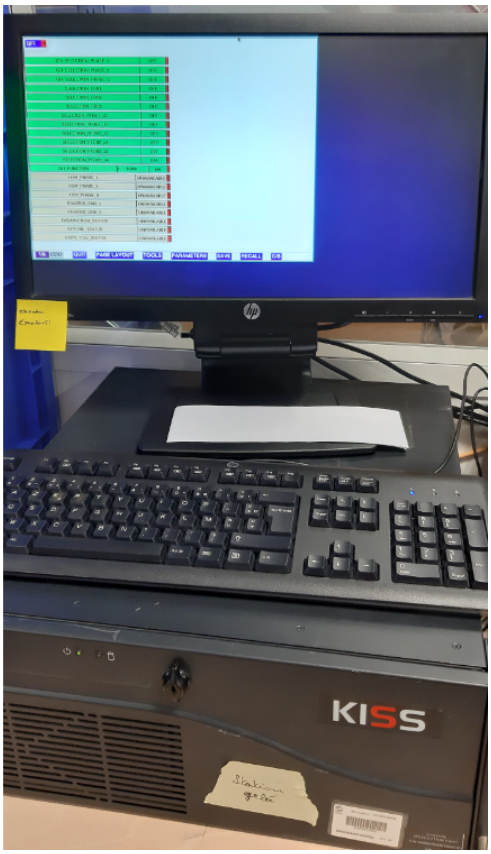


Figure III.5: Picture of the ESAO workstation computer during the tests session

Conclusion

We can conclude that the second test was very convincing and the code's first version was conclusive.

9.4. Third test

Purpose

The purpose of this test was to test reliability of the Digi radio module communication regarding the distance.

Procedure

Similarly to the first test procedure, we set up the communication between gateway and test box in order to exchange message and make the distance between them fluctuate. The messages sending were realized from different locations inside the airplane construction site.

Conclusion

Those tests observed that the link between the gateway and the test box is highly reliable but the latency between sending and receiving could be unsatisfactory as it could reach 2 seconds. However, the distance between the emitter and receiver were not a significant factor in latency values.
Complementary tests will allow us to understand this latency source (radio module, PC processing). It's important to note that the test box is increasing the latency because of the necessary tests it executes each it receives a command (resistance measurement, phase order test).

9.5. Tests conclusion

These real-conditions tests allowed us to verify and validate the communication between each part of the final proof of concept. Further tests will be scheduled in order to make the gateway mediate the communicate between the ESAO and the test box when the database is completed. Moreover STERELA will have to provide us with the hardware for the gateway.

IV Conclusion

This project gave us the opportunity of participating in a real industrial project with a direct impact for our client. We could put all of our software, communications, network and project management skills that we had acquired through the last years at INSA into practice.

Having a multi-background team enabled the different design and development tasks to be shared effectively. During the software conception and development, we mainly focused on maintainability and scalability which are important factors to take into account for a proof-of-concept, because STERELA will probably proceed to update the commands/responses association between the ESAO workstation and the test box.

The real tests made at Airbus allowed us to validate the robustness of the communication between the test box and our gateway but also the communication between the ESAO workstation and our gateway. We lacked the ability to test the entire device, mainly the parsing part of the gateway and its role as a communication intermediary between the two entities (ESAO workstation and test box). The final test with the whole architecture is to be done the week following our presentation.

If STERELA and Airbus decide to improve or add some new functionalities to our gateway, we hope that they will ask future ISS students to continue the work we started and keep this fruitful partnership alive.

List of Figures