Brevet de Technicien Supérieur SN

Session 20XX

Lycée Saint Gabriel – Saint Michel

SG01 – MakAir

Guillouet Geoffrey

Dossier technique du projet – partie individuelle

I - Situation dans le projet

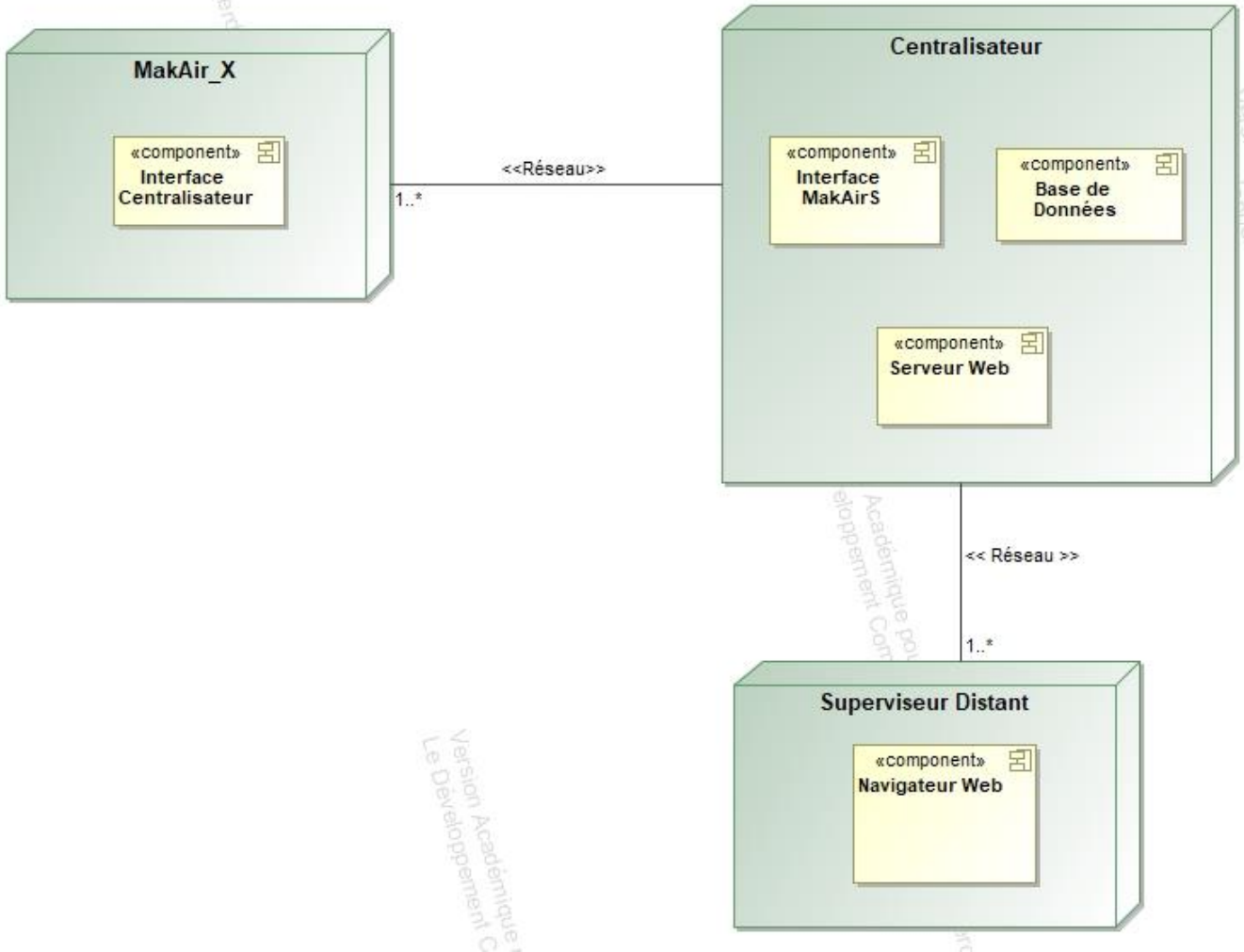
I.1 - Introduction

Dans ce projet, nous travaillons sur un appareil d'aide à la respiration, qui se nomme MakAir. Notre but principal est de récupérer les données de l'appareil et de les envoyer sur un centralisateur afin de les afficher sur une interface web.

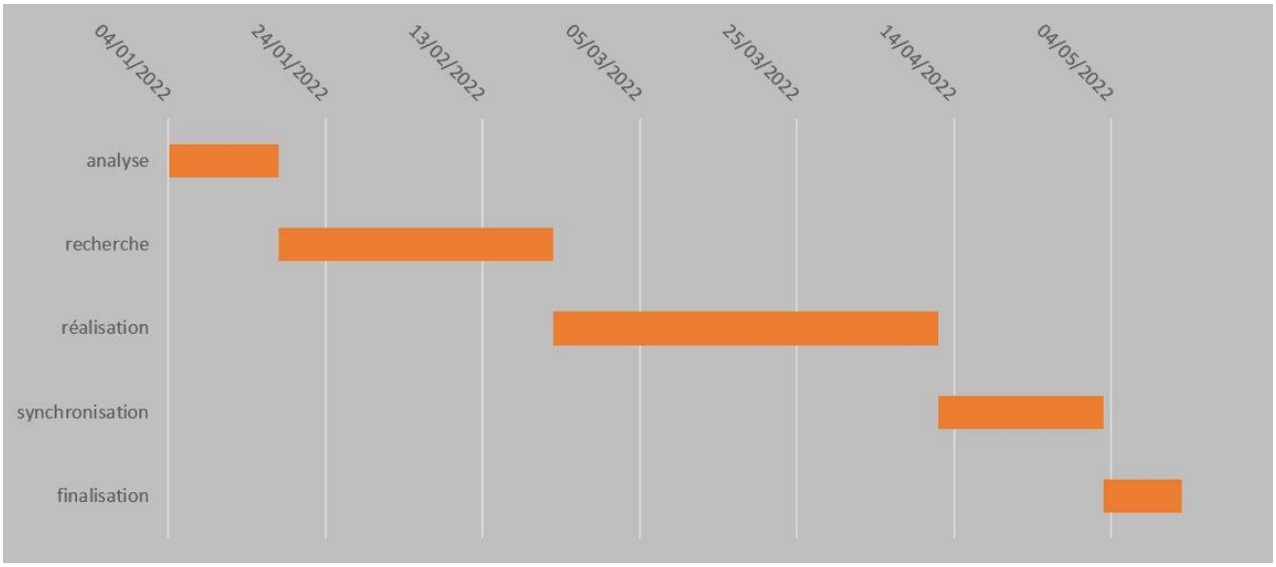
- *A partir de but principal, il découle des tâches parallèles, comme par exemple la détection et l'enregistrement des MakAirs afin de pouvoir communiquer avec eux et récupérer leurs données. Ce travail est une de mes parties.*

- *Une autre de mes tâches est de mettre en place des alarmes par rapport à l'état du MakAir mais aussi l'état du patient.*

Ainsi je travaille sur les deux côtés, c'est-à-dire sur le MakAir mais aussi sur le centralisateur. Comme le montre le diagramme de déploiement suivant :



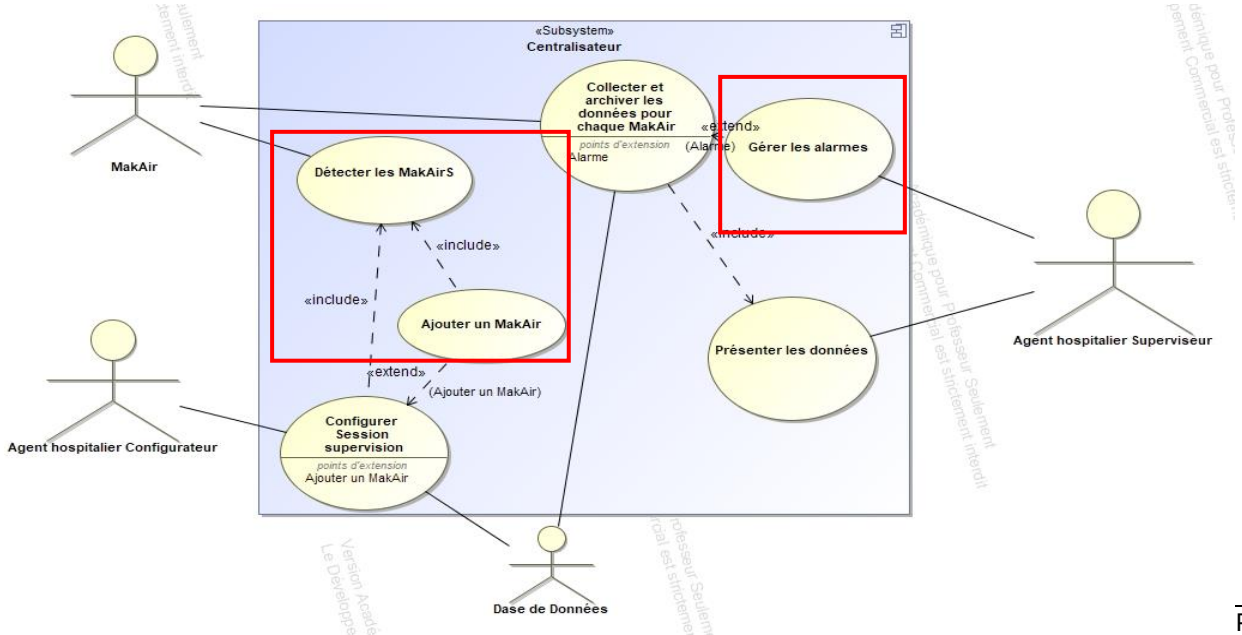
- Nous avons commencé notre projet au mois de janvier, d'abord nous avons analysé le cahier des charges qui nous a été données, d'abord de manière individuelle puis nous l'avons fait en groupe, pour voir si tout le groupe avait bien compris le projet ainsi que ses tâches. Ensuite, nous avons entamé une longue période de recherches afin de savoir comment faire pour réaliser nos tâches avec quels outils, etc…
  Suite à cela nous avons eu notre première revue de projet avec nos professeurs, qui nous ont soit confortés dans nos choix soit guidé vers une autre solution. En fonction de leurs conseils nous avons repris nos recherches pour renforcer nos choix. Puis nous avons commencé la réalisation, actuellement nous sommes encore dans cette phase pour tout le groupe. Voici le diagramme de Gant commun avec tout le groupe :



I.2 - Synoptique de la réalisation

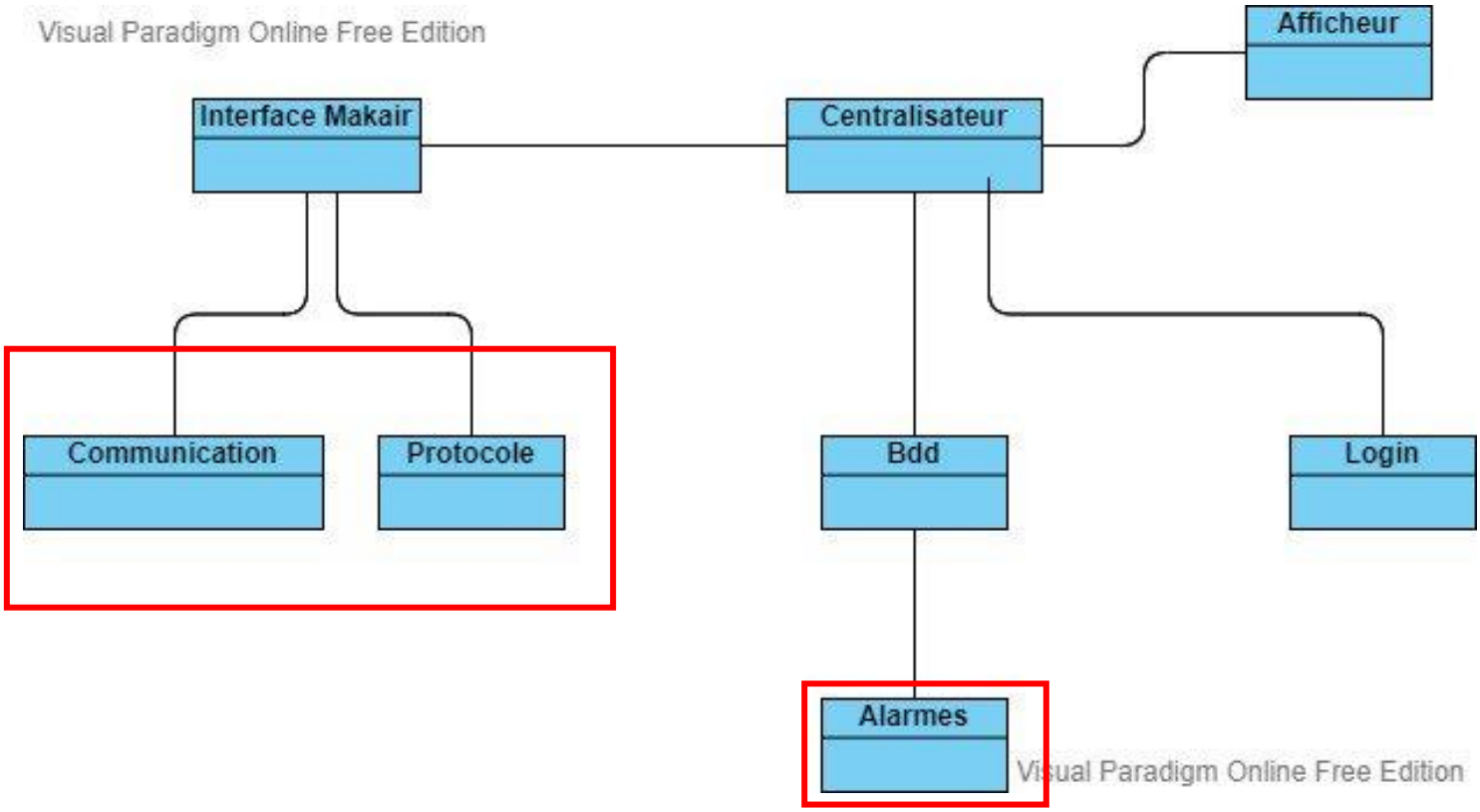
- Maintenant je vais vous montrer où se trouve ma partie plus en détail. Ci-dessous, je vous explique succinctement la partie centralisateur et sur l'embarqué, ma partie se trouve dans les encadrés rouges.

Sous-Système Centralisateur :



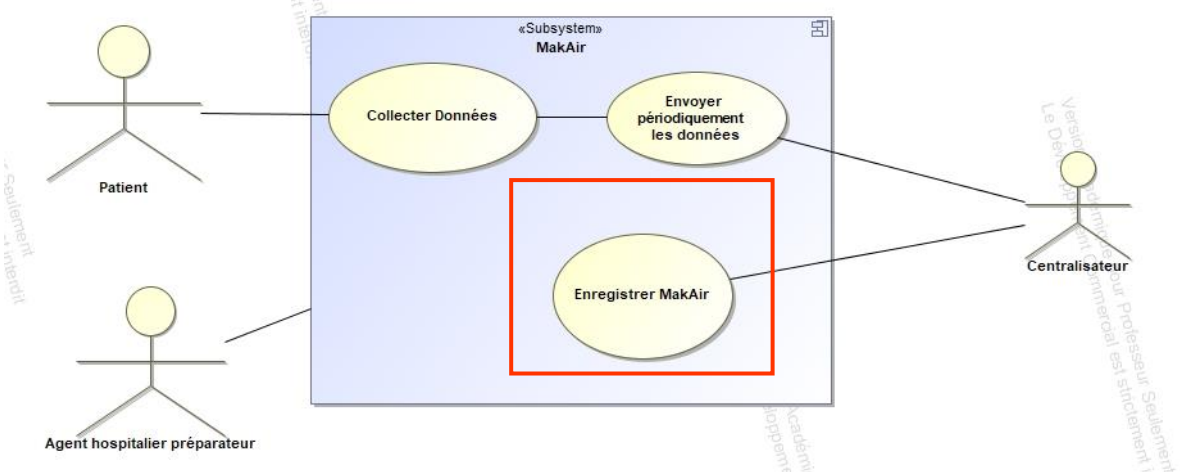
- Cet agent déclenche la reconnaissance des MakAirs disponibles.
- Des alarmes se déclenche au moindre problème.

Une fois configuré, le Centralisateur fonctionne de manière automatique, autonome, sans intervention d'un opérateur.
Ce système est simulé sur un ordinateur via une machine virtuelle avec un serveur LAMP (Linux Apache Mysql Php).



Ci-dessus, vous pouvez voir le diagramme de classes pour la partie centralisateur, pour ma part je m'occupe de coder les classes dans les encadrées rouges.

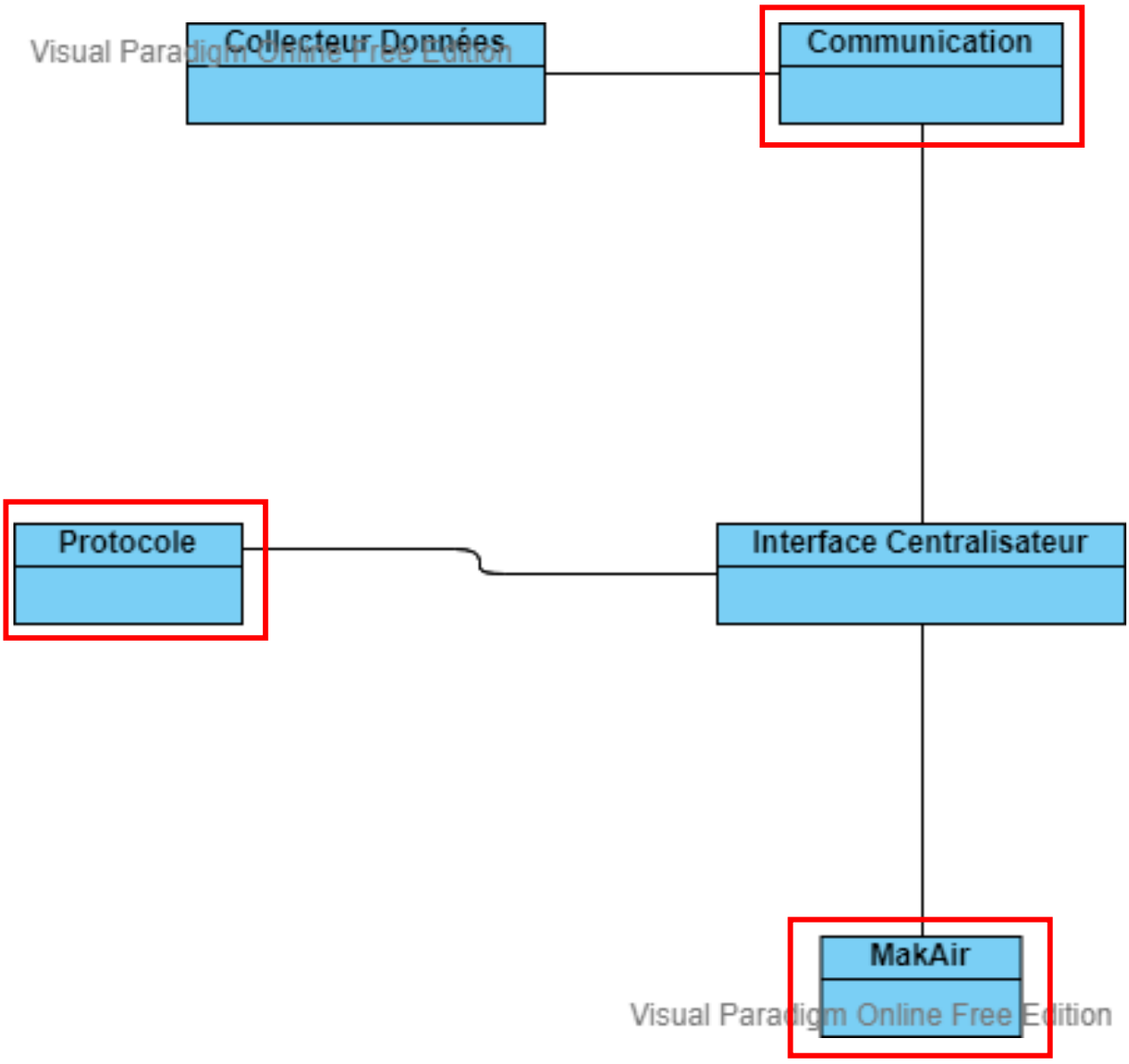
Sous-système Embarqué



Ainsi comme sur l'image ci-dessus, mon but est que :

- Quand le centralisateur fait un scan pour retrouver les MakAirs, ils répondent pour s'enregistrer.

Une fois configuré, l'embarqué fonctionne de manière automatique, autonome, sans intervention d'un opérateur, c'est-à-dire qu'il s'enregistre tout seul puis envoie les données. Ce sous système est simulé sur une Raspberry pi 3b, qui directement relié à un MakAir fourni par l'entreprise Makers for life pour notre école.



Voici ci-contre, le diagramme de classes pour la partie embarquée. Mes parties sont dans les encadrés en rouges.

II - Réalisation du cas d'utilisation « UCE2 – Enregistrer MakAir » et du cas d'utilisation « UC3 – Détecter les MakAir »

II.1 - UC3 – Détecter les MakAir

II.1.1 - Analyse

Ce cas d'utilisation consiste à faire une recherche automatique et répéter sur le réseau pour détecter les nouveaux MakAir.

Extrait du cahier des charges :

Nom du cas d'utilisation	**UCC3 – Détecter les MakAirs**
Pré-condition(s)	Les MakAirs sont disponibles sur le réseau et prêts à être détectés.
Scénario nominal	Le Use Case détecte les MakAirs en scrutant le réseau
Séquencement	En boucle, (mode polling). Scruter le réseau et faire la liste des MakAirs disponibles
Post-condition	Tous les MakAirs du réseau sont listés
Exigences	

Plus précisément, ce cas d'utilisation demande la mise en place d'un envoi bouclé afin de faire la liste de tous les MakAir.

II.2 - UCE2 Enregistrer MakAir

II.2.1 - Analyse

Ce cas d'utilisation consiste à mettre à jour la base de données du centralisateur en y enregistrant automatiquement le nouveau MakAir.

Extrait du cahier des charges :

Nom du cas d'utilisation	**UCE2 – Enregistrer MakAir**
Pré-condition(s)	Le MakAir est allumé et configuré. Il est opérationnel. De plus le MakAir est connecté au Centralisateur.
Scénario nominal	De manière automatique et complètement autonome, le MakAir s'enregistre auprès du Centralisateur.
Séquencement	Le Centralisateur effectue une recherche et l'enregistre
Post-condition	Le MakAir est enregistré sur le Centralisateur
Exigences	

Ici, le MakAir doit s'enregistrer tout simplement en envoyant ces données au centralisateur pour les écrire dans la base de données. Le choix d'envoi des données dépend des clients, en fonction des besoins et des envies. Pour ma part, j'ai choisi comme données :

- L'identifiant du MakAir ;
- L'adresse IP du MakAir ;
- Le nom du patient attribué à l'appareil ;
- Le nom du médecin en charge du patient ;

II.3 - Mise en relation

Après une analyse approfondie, j'ai compris que le Centralisateur effectue une recherche sur le réseau pour connaitre les différents les MakAirs présent ce qui correspond au cas d'utilisation 3. Une fois la demande reçue le MakAir doit lui répondre en lui envoyant toutes ces données pour s'enregistrer ce qui correspond au cas d'utilisation 2. C'est pourquoi j'ai décidé de lier ces deux cas d'utilisation, car ils dépendent l'un de l'autre.

Comme les données a envoyé peuvent changer en fonction des besoins, j'ai décidé de créer une classe qui contiendrait toutes les données a envoyées. Cette classe se nomme MakAir. Ainsi en utilisant cette classe, je peux la configurer facilement pour chaque client.

Ensuite, pour envoyer les données le MakAir va devoir communiquer avec le centralisateur. Ainsi j'ai décidé de réaliser une classe « Communication ». Cette classe sera utilisé par les deux appareils (le centralisateur et la Raspberry).

Comme on demande au respirateur d'envoyer ses données via le réseau, mais qu'il s'agit de données médicales alors on a décidé d'instaurer un minimum de sécurité. Ainsi j'ai choisi de créer un protocole unique et interne à mon code pour pouvoir communiquer sans risque de fuite des données. J'ai par conséquent codé une classe « Protocole ». Cette classe sera aussi utilisée par les deux appareils, puisque les deux appareils doivent envoyer un message mais aussi le décoder.

II.4 - Partie codage

Maintenant je vais vous montrer et expliquer les classes que j'ai codées pour répondre aux cas d'utilisations.

II.4.1 - La classe « MakAir »

Pour rappel, cette classe sert à stocker les données du MakAir que je dois envoyer. Je vous rappelle que les données à envoyer varie en fonction du client et des besoins.
Pour réaliser cette classe, j'ai préalablement choisi quelques données à envoyer, que j'ai moi-même créer.
J'ai choisi comme données :

- L'identifiant du MakAir ;
- L'adresse IP du MakAir ;
- Le nom du patient attribué à l'appareil ;
- Le nom du médecin en charge du patient ;

Pour cette classe, je n'ai codé que le « header », ainsi on peut dire que cette classe est plus une bibliothèque.

```
#pragma once
class MakAir
{
private:
    char identifiantMakAir[2] = "1";
    char adresseIP[16] = "192.168.114.100";
    char nomPatient[100] = "RICHARD Bertrand";
    char nomMedecin[100] = "SHEPERD Derek";
};
```

Voici ci-dessus une capture d'écran de la classe MakAir, vous pouvez voir que tous les éléments sont déclarés en tant que « char », l'explication se fera dans la classe « Communication ».

II.4.2 - La classe « Communication »

La classe « Communication » est la classe centrale de ma partie, elle permet la communication entre le centralisateur et le MakAir. Après plusieurs recherches et échanges avec les professeurs, je me suis renseigné sur les sockets.
Un socket est un « connecteur réseau » ou « interface de connexion », il permet d'établir une session UDP, puis de recevoir et d'expédier des données grâce à elle.
J'ai donc utilisé les sockets pour faire communiqué le centralisateur avec le MakAir.
Normalement, le socket met en place un serveur et un client, cependant pour mon travail je vais mettre en place mon code comme si j'avais deux serveurs.
Parce que le MakAir doit attendre une demande du serveur puis envoyer une réponse. Alors que si je restais dans une structure Serveur/Client, cela voudrait dire que le client doit connaitre au préalable l'adresse IP du serveur, ce qui dans notre situation n'est pas le cas.
Cependant pour m'aider dans la réalisation j'ai défini l'adresse IP du centralisateur
Sur la page suivante une capture d'écran du header de la classe communication :

Communication.h

```
#pragma once
#include<iostream>
#include<arpa/inet.h>
#include<unistd.h>
#include<sys/socket.h>
//#include<sys/types.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
using namespace std;

#define LENGTHDATA 512
#define PORT 8888

class Communication
{
private:
    struct sockaddr_in serveur, client;
    int socketUdp, i, slen = sizeof(client), recv_len;
    char dataRecu[100];
    char reponse[100] = "ok";

public:
    Communication();
    void CreerSocket();
    void Bind();
    char RecevoirDonnees(char test[10]);
    void EnvoyerDonnees();
    void DetruireSocket();
};
```

Je vais maintenant détailler succintement le rôle de chaque fonctions :

-CreerSocket() est la fonction qui va créer le socket et par conséquent le canal de communication entre le centralisateur et la raspberry pi.

-Bind() est la fonction qui permet de rester à l'écoute sur un port bien précis, il se ferme dès qu'il reçois un message.

-RecevoirDonnees est la fonction qui permet de recevoir les données et qui va les transmettre à la fonction de décodage.

-EnvoyerDonnees() est la fonction qui envoie les données, je vous l'explique plus en détail à la suite.

-DetruireSocket() est la fonction qui va détruire le socket car nous devons supprimer le socket une fois la communication terminé.

Maintenant, je vais un peu plus détailler la fonction EnvoyerDonnees(). Tout dabord voici le code de cette fonction :

```
void Communication::EnvoyerDonnees()
{
    if (sendto(socketUdp, reponse, strlen(reponse), 0, (struct sockaddr*) &client, slen) == -1)
    {
        cout << "erreur" << endl;
    }
}
```

Ici l'élément principale est la fonction « sendto » qui est une fonction fournit avec une bibliothèque qui est « sys/socket.h »

Pour utiliser cette fonction, nous devons mettre certains paramètres, mais ici celui qui va nous intéresser pour la suite est le paramètre dans l'encadré rouge ci-dessus. Ce paramètre est un tableau de char.

Je le précise parce que cela va définir mon protocole qui sera obligatoirement un tableau de char pour pouvoir utiliser cette fonction et transmettre des données.
Ci-dessous, se trouve les différents tests que j'ai effectué sur cette fonction.

Élément testé :	Test de fonction EnvoyerDonnees		
Objectif du test :	Si la fonction envoie les données		
Nom du testeur :	Guillouët Geoffrey	Date :	09/05/2022
Moyens mis en œuvre :	Logiciel : Visual studio MobaXternm	Matériel : Machine virtuelle Raspberry pi	Outil de développement : Visual studio

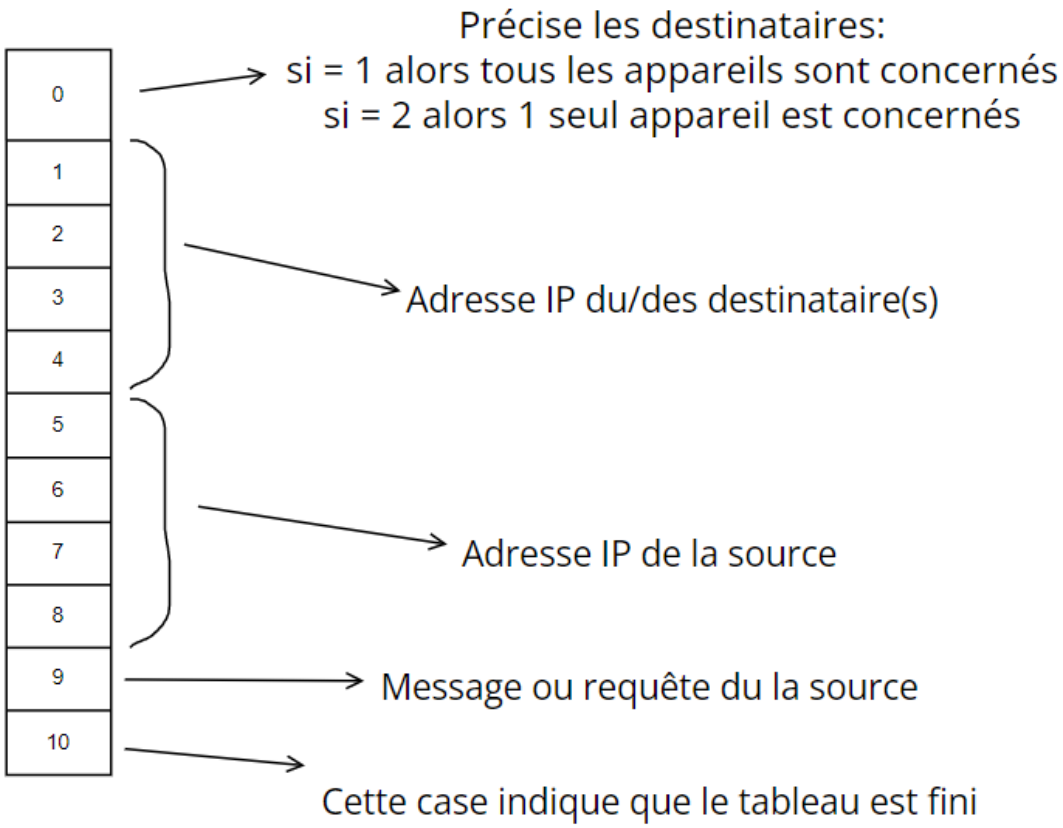
Procédure du test :

Id	Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation (O/N)
	Test de la fonction envoyerDonnees avec en paramètre un caractère	Réception de la valeur identique	Valeur identique	O
	Test de la fonction envoyerDonnees avec un tableau de char en paramètre	Réception du tableau à l'identique	Tableau identique	O

Conclusion du test :	Le test est validé partiellement

II.4.3 - La classe Protocole

La classe « Protocole » est celle qui créer le paquet que je vais envoyer via la classe « Communication ».
Etant donné que j'ai du créer mon protocole, je vais le détailler.
Voici ci-dessus un schéma du protocole :



Je précise qu'il s'agit d'un tableau de char par obligation avec la fonction « sendto » (vu précedemment).
Pour construire ce tableau, j'ai codé plusieurs fonctions que vous pouvez voir ci-dessus dans le header de la classe « Protocole ».

Protocole.h

```
1   #pragma once
2   #include <iostream>
3   #include <string>
4   using namespace std;
5
6   class Protocole
7   {
8   private:
9
10
11  public:
12      Protocole();
13      void choixDestinataire(int protocole[]);
14      void creerMessage(int protocole[]);
15      void afficherTrame(int protocole[]);
16      void decoderTrame(int protocole[]);
17  };
18
19
```

Je vais maintenant expliquer brièvement le rôle de chaque fonctions :

-choixDestinataire() est la fonction qui va permettre de choisir qui est/sont le/les destinataire(s). Il existe 2 états : soit tous les ordinateurs sont les destinataires soit il y a que 1 seul ordinateur destinataire et elle va aussi remplir l'adresse IP destinataire et source dans le tableau.

-creerMessage() est la fonction qui permet de choisir le type du message soit c'est une demande de données soit c'est une réponse, un envoie de donnée.

-AfficherTrame() est juste une fonction qui permet d'afficher le tableau que j'ai créée, juste pour du debug.

-decoderTrame() est la fonction qui va décoder le tableau qu'il va recevoir et savoir si il s'agit d'une demande ou d'une réponse et agir en fonction.

Voici ci-dessous la fonction choixDestinataire(), pour pouvoir faire des tests j'ai rajouté des petites interactions.

```
void Protocole::choixDestinataire(int protocole[])
{
    int reponse;

    cout << "Avec qui souhaitez-vous communiquer? " << endl;
    cout << "1. Tout le monde (broadcast)" << endl;
    cout << "2. un pc" << endl;
    cin >> reponse;

    if (reponse == 1)
    {
        protocole[0] = 1;      // Remplacer les 255 par l'addresse de broadcast
        protocole[1] = 255;     //Suggestion ecrire l'addresse de broadcast dans les données membres
        protocole[2] = 255;    // Suggestion l'écrire dans la bdd comme données fixe et aller la chercher
        protocole[3] = 255;
        protocole[4] = 255;
    }

    else if (reponse == 2)
    {
        protocole[0] = 2;
        protocole[1] = 192;
        protocole[2] = 168;
        protocole[3] = 115;
        protocole[4] = 101;
    }
    else {
        //end of the fonction
    }
}
```

```
//DEBUG de la trame
void Protocole::afficherTrame(int protocole[])
{

    cout << endl;
    cout << endl;

    //boucle afficher tableau protocole to check puis refaire tourner
    for (int i = 0; i < 6; i++)
    {
        cout << protocole[i] << endl;
    }
    cout << endl;
    cout << endl;

}
```

Maintenant je vais donc tester si en fonction des variables que je choisi le tableau est créer parfaitement. Puis je vais l'afficher avec la fonction « afficherTrame ».

Élément testé :	Test de fonction choixDestinataire		
Objectif du test :	Le tableau est créer correctement		
Nom du testeur :	Guillouët Geoffrey	Date :	09/05/2022
Moyens mis en œuvre :	Logiciel : Visual studio MobaXternm	Matériel :	Outil de développement : Visual studio

Procédure du test :

Id	Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation (O/N)
	Lancement de la fonction et sélection des variables proposé	Le tableau est créé correctement	Tableau ok	O
	Lancement de la fonction et sélection de variable non proposé	La fonction se relance et redemande à nouveau	La fonction se relance et redemande à nouveau	O

Conclusion du test :	Le teste est valider

III - Réalisation du cas d'utilisation « UCC5 – Gérer les alarmes »

Pour ce cas d'utilisation, je dois la réaliser en association avec l'étudiant 3.

Nom du cas d'utilisation	**UCC5 – Gérer les alarmes**
Pré-condition(s)	Les données sont disponibles ainsi que les niveaux d'alarmes.
Scénario nominal	Le Use Case gère les alarmes en fonction des données qui arrivent en temps réel. (période T)
Séquencement	Lorsque nécessaire, les alarmes sont déclenchées et l'Agent hospitalier superviseur est informé par sms.
Post-condition	L'alerte est émise vers l'agent hospitalier
Exigences	

III.1 - Conception détaillée

III.1.1 - Analyse

A l'heure, je n'ai pas encore réalisé ce cas d'utilisation. Tout simplement parce que nous avons quelques difficultés. De plus nous avons jugé que ce cas d'utilisation n'était pas une priorité pour notre projet. Par conséquent, nous nous sommes concentrées sur nos autres taches. Cependant nous avons un petit peu étudié ce cas d'utilisation et nous partirions sur une comparaison de table dans la base de données.
C'est-à-dire que nous allons créer une table avec des données que les médecins nous dirons critiques. Puis nous la comparerons avec la table où se situe les données du patient. Si les données du patient sont inférieures ou supérieures aux données critiques alors on déclenche une alarme soit sur l'ordinateur qui se chargera du centralisateur soit directement sur le portable du medecin chargé du patient via sms.
Toute cette partie se fera en php.

IV - Bilan de la réalisation personnelle

- A l'heure actuelle, j'ai fini l'installation et la configuration du serveur LAMP puisqu'il tourne parfaitement.
- De plus, j'ai fini de codé toutes mes classes, j'ai essayé de les regrouper cependant je n'ai pas réussi à les faire fonctionner ensembles.
- Il me reste plus que le cas d'utilisation sur les alarmes mais nous avons déjà des pistes pour le réaliser.
- Sur le point personnel, je trouve ce projet très intéressant, premièrement par le fait qu'il s'agisse d'un projet d'actualité avec le Covid. Puisque nous avons un entretien avec le CHU de Cholet pour présenter notre projet.
- Cependant, il était aussi très stressant par ce que je dois faire des recherches pour apprendre des choses que je ne connaissais pas et sans l'aide des professeurs, en revanche cela me permet d'explorer encore plus l'informatique ainsi cela m'a aidé dans les choix pour mon parcours professionnel.