Internship report

Development Consultant for a Transport Enterprise

Student:
Fabião Giserman Luiz

Centrale Lille Tutor:
Mr. Bordeaud'huy Thomas
UFRJ Tutor:
Mr. Costa Luís

November 9, 2023

1 Abstract

In this report, I firstly talk about my expectations and objectives before joining the company as an end of Studies Intern at Ecole Centrale de Lille and Universidade Federal do Rio de Janeiro. By describing the company and the projects I made part of, I put in perspective both the technical and social relational aspects of an ancient industrial project of national scale. I relate my engineering studies to the day to day needs as a software engineer and analyse the level of liberty that was given to me as an intern for decision making. All the acquired experience and elements during my work are also described in the later sections of the document, as well as how they shaped my prospects for after my graduation.

2 Acknowledgements

I thank Brazil's higher public education for being of the highest quality, totally free and open for everyone, in the hopes that it can one day reach the most inaccessible citizens.

I thank CAPES (*Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*) for the scholarship and opportunity given to me for my double degree program at Ecole Centrale de Lille.

I thank all of my professors from Universidade Federal do Rio de Janeiro and Ecole Centrale de Lille for the wonderful role they have in our society and, specially, for the role they had in my development both as an engineer and as a person.

I thank Monica Fabião, Cesar Giserman, Paulo Giserman and André Giserman for always supporting my decisions, my formation as a citizen and for looking out for my well being and happiness, above all.

I thank Angela Deiro for all the time and care she consacrated me.

I thank Jean-Jacques Le Gall, for the continuous unprecedented support during these two years in France, and for the years to come.

I thank my tutors Mr. Luís Costa, Mr. Thomas Bordeaudh'huy, for all the help received during my time as a student.

Contents

3 Presentation of the company

The company, which I will not disclose, is a french Digital Services Entreprise.

It is divided into 3 major fields:

- Integration
- Consulting
- Cybersecurity

Within the Integration field, of which I was a part of, there are different ramifications such as transport, banking and insurance.

Amongst the clients in transport, we identify the one I worked for, which is responsible for a significant amount of transport in the national and international territory

4 Analysis of the objectives for the internship

To my understanding, the end of studies internship is to learn how to adapt all of the knowledge and experience acquired during my studies into an enterpreneurial environment for real life needs and projects. It also serves as a way of experiencing the engineering profession and making my own conception of how it can serve our society. It also gives me an understanding of the other professionals in the market, how decisions are operated and how enterprises are organised.

From another point of view, I had the objective of assuring and developping my practical and relational competences. I expected to learn new technologies, methodologies and to gradually feel more comfortable with my day to day work. I hoped to understand my place and how I'm expected to communicate with my colleagues and clients to achieve our goals.

5 Initial contract definition

Initially, I was supposed to join the billing project, that develops embedded systems for the client's identification and payment systems. Shortly before the start of my internship, I was called to ask if I could join Godec instead, described in section 6.1. The terms were that I would still be working with the C language as a developer.

6 Projects

6.1 Godec

6.1.1 Presentation of the project

6.1.1.1 General view

The main project of which I made part of is called GODEC. It's objective is to acquire real time and theoretical transport data, manage incidents and real time events and propagate them throughout the other applications of the client's network. Our project is composed of four main applications, listed below:

- Bannec;
- Concentrator;
- Aquisition;
- Post;

In general terms, Bannec is responsible for the treatment and transfer of all theoretical transport circulations to come, describing the expected time of departure, passage and arrival of the transport along its course.

Concentrator is an application with multiple instances. Each instance is assigned to a major region in France to automatically collect data from the transport's infrastructure as it advances in it's route. Unfortunately, not all transports have the infrastructure to automatically communicate, so there is still some significant manual work to be done daily. The data collected by the Concentrators are then sent to Acquisition, for further treatment.

Acquisition is the core of the project. It receives all of the theoretical, real time automatical and manual data of the trajectories. It then manages the transport's passages, incidents, events and delays. It completes its mission by forwarding messages with the collected information to other important services such as *Traveller Information* - which informs real time delays to stops and travellers - and foreign entities, when the route includes international travel.

Post is a graphical user interface in which the transpot's inspectors can see informations about circulations, manually enter real time data about them and create incidents and events.

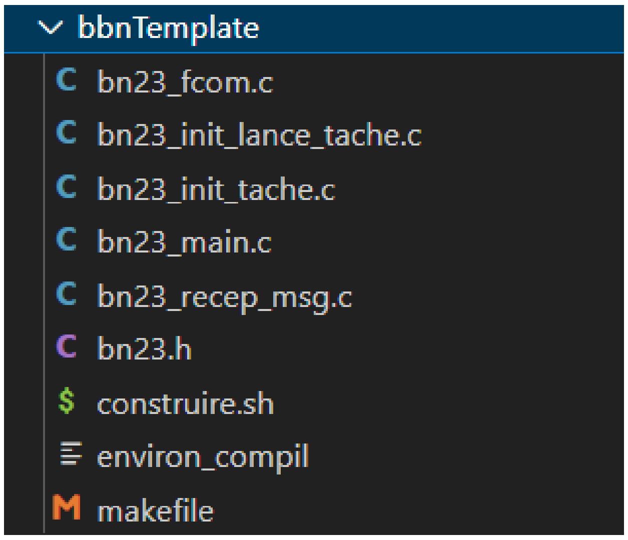
6.1.2 Products

6.1.2.1 Godec Acquisition

Godec Acquisition is the heart of the application. It is composed of different processes called GOXX, being XX their ID. Each process has a specific function and communicates with each other by an old protocol proprietary to the client. This protocol implements non secure socket communications and message serialization.

The communications with the database are performed in layers. The process GOXX calls a library of functions that serves as an interface. It then calls the functions that are SC programs converted to C by Ingres from the SC (Ingres Embedded SQL/C source code) files. The database is an Ingres instance. There are no constraints and no direct association for foreign keys in place. Looking for relations between tables is a difficult task as it depends on a non updated word document.

A GOXX implementation has a template to be followed, illustrated by Figure 1.



```
bbnTemplate
  bn23_fcom.c
  bn23_init_lance_tache.c
  bn23_init_tache.c
  bn23_main.c
  bn23_recep_msg.c
  bn23.h
  construire.sh
  environ_compil
  makefile
```

Figure 1: BGOXX Template.

The file *fcom* implements communication with other processes via that protocol. One example is the message we send to the process that is responsible of all of the log management.

The file *recep_msg* implements functions that are callbacks to the reception of these protocol's messages. In these functions, we verify their headers and send them to their proper treatment within the program.

The file *init_tache* is responsible of collecting the parameters file and charging its content before any other treatment starts.

The *main* file ties the other ones together and implements a *while* loop to listen to incoming messages of that protocol.

To communicate with external applications within the network we operate with two protocols: CFT and MQSeries.

The Cross File Transfer (CFT) protocol is used to transfer files between two servers and offers security, acknowledgement, integrity and error logging. It is used, for example, to transfer files with the theoretical transport information from Bannec to GODEC.

On the other hand, MQSeries is a service developped by IBM which consists of a MQServer to which a task A can depose a message and that another task B can retrieve it. It implements security, integrity and can be configured for multiple queues, connecting multiple senders and receivers. It can also function in a publish/subscribe mode.

The task GO20 treats theoretical transport information and registers them in our Ingres database. GO12 gathers the collected data and determines which points in the trajectories requires a human intervention to indicate it's real time passage. GO14 will analyse these points in order to attach a responsible entity/inspector to them. GO18 diffuses the respective messages to the different Godec Posts.

GO4 receives observations, incidents and circulation events created by Godec Post's interface. GO5 treats the observations and calculates in real time the difference between the theoretical and real transport's passage times (EH), as well as it's variations (VEH). GO19 will generate a message ID for every *object* that is to be forwarded (observations, incidents). GO50 and GO3 will forward the messages that are destined to external projects with MQSeries.

6.1.2.2 Godec Post

Godec Post is a graphical user interface (GUI) made in Uniface for the use of transport administrators and inspectors. In this interface, they can search and follow circulations, create incidents and events and inform the real time transports' passages within their different geographical points of interest.

6.1.2.3 Godec Concentrator

Some transports are equipped with an embedded system that detects it's passage in a point of interest, like bus stops, with the help of on the ground infrastructure. When this happens, we can determine when the transport passed at that location, which prevents a human inspector from observing it and informing it manually. This is what we call an automatic observation. The superconcentrators are regional servers that are responsible of listening and forwarding these messages via a SAAT connection. They are then receptioned by GODEC TI and Godec Acquisition. There are 19 concentrators in total in order to balance the charge between all circulations that passes through the french territory.

6.1.2.4 Godec Bannec

Theoretical circulation data is sent to us in the format of text files by Bannec - another project in the client's network. Godec Bannec treats the raw data and forwards it to Acquisition so that it can be charged in our Ingres database.

6.1.3 The team

Our team is composed of 7 persons, divided in the following roles:

- 1 C Developper
- 1 Uniface Developper
- 1 Business Analyst Intern
- 2 Technical Responsibles
- 1 C Intern Developper (me)
- 1 Project Responsible

6.1.4 Evolutions

As the application is ancien and is also made of multiple technologies known to be obsolote in the current days, there has been some studies towards new solutions to update it.

One of the focuses of these studies is to decommissionate Godec POST, which is very complex, not well documented and implemented in Uniface.

Another point is to centralize and better implement the functionalities within Godec Bannec and concentrators, which gave life to a new application called Super Concentrators.

6.1.4.1 Logoden Web Service

As a solution to decomissionate Godec POST, the Logoden web service was proposed to implement the creation of incidents and circulation events of type “circulation deletion”. The service was developped in java and takes an HTTP request as its input. I participated in its initial qualification tests and in the evolutions that followed. The methodology for testing and the relevant problems encountered are described in section 6.1.5.3.

6.1.4.2 Super Concentrators

Super concentrators (SCs) is an evolution to replace a great part of functionalities of Godec Acquisition and their integration with external applications. The main target is to decomissionate GODEC TI, that forwards real time data to the traveller information project.

The SCs receive all of the messages sent by the existing concentrators and forwards them as messages 213 and 215 after treatment.

6.1.5 Tasks

6.1.5.1 Material Number sent NULL to an application

This was my first task on the project. It is a fairly simple one, compared to the following others, but it demanded a fair level of comprehension of the application’s code. Apparently, some observation messages that we diffuse to other applications had the field *Material Number* - used to indicate the source application in which the observation was created - set to NULL even though it had a non NULL value within our data base. It was also seen that whenever that was true, the field *Window Number* - to describe a set of remarkable points on which the transport passes - was also set to NULL, but in this case, NULL as well in our database. Through analysis of the message’s composition, I saw the following extract of code, that seemed to create the problem.

```
/* Si observation manuelle */
if (atol(s_msg_obs_ef->I_Num_Fen) == B_NUM_NULL)
{
   Bn3Journal(-402, NOMFONC, BAVARD_LOCAL,
   "Numero de fenetre null -> Num_fen: %s, Num_Mat_Mod: %s",
   s_msg_obs_ef->I_Num_Fen,
   s_msg_obs_ef->I_Num_Mat_Mod);
   strcpy(s_msg_obs_ef->I_Num_Fen, "");
   strcpy(s_msg_obs_ef->I_Num_Mat_Mod, "");
}
```

We can easily see that this sets *I_Num_Mat_Mod* to blank if *I_Num_Fen* is set to NULL.

This same extract of code occurred three times throughout the code for the conception of messages. I commented the lines like line 9 of the extract and pushed the changes to our GitLab project. I then transferred the changed files to the DEV DRF server, compiled it and sent it to the products directory, so that it could be executed. With the new messages that were being sent, I validated that the anomaly was gone.

6.1.5.2 Frontier Circulations

The client has transports that crosses the border in a single circulation. They can be either departing from France and arriving at foreign territory 'S' (*Sortie* or Exit) or the other way around 'E' (*Entrée* or Entry).

For the exiting circulations, at every point of interest, we send an observation to both our system and a common interface, where the other countries can recover the data. On the other hand, for the entry transportation, as soon as they step in French territory, the messages to the common interface stop being sent.

The communication is made by means of the message 2002 emitted by GO50. There is a variant of this message, called 2002 TIS, which works for both entry and exiting transports, so it seemed important to see how exactly their conception and diffusion differ.

In the code extract down below, the circulation to be sent as form of message 2002 is assigned to the pointer *circ_front*. The if in line 12 checks if the attribute *B_ Cod_ Frontiere* is of type 'ES' - *Entry and Exit*, since some transports enter and exit the French territory more than once, if they circulate close enough to the border. The else should treat all of the other cases, 'E' and 'S', but only the 'S' ones were getting recovered.

```
1   /* Recuperation de la circ. au point frontiere */
2   for (cpt=0; cpt <Lst_Circ_Front.Nb_Elt_Circ_Front; cpt++)
3   {
4       if (strncmp(clef->B_Cod_Circ_Ori,
5       (Lst_Circ_Front.Circ_Front)[cpt].B_Cod_Circ_Ori,
6       LG_NUM_CIRC-1)==0)
7       if (strncmp(clef->B_Dat_Hh_Ori,
8       (Lst_Circ_Front.Circ_Front)[cpt].B_Dat_Hh_Ori,
9       20)==0)
10      {/* trouve */
11
12          if (strcmp((Lst_Circ_Front.Circ_Front)[cpt].B_Cod_Frontiere,
13          B_IND_FRONT_ENTR_SORT) == 0)
14          {
15              if (strcmp((Lst_Circ_Front.Circ_Front)[cpt].B_Ind_Frontiere,
16              B_IND_FRONT_SORT) == 0)
17              {
18                  *circ_front = &(Lst_Circ_Front.Circ_Front[cpt]);
19                  break;
20              }
21              /* Au suivant */
22          }
```

```
        else
        {
            *circ_front = &(Lst_Circ_Front.Circ_Front[cpt]);
            break;
        }
    }
}
```

This *Lst_Circ_Front* that is being iterated in the for loop is a global linked list that is initalized in the start of GO50. The hypothesis here is that maybe the 'E' circulations are not in the list.

The list is created in the following function:

```
cod_retour = IadCircFrontEUouGILireSortantes(&(Lst_Circ_Front.Circ_Front),
&(Lst_Circ_Front.Nb_Elt_Circ_Front));
```

Which is defined in our functions' interface product: *biad/src/iad/iad_circ_front.c*

That references our functions' SQL product:

```
resultat = BdbCircFrontEUouGILireUniDir(B_IND_FRONT_SORT,
 s_circ_front, nb_elt);
```

In this function, we read all of the circulations in the table *B_CIRC_FRONT_EU_OU_GI*, but with a filter in the first argument as *B_IND_FRONT_SORT*, only reading the 'S' circulations.

After verifying that this function was not used elsewhere, I changed it's header and format, deleting the possibility of filtering, since we wanted all types of circulations to be taken into account.

To validate it, I deployed the application in our DEV server and filtered the log file updated by GO50 to verify the 2002 emissions for both types of circulations: 'E' and 'S'. This update was then deployed in production, a few months later.

Shortly after, we started to get complaints that some data was coming in as NULL to the common interface. After some more meticulous analysis, I realised only the circulations with 'E' were presenting this issue, which meant that the update didn't interfere with the service we already had, but only with what had been added (the 'E' circulations).

The problematic data was always the same field: *b_dat_hh_fr*, from the *B_CIRC_FRONT_EU_OU_GI* table, which is described in the code as the time the transport crosses the french border. There is no trace of this table in any of our documentation. The description in the code is possibly the right one, but we cannot be sure of it, as by analysing the code alone, it's functionality is not easily inferred.

I then ran a few SQL queries to further investigate. The queries indicated that there were no 'S' circulations with *b_dat_hh_fr* set to NULL, but plenty with 'E' set to NULL. It was complicated to tell if this beahvior was actually normal, since no one in the team seemed to know what that field in that table meant.

I then tracked the moment where this table is initially loaded, to possibly find out more if the data came that way from the source or if we set it to NULL within a condition in our code.

The data's point of entry is in Godec Acquisition, sent by Godec Bannec. In that point of entry, the date field was already widely set to NULL. Since Godec Bannec is still in our perimeter, I started to analyse it.

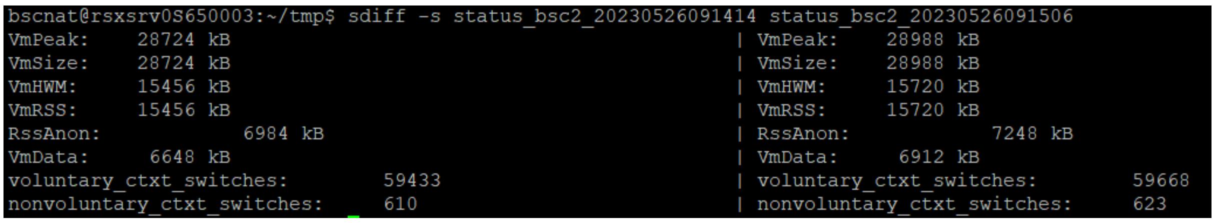
This was somewhat challenging because I had never looked at the Godec Banec's source code, nor anyone had explained me how it worked. With a lot of digging, I saw a simillar code in one of the Godec Acquisition task's source code, only with one difference: a condition with a comment saying *BAN-130*. After some research, I found out that *BAN-130* was a norm the team defined back in 2012 for this specific treatment. It was implemented in Acquisition, but not in Bannec. So I added it to Godec Bannec. I since haven't been able to test it as it is inside a code that does not run in the DEV server. My colleague was supposed to help analyse how to test it with me, but he went on holidays and I had to prioritise other subjects.

6.1.5.3 Memory Leak - Super Concentrator

One of the new applications that will lead the evolution within GODEC is the Super Concentrator. It is better described in section 6.1.4.2.

Within this newly developed application, the team realized that one of it's processes, *bsc2*, was leaking memory in our DEV environment. In just a few days it was responsible for taking 80% of the server's RAM.

We measured the difference in memory consumption of that process within a one minute interval, illustrated by Figure 2.



```
bscnat@rsxsrv0S650003:~/tmp$ sdiff -s status_bsc2_20230526091414 status_bsc2_20230526091506
VmPeak:    28724 kB                                          | VmPeak:    28988 kB
VmSize:    28724 kB                                          | VmSize:    28988 kB
VmHWM:     15456 kB                                          | VmHWM:     15720 kB
VmRSS:     15456 kB                                          | VmRSS:     15720 kB
RssAnon:            6984 kB                                  | RssAnon:            7248 kB
VmData:     6648 kB                                          | VmData:     6912 kB
voluntary_ctxt_switches:        59433                        | voluntary_ctxt_switches:        59668
nonvoluntary_ctxt_switches:     610                          | nonvoluntary_ctxt_switches:     623
```

Figure 2: Difference in memory consumption before the fix.

We can see that in just one minute, the VmSize grew 0.1% , which is very significant for an application that runs non-stop.

After a quick analysis, we identified some points in which the leak was clear, where we had a *malloc* that wasn't followed by a *free*.

This function allocates memory pointed by the *char* pointer *messageTraite*:

```c
// ############## Ajout/ecriture message dans fichier  ##############

char *bsc2_printableMessage(char *message) {

        int i = 0;
        int length = strlen(message);
        char *messageTraite = malloc(length + 1);

        if(messageTraite != NULL) {
                for (i = 0 ; i < length ; ++i)
                {
                        messageTraite[i] =
            (isprint(message[i]))? message[i] : '.';
                }
                messageTraite[length] = 0;
        }
        return messageTraite;
}
```

It then returns the pointer, leaving it to the user to free it. When used, the pointer was not freed.

```c
fprintf(inputFile, "%s\n", bsc2_printableMessage(message));
```

So we fixed it and added an error test for the allocation:

```c
tmpMsg = bsc2_printableMessage(message);

if (tmpMsg == NULL)
{
    bsc2_journal(0,NOMFONC,JNL_TRAC,
"Message revenue null de bsc2_printableMessage pour l'ecriture dans %s.",
filepathname);
    return (-1);
}

fprintf(inputFile, "%s\n", tmpMsg);

bsc2_journal(0, NOMFONC, JNL_TRAC,
"ajout du messsage %s dans le fichier %s",
tmpMsg, filepathname);

free(tmpMsg);
```

Within the composition of both messages that *bsc2* sends, the string that holds their formats was not being freed either, which was corrected by the addition of line 59 in the following code.

```
 1  if ((format =
 2       bsc2_FormatMessageGetFormat(b, msg->I_Typ_Msg, size_message))
 3       != NULL)
 4          {
 5          if(strcmp(msg->I_Typ_Msg, "213") == 0)
 6          {
 7              strncpy(numConc, (msg->I_Id_Emet)+3, LG_NUM_CONC);
 8              //recupere le num du concentrateur sur 2+1 caracteres
 9              numConc[LG_NUM_CONC-1] = '\0';
10  
11              result = briv_util_sprintf( format,
12              msg->I_Typ_Msg,
13              /* information volontairement supprime du message */
14              /*
15              msg->I_Id_Emet,
16              msg->I_Id_Recep,
17              */
18              msg->I_Num_Msg_Dif,
19              msg->I_Cod_Oper,
20              msg->I_Num_Circ,
21              msg->I_Cod_Ci,
22              msg->I_Cod_Ch,
23              msg->I_Horaire,
24              msg->I_Dat_Recep_Conc,
25              msg->I_Typ_Hor,
26              msg->I_Num_Eqp_Obs,
27              msg->I_Fenetre,
28              msg->I_Nat_Msg,
29              numConc ); //PNN : Ajout du num du contrateur
30              //msg->I_Id_Emet ); //PNN : Ajout du num du contrateur
31  
32  
33                          //*size_message = *size_message
34              //+ strlen(msg->I_Msg_Saat_Brut) + 1;
35                          *size_message = *size_message + 1;
36                  }
37                  else //Type 215
38          {
39                          result = briv_util_sprintf( format,
40              msg->I_Typ_Msg,
41              /* information volontairement supprime du message */
42              /*
```

```
            msg->I_Id_Emet,
            msg->I_Id_Recep,
            */
            msg->I_Num_Msg_Dif,
            msg->I_Cod_Oper,
            msg->I_Dat_Recep_Conc,
            msg->I_Nat_Msg,
            msg->I_Num_Eqp_Obs,
            msg->I_Fenetre,
            msg->I_Cod_Ci,
            msg->I_Cod_Ch,
            msg->I_Msg_Saat_Brut );

                        *size_message =
            *size_message + strlen(msg->I_Msg_Saat_Brut) + 1;
                }
                free(format); /*FREE ADDED*/
        }
```

With these adaptations, everything that was easily noticeable had been corrected. Nonetheless, the process was still leaking significantly. In order to further investigate the leaks, we were in need of a debugging tool.

Unfortunately, the integration environment didn't have one installed, and we did not have the permissions to install one. On the other hand, the development environment had *GDB* - GNU Debugger - but no reception flow to test the leak during the treatment of messages. With the help of a team member, we established the flow for the development environment.

We then established a procedure to find the leaks with GDB:

1. Save a copy of /proc/[PID_bsc2]/smaps as *smaps1*
2. Wait 10 minutes so that the leak is significant
3. Save a new updated copy of /proc/[PID_bsc2]/smaps as *smaps2*
4. Run the diff command with *smaps1* and *smaps2* to see which parts of the memory had grown
5. Use GDB on the process to dump the address pool that had grown
6. Analyze the content of the memory space that leaked to find what was not being freed
7. Correct the code

Within the output of this procedure, we identified lines as follows:

‘%d‘

'%d%s'

'%d%s%c'

That indicated that there was a problem with the conception of the message's format.

The function that was responsible of dynamically building the format is *bsc2_FormatMessageBuildFormat*, with the following content:

```c
char *bsc2_FormatMessageBuildFormat
(char *result, int fields_size[], int *size_message)
{
        if (result == NULL)
        {
                int i = 0;

                *size_message = 0;

                while (fields_size[i] >= 0)
                {
                        result =
            briv_util_sprintf("%s%%%ds", (result != NULL) ?
            result : "", fields_size[i] - 1);
                        *size_message =
            *size_message + (fields_size[i] - 1);
                        ++i;
                }
        }

        return (result);
}
```

The while loop iterates over all of the field types in the given message, so that it allocates their sizes as it goes. The problem there lies within *briv_util_sprintf*, which is an auxiliary function that returns a pointer for a memory space it allocates. As a consequence, the code was allocating - and not freeing - space at every while iteration. So, even though we added a *free(format)* in the previous correction, all of the space allocated to get to the final format was not freed.

To correct the leak, we had to restructure the way the formats were built. Since there are only two message types being treated in this task and, for the long term future, there will be no need to add a lot of different message types, there is no real reason to build this format dynamically.

I then defined some MACROS to hold the formats, as follows:

```c
//Format des messages - valeur des constantes -1
#define FORMAT_MESSAGE_213  "%3s%13s%1s%6s%6s%2s%6s%14s%1s%5s%8s%1s%2s"
```

```
/*LG_TYP_MSG,LG_ID_MAT,LG_ID_MAT,LG_NUM_MSG,
  LG_ETAT,LG_NUM_CIRC,LG_COD_CI,LG_COD_CH,
  LG_HH,LG_DAT_HH,LG_TYP_HOR,LG_ID_MAT,
  LG_NUM_FEN,LG_ETAT,LG_NUM_CONC */

#define FORMAT_MESSAGE_215        "%3s%13s%1s%6s%6s%2s%6s%14s%s"
/*LG_TYP_MSG,LG_ID_MAT,LG_ID_MAT,LG_NUM_MSG,LG_ETAT,
LG_DAT_HH,LG_ETAT2,LG_ID_MAT,LG_NUM_FEN,LG_COD_CI,
LG_COD_CH, LG_MAX_REC (taille variable max 65) */

#define TAILLE_MESSAGE_213                              68 + 1
#define TAILLE_MESSAGE_215_BASE                      51 + 1
```

The only complication is that the message 215 holds a field of variable size *LG_ MAX_ REC*, of range 0-65. This is why the last macro has the suffix **BASE**, as it doesn't hold the real size of the message.

We then transformed the following code:

```
char * bsc2_FormatMessageGetFormat
(bsc2 * b, char *typeMessage, int * size_message)
{
#ifdef NOMFONC
#undef NOMFONC
#endif
#define NOMFONC "bsc2_FormatMessageGetFormat"

        char * result = NULL;
        char * type_result       = NULL;

        int    type_size_message = 0;

    if(strcmp(typeMessage, "213") == 0)
        {
        int fields_size[] = {
          LG_TYP_MSG,/* Type de message*/
          /* information volontairement supprime du message */
          /*
          LG_ID_MAT,
          LG_ID_MAT,
          */
          LG_NUM_MSG,/* Identifiant BREHAT*/
          LG_ETAT,/* Code operation 1= creation*/
          LG_NUM_CIRC,
          LG_COD_CI,
          LG_COD_CH,
          LG_HH,        //PNN : Date observation SAAT
          LG_DAT_HH,/* Date heure de reception par le concentrateur*/
          LG_TYP_HOR,
          LG_ID_MAT,/* Numero de materiel localisateur*/
          LG_NUM_FEN,/* Numero de fenetre*/
          LG_ETAT,
          //LG_ID_MAT,//PNN : Ajout du champ num du
          //concentrateur, taille du champ num du concetrateur
          //= LG_ID_MAT (champ I_Id_Emet)
          LG_NUM_CONC,
          -1
        };

                type_result   =
        bsc2_FormatMessageBuildFormat
        (type_result,fields_size,&type_size_message);
        }
    else // Type 215
        {
        int fields_size[] = {
          LG_TYP_MSG,/* Type de message*/
          /* information volontairement supprime du message */
          /*
          LG_ID_MAT,
          LG_ID_MAT,
          */
          LG_NUM_MSG,/* Identifiant BREHAT*/
          LG_ETAT,/* Code operation 1= creation*/
          LG_DAT_HH,/* Date heure de reception par le concentrateur*/
          LG_ETAT2,/* Type message SAAT*/
          LG_ID_MAT,/* Numero de materiel localisateur*/
          LG_NUM_FEN,/* Numero de fenetre*/
          LG_COD_CI,/* CI du PR associ  la fenetre module I,S,R,N */
          LG_COD_CH,/* CH du PR associ  la fenetre module I,S,R,N */
          0,/*LG_MAX_REC,*//* Message Brut Saat*/
          -1
        };
                type_result   =
        bsc2_FormatMessageBuildFormat
        (type_result,fields_size,&type_size_message);
        }

        result        = type_result;
        *size_message = type_size_message;

        return(result);
}
```

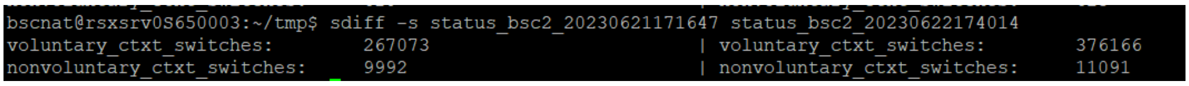
```
```

Into

```c
char * bsc2_FormatMessageGetFormat
(bsc2 * b, char *typeMessage, int * size_message)
{
#ifdef NOMFONC
#undef NOMFONC
#endif
#define NOMFONC "bsc2_FormatMessageGetFormat"

        char * result                    = NULL;
        char * type_result  = NULL;

    if(strcmp(typeMessage, "213") == 0)
        {
                result = FORMAT_MESSAGE_213;
                *size_message = TAILLE_MESSAGE_213;
        }
        else if(strcmp(typeMessage, "215") == 0)
        {
        result = FORMAT_MESSAGE_215;
        *size_message = TAILLE_MESSAGE_215_BASE;
        }
    else // Type 215
        {
        bsc2_journal(-10,NOMFONC,JNL_TECH,
        "Type de message non identifié : %s\n",
        typeMessage);
        }

        return(result);
}
```

The updated size of the messages 215 was calculated in the function that calls *bsc2_FormatMessageGetFormat*.

Within a 40 minutes interval, the *sdiff* command on the status of the process showed us that there was no growth at all in memory consumption, illustrated in Figure 3. Also, within 15 hours, the process treated 727 683 messages with no augmentation in memory consumption.



```
bscnat@rsxsrv0S650003:~/tmp$ sdiff -s status_bsc2_20230621171647 status_bsc2_20230622174014
voluntary_ctxt_switches:        267073                          | voluntary_ctxt_switches:        376166
nonvoluntary_ctxt_switches:     9992                            | nonvoluntary_ctxt_switches:     11091
```

Figure 3: Difference in memory consumption after the fix.

6.1.5.4 GO50 - Message recovery, delay and DB purge

The GO50 task is one of the two in GODEC Acquisition that forwards messages to the external projects, and is therefore of great importance. It had come to our attention, by various anomaly tickets that messages were sometimes not delivered after a crash. Normally, this task is made to recover any messages that were not acknowledged and resend them, as to guarantee no loss in communications.

We also received complaints about the delay in messages, which had started a few months before. Multiple conference calls were organized to look for the source of the problem, since the code in production had not changed in over a year. This task was created after the decommissioning of an old application, and therefore would sometimes use old database tables. These tables were not being purged and accumulated over 6 millions entries, which slowed down any query that tried to gather information from it.

For the recovery problem, I proceeded with a hypothesis and some actions to validate them.

Does the application restart from where it stopped?

1. After a manual restart
2. After a forced kill of the processes
3. After a disconnection from the MQ server to which the messages are forwarded to

To start testing these, I had to understand what was the mechanism set in place for the recovery. At every 10 messages, the application stores the last acknowledge message's ID (which is always chronologically incremented) in a table of our database. This way, if the service is stopped, it will take that number from the database and restart from there.

For the first and second tests, the recovery was working well. To be able to test the third one, I had to pass in bad parameters to the MQ framework, to simulate a disconnection from the server. Once this was done, I realized that the recovery was not working as it should, skipping a few messages when the connection was reestablished.

I had the help of my tutor to identify where we could be having problems in the code, which was very helpful since he is the author of GO50.

Lost in a lot of lines of code, we found this extract that should return error instead of *OK*. The fact that it returned *OK* would make the sent messages' counter go up and update the last sent message ID in the database, even though they were not being sent.

This solved the issue and was validated with the same 3 initial tests.

For the delay in message forwarding, we suspicioned some database tables being too large to read. I made an analysis of the number of lines in every table that was used by GODEC diffusion (the decommissioned task), those that were currently in use and those we kept for compatibility reasons. The number of lines in each table concerned in the migration but not purged is seen in Table 1.

Table 1: Tables concerned in the migration but not currently purged

Table name	No of lines in DEV	No of lines in Prod
B_CIRC_FRONT_EU_OU_GI	612335	628229
B_EVT_CIRC_DIFF	191052	2805578
B_JOURNEE_HT_EU_OU_GI	1104	1152
B_CIRC_FRONT_OAD_EU	42048304	43351089
B_CIRC_FRONT_EU	1190818	1229664
B_ACQUIT_INTERNE	34	32
B_MSG_INTERN_DIFF0	281821	288619
B_MSG_INTERN_DIFF1	438442	458474
B_MSG_INTERN_DIFF2	404833	423847
B_MSG_INTERN_DIFF3	154895	162066
B_MSG_INTERN_DIFF4	0	0
B_MSG_INTERN_DIFF5	472085	490992
B_MSG_INTERN_DIFF6	307051	314041
B_MSG_INT_COMP0	168	6498
B_MSG_INT_COMP1	319	18131
B_MSG_INT_COMP2	339	16148
B_MSG_INT_COMP3	100	6709
B_MSG_INT_COMP4	0	0
B_MSG_INT_COMP5	379	18065
B_MSG_INT_COMP6	254	6778
H_EUROP_PR_TRANSMIS	3902	8836
B_EUROP_TYPO_CAUSE	2153	2163
B_EUROP_TYPO_CAUSE_IND	97	97

The tables with the most significant amount of data are in bold (as well as the ones with no data). *B_CIRC_FRONT_OAD_EU* has approximately 4 million lines of data, which can be a threshold for when we query it.

After presenting the results, I was asked to implement a purge on *B_EVT_CIRC_DIFF*. The existent purge for the other tables was done with an interface function, that called a SQL function to operate the database. This first function was executed at 02:00 everyday by crontab, that executes a shell script. To know how many lines each table should be purged, there is a reference table called *B_PURGE_REFERENCE* that contains the table names and for how many days a line should last based on their creation timestamp.

I noticed there was already a line referencing *B_EVT_CIRC_DIFF*, but no mention of it in the function that would read the reference table and treat each line of it. So, in this function, I implemented the following code:

For the interface function, it calls *BdbPurgeCalculDate* from the SQL function to execute the command. It also commits the transaction. The SQL function is implemented as follows:

The morning after, I was able to see that the lines had been effectively purged from the table. We conducted tests in the DEV environment and then forwarded it to PREPROD and PRODUCTION, where it hasn't been tested yet.

6.1.5.5 Logoden Web Service

This service creates incidents and events in order to suppres a future circulation.

It is a Java web service backed by a Tomcat web server that takes an XML body HTTP request in order to process the logical deletion of an expected circulation at a given day and time.

As I had acquired knowledge of the flow of incidents and events in the GODEC system and database, I was assigned to assist the other intern - a business analyst - to the creation and execution of qualification tests.

The tests were registered in Squash, a tool for software development testing, where we could keep track of test series, describe the actions for their execution and comment on what exactly went wrong.

The service would create an incident in Godec's database, that was later treated and forwarded to the external applications that receive them, propagating the deletion of the circulation.

Initially, we identified a few issues with the first version of the service. Comments in the incidents needed to be created with a tag system in order to identify which applications had access to which part of the comment section.

Another issue is that, since the service created the incident in the database and didn't notify Godec of it's existence, the treatment would take up to 15 minutes to start. This is the interval of operation of a task that checks for non treated instances in the database.

We also had to assure that the service could coexist with Godec Post, and that the changes made by it would reflect in the latter.

To have GODEC treat the incidents in real time, I digged in it's documentation to find a task that receives messages from other web services and sends a notification to Godec with the object's primary key so that it can be treated directly. This task is a simple socket server that filters messages by their types (incidents, observations, events).

After understanding how it worked, I created a dedicated instance for the Logoden web service, in a given port and instructed the developer on how to establish the connection to it and what kind of message he had to send, once the connection established. I also worked on a technical document explaining the service for future developers that may find the need to use it.

For every incident and event now created, the developer started sending the respective message to that application. In the new test series, we confirmed that the new objects were being treated in real time.

As for the comments problem, I studied the tag mechanism that was in place for Godec Post and instructed the developer on how to solve the issue, which was also successfully tested later.

For every new version, I deployed the application in the development server for testing, which was a knowledge I had passed on by our technical responsible, since I did not know how to compile a java web service with a war file backed by a Tomcat server.

In the last update, I was also responsible for the production of the deployment instruction documents for our PREPROD and PROD environments. I also assisted the integration process in those environments.

6.2 Iluric

6.2.1 Presentation of the project

Iluric is a project for the transport's visualization and coordination. It provides graphical user interfaces with visualization of transport's movements. Inspectors can adapt the paths in order to prevent concurrency within transports.

6.2.2 The team

The team is composed of 2 business analysts, one technical responsible and a manager. The last developer in this project was a part of the previous enterprise that worked with our client, 8 years ago. This meant that there was no one that could explain to me how the project worked in technical terms, and the documentation was not expressive and not very accessible.

The communication with the client was also very different from my experience with Godec. In the latter, while they would work with us and help in every possible way, the former would do the complete opposite.

6.2.3 Tasks

6.2.3.1 MQPREVHU

I was asked to conceive a new C application for Iluric, which would forward transports' forecast states to another application called Pladic via the IBM MQ series protocol. I wasn't given a proper document with a study for the new application, but a general documentation of Iluric in which someone had described the behavior the task should have. After meticulous reading, I worked on a ppt presentation for the client's Iluric team to validate what we had understood and what exactly was expected of my development.

This new application that was to be developed - MQPREVHU - has to connect to another task in Iluric in the same server via sockets, acknowledge the messages sent by the latter for integrity purposes, verify that their headers are not corrupted and forward their content via MQ Series to Pladic - another of the client's application.

I then had to estimate the time I'd spend in each action for the development of the new task. I based myself on the existing code of another task called MQOCCHO that has a very similar behavior, only with a different type of message header, content, and destination.

The result of the estimation is represented below:

Table 2: Estimation of the actions for the development of MQPREVHU.

Action	Estimation (days)
Study the implementation of MQOCCHO	1
Adapt the Socket Connection to C2D from MQOCCHO	1
Adapt the message treatment and forwarding from MQOCCHO	1
Logging in the appropriate text file	0.5
Clean up deprecated functions	0.5
Unit Tests	2

When everything was clear, I started the development, which went quite smoothly due to the preparation described in the previous paragraphs and to my studies in C during my time at UFRJ and ECL.

The real problem was compiling the new product for a Windows Server 8. Since the programs in Iluric are very old, and that there has not been a lot of development in the last decade, the server in which the tasks are deployed is a Windows Server 8. I had never dealt with any Windows Server before, let alone one that is not currently supported. The compilation of C programs, which I'd normally do with GCC on the UNIX command line, had to be done with Visual Studio. I also had to deal with a Windows wrapper for installation and execution of the program, but with the 2008 software development kit.

After compiling the program, I encountered numerous problems with dependencies like *MCVS80.dll* that were not currently available. A colleague eventually found a copy of Visual Studio 2008 so that I could compile the project. Even after deploying the developed task, I had to familiarize myself with the windows registry, which brings a similar functionality to environment variables in UNIX systems.

Once MQPREVHU was up and running, I could finally start the unit tests. They are defined down below:

Table 3: Unit tests for MQPREVHU.

Test No.	Title
1	Logging and data files generated are in the right place with their right names
2	Data Loading
2.1	The values in the Windows registry are being loaded
2.2	The values in the configuration file are being loaded
3	Messages
3.1	When headers and message are well formatted, send an ACK
3.2	When headers are badly formatted
3.2.1	Bad header ABD1 rejects message
3.2.2	Bad header PREV rejects message and sends back a NACK

The incoming messages have 2 headers and a content in XML, as illustrated in the table:

Table 4: Composition of the incoming message

Part	Size
ABD1 header	13
PREV header	9
XML content	variable lgth
ASCII NULL	1

The headers in the incoming messages are described below:

Table 5: ABD1 Header.

Field	Size	Type	Presence	Comment
Prefix(1)	1	Char	Mandatory	This field always holds the '+' char
PREV message size	8	Numeric	Mandatory	This field is filled by zeroes to the left if needed. it indicates the size of [PREV header] + [XML content]
Separator(1)	1	Char	Mandatory	This field always hold the space char (hex 0x20)
Chronological indicator	2	Numeric	Mandatory	This field is filled with a number from 0-99 for use in the acknowledgments. It resets to 0 after 99.
End of message(1)	1	Char	Mandatory	ASCII NULL
Total Length	13		Optional	

Table 6: PREV Header.

Field	Size	Type	Presence	Comment
Message type	4	Char	Mandatory	This field always holds "PGA" followed by a NULL char
Content type	5	Numeric	Mandatory	This field always holds " PREV" followed by a NULL char
Total length	9		Optional	

Since the chronological indicator for the acknowledgments is received in the ABD1 header, we can't send a NACK back to C2D as that field could be corrupted.

I implemented a test for each fixed field in the headers:

```
int mqprevhuVerifieENTETEX(s_ENTETEX entete)
{
    return
        (
            entete.pfix != ENTETEX_PFX_DEFAUT ||
            entete.blanc2 != ENTETEX_SEPARATEUR_DEFAUT ||
            entete.zero != ENTETEX_ZERO_DEFAUT
        );

}

int mqprevhuVerifieENTETE_PREV(s_ENTETE_PREV entete)
```

```c
{

    if (strcmp(entete.Type_message,
    ENTETE_PREV_TYPE_MESSAGE_DEFAUT) != 0)
    {
        return 1;
    }
    if (strcmp(entete.Type_contenu,
    ENTETE_PREV_TYPE_CONTENU_DEFAUT) != 0)
    {
        return 1;
    }

    return OK;

}
```

For the code clean up, I replaced all of the *atoi* function occurrences, since it doesn't give the possibility of error checking, for *strtoul* with the appropriate check - as seen down below:

```c
int mqprevhuCheckErrorstrtol
(char *checkError,char*chaine);

/* PARAM_TEMPO_CONFIG */
/*----------------------*/

(void) mqprevhuLectureParam
(PARAM_TEMPO_CONFIG, tempo_config_str,
PARAM_TEMPO_CONFIG_DEFAUT);

tempo_config = (int) strtol
(tempo_config_str, &checkError, 10);

if (mqprevhuCheckErrorstrtol
(checkError, tempo_config_str) == KO)
{
    mqprevhuTrace("Utilisation de tempo \
    config defaut: %s",
    PARAM_TEMPO_CONFIG_DEFAUT);
    tempo_config = (int) strtol(PARAM_TEMPO_CONFIG_DEFAUT,
    &checkError, 10);
}


```

```c
int mqprevhuCheckErrorstrtol(char *checkError, char*chaine)
{
    if (checkError == chaine)
    {
        mqprevhuTrace("Erreur de conversion \
        de string en numero. \
        string : [%s]", chaine);

        return KO;
    }

    return OK;
}
```

To conclude the task, I wrote a technical documentation for MQPREVHU.

6.2.3.2 MQ LS Network Flow

I performed a task in Galite that restructured some of the intra project communication. There were quite a few IBM MQ communications in place, each with a local MQ server in place as the middleware. Unfortunately, the price for the server's licenses has gotten significantly higher with time so the project decided to centralize all communications in a service line, as to only pay for one license.

This service line is in another environment and thus requires a distant connection, whereas with the local server, we used a local connection. I was responsible for the development in all of the C tasks that used the MQ protocol to adapt them for this new type of connection.

Every connection is established with the following information:

- **Server IP**
- **Server port**
- Queue manager
- Alias queue
- **Channel server**
- **User name**
- **Password**

The items in bold were not required before, for the local connection, so for every service concerned by the change, I implemented an import from the configuration file and a treatment for when the task failed to read those values.

The task was also followed by compilation for Windows Server 2008 and 2019, as well as unit testing and configuring for the DEV, PREPROD and PROD environments.

7 Relation between the engineering studies and the work experience

In my daily work experience I often encountered a lot of subjects that we studied at my last year at École Centrale de Lille. Even though I didn't work with embedded systems, which was my specialization, I used most of the concepts learned when we studied the C language and the UNIX system. The study of distributed systems and real time programming was also essential to the understanding of the project's integration.

I identified the same workflow for socket connections between C processes that we had gotten the practice of developing in distributed systems and in our final project.

What I learned about project management was also of the highest importance to understand work communications and how to validate, present and organize what I produced, which I sense was a very valuable asset to the team.

The enterprise notions I acquired with my *Filière Métier* - Responsible of Supply - were also very useful knowledge to understand the company's and project's context. I could understand more of my value and what was expected of me.

8 Level of liberty in choosing one of the possible solutions

Mostly, I'd be given a task to be implemented from a study that had already been done and where the decision had already been made.

Nonetheless, I encountered some situations where I had to help to choose and propose a solution for a problem. This was the case for the purge of the GO50 tables described in section 6.1.5.4, where I helped decide the amount of days the purge should conserve.

However, most of the times my suggestions were not taken into account. Either because it was not simple to address them or because it wasn't in the short term interest of the company. One example was when I suggested to implement a DevOps system for the monitoring of Godec since we had to do that manually and were sometimes surprised with a function that had stopped working days before we could realize. I talked about this with my upper manager and he told me that this service already existed, but it had limited access, limited functionality and wasn't managed by our team. Later on, other teams of the client developed some datadog dashboards to which we had access after realizing the real need of this monitoring.

I also proposed to turn automatic the every day rite that is performed to check the status of all of the components of the application. This task takes an average colleague about 30-50 minutes and is always seen as something dull. Again, this proposition was turned down because this task, for a very few number of colleagues only takes 10 minutes to finish. Even so, it is still subject to human error which would still justify the need for automation. The project seemd to have other priorities, which is understandable.

9 Clarification of the methodology

As we worked with ancient applications, our team operated in the V cycle. We'd have daily meetings to share the advances made in the day before and what we had to do for the current day. Our project manager could intervene by helping us prioritizing our tasks or assigning us new ones.

The tasks were followed by JIRA tickets, where we could account the hours spent in them, as well as estimate the remaining work time to finish them.

Every development was followed by a technical document, explaining every function, global variable and macros used in the program. Unit and qualification tests were registered in SQUASH and results were presented to the client.

The modifications made are registered in our client's GitLab and integrated in the different environments after testing.

I'd also respond to incidents, that were in the form of service now tickets. On a weekly basis, we'd have meetings with the client to talk about the advancements in our analysis of the problems, which would mostly lead to anomaly correction, that then enters the regular development workflow.

Weekly meetings for feedback and general perspective of the projects were in place, but most of the time canceled due to deadlines or last time problems the team had to address.

10 Reflective analysis : developed competences

The point I developed the most during this internship is my understanding of business operations. To be able to better define which decisions are better both for the enterprise and for the client. Communicating with my colleagues and with the client were key aspects of my role and, during this time, I was able to sharpen this ability. I also developed my communication skills in French, both personally and professionally.

From a technical point of view, I really enjoyed learning the development and deployment workflow used by the company. I experienced industrial scale deployments in C and the complexity of all the dependencies and includes. I learned how to compile C for a Windows Server 2008 and java for a Tomcat web service, as well as to formalize tests in a tool like SQUASH.

11 Intern's place and experience in the enterprise and relation with colleagues

I had a very slow start in the company. This can be considered normal behavior for the first week or two, but I'd say that it lasted for at least a month and a half. I joined a very complex project - GODEC - and had no one to formally explain it's functional nor technical aspects to me. I was given an old version of it's documentation on the first day, a brief explanation of the project's architecture and the source code. Within a week I had a few incidents to correct in the application, which took more time than they should, had I had a proper guidance in the project. Later on, I was often given incidents to treat, each time demanding a bigger understanding of the project, which I had nowhere to get from. I realized those incidents were considerably old and were still there because they were hard and boring to solve. I would contact the other developers of the team and get no response for days. I eventually got out of this situation when I proposed to work with the memory leak problem in the Super Concentrator. I then grasped better knowledge of the project and started being assigned to tasks related to the evolution of the project. With the accumulated knowledge, I started to get more at ease with the incidents that were initially sent to me for correction, as well as with the other colleagues, as they had had the opportunity to work a bit more with me.

A part of this initial communication problem with my colleagues is due to the fact that they were all situated in Paris, while I was at Lille. Having the opportunity to go to Paris to work for a day, I saw how much easier it would have been to learn from the others, had I been physically there. Information is exchanged a lot faster, naturally and ignoring a coworker becomes a lot harder.

When I didn't have communication issues, I had a rather pleasing experience in working with my team. When it concerned something other than an incident, they were very reliable and eager to help.

I was also able to talk to other colleagues in Lille that were not a part of my project about career prospection and what they expected for the future.

12 Conclusion

The experience was very valuable to my formation as an engineer, as I developed knowledge and practice in my field that will certainly serve me in the future. I realized how the engineering career means a lot more than the technical knowledge alone. I'd say I now have a better vision of what I still have to learn in order to become a better professional.

The communication with my colleagues was not the simplest, since my team was located in Paris, except for my tutor. Even though, I was able to create bonds with them as well as with my co-workers at Lille. The follow up on my integration wasn't what I expected, but the efforts I put in seem to have compensated for that. I was able to actively contribute to the team and to the project, which is very satisfying.

I was also able to see the importance I give to working towards something directly useful to society, such as the evolution in the transporting sector. Before, I was offered a spot for the banking/insurance sector of the same company, which I am happy to have refused over transport. If I can find myself useful to others throughout my career, it will likely be the more attractive track to follow.