PARTIE DE L'ETUDIANT B - LUDWIG ALEXANDRE

Table des matières

Présentation du matériel utilisé

Présentation du XBEE

Un module XBee est un circuit de communication sans-fil utilisant les **protocoles 802.15.4 et/ou Zigbee**, permettant de réaliser différents montages, d'une liaison série RS232 classique à un réseau maillé (mesh) auto-configuré.

Un composant XBee série 1 coûte désormais **moins de 20 euros**.

Selon les modèles et les pays, ils utilisent la bande des 2,4 gigahertz (comme le Wifi ou le Bluetooth) ou les 900 MHz. Selon leur puissance, ils émettent à une distance comprise entre 30 mètres et 1,5 kilomètre.

Ils sont développés à l'origine par la société Maxstream, devenue Digi. Vous pouvez toujours trouver en vente les premières séries du module marquée Maxstream.

Le débit peut atteindre 250 kbps, mais si on les utilise pour réaliser une liaison série sans fil, les débits standards sont compris entre 9600 bps à 38400 bps.

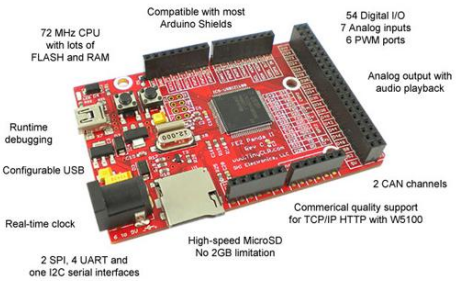
Ce module possède des cartes de tests et un logiciel de configuration (XCTU) afin de faciliter son utilisation.

Le XBee possède trois modes : **TRANSPARENT**, **COMMAND** et **API**.

- Le mode TRANSPARENT est le mode par défaut à la mise en marche du module, celui qui reçoit et envoie les données.
- Le mode COMMAND permet de configurer le module, ses entrées, ses sorties, son adresse, l'adresse de destination de ses messages, etc.
- Le mode API est un peu plus compliqué, il faut fabriquer ces propres trames et les envoyer au module. Ce mode est plus puissant et rapide mais nécessite plus de programmation et un firmware de type API.

Ces modes sont exploitables via X-CTU ou bien HyperTerminal.

Présentation du calculateur PANDA



La platine FEZ "PANDA II" est probablement le système de développement embarqué (capable d'être programmé sous environnement Microsoft™ .NET Micro Framework™) parmi le plus petit et le plus économique du marché.

Se présentant sous la forme d'une petite platine, le FEZ "PANDA II" bénéficie d'une librairie de fonctions étendues lui permettant de gérer très facilement des entrées/sorties tout ou rien, des entrées de conversion analogique/numérique, une sortie analogique, des ports de communication mais également la gestion de cartes mémoires SD™.

Un connecteur mini USB permet de relier le module à un PC afin de pouvoir télécharger et debugger les applications

Des extensions pour la carte PANDA sont disponibles, comme par exemple un afficheur tactile.

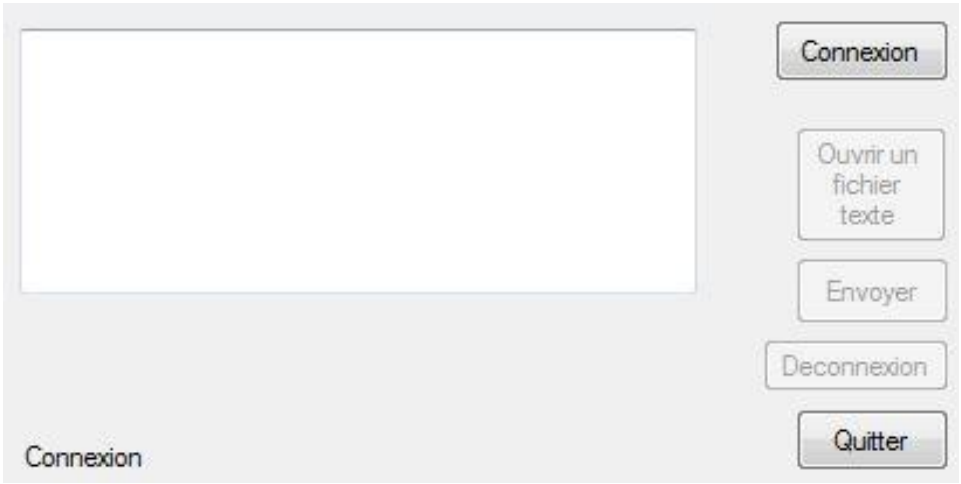
Nous utilisons cette carte car elle est compatible avec les cartes ARDUINO, permet d'accueillir un capteur de poids, de température et notre module Xbee sans fil !

Tests unitaires

Premier test unitaire : Communication entre les deux modules Xbee

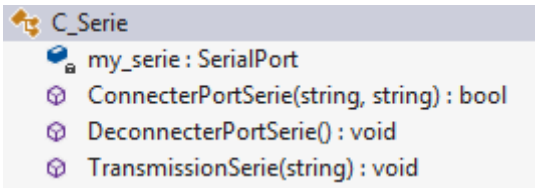
Afin de pouvoir faire communiquer les deux modules Xbee, j'ai créé deux applications : Une pour le module émetteur, et une pour le module récepteur.

L'application Xbee émetteur est une application Windows forms codée en c# :



Tandis que l'application Xbee récepteur est une application console codée en c# :

L'application émettrice possède une classe C_Serie qui contient les méthodes ConnecterPortSerie, TransmissionSerie, DeconnecterPortSerie, et une instance de type SerialPort nommée « my_serie » :



La méthode ConnecterPortSerie permet d'ouvrir le port série auquel le Xbee est connecté (COM1) :

```
public bool ConnecterPortSerie(string portCom, string baudrate)
{
    bool EstOuvert = false;
    my_serie.PortName = portCom;
    my_serie.BaudRate = Convert.ToInt32(baudrate);
    my_serie.DataBits = 8;
    my_serie.StopBits = StopBits.One;
    my_serie.Parity = Parity.None;
    my_serie.Handshake = Handshake.None;
    my_serie.Open();

    if (my_serie.IsOpen == true)
    {
        EstOuvert = true;

    }
    return EstOuvert;
}
```

Cette méthode est appelée lors du clique du bouton « Connexion » de l'interface graphique :

```
private void b_connect_Click(object sender, EventArgs e)
{
        bool connect = false;
        connect = Xbee.ConnecterPortSerie("COM1", "9600");
        if (connect == true)
        {
            l_connexion.Text = "Connecté !";
            b_connect.Enabled = false;
            b_deconnect.Enabled = true;
            b_ouvrir.Enabled = true;
            b_envoi.Enabled = true;

        }
        else if (connect ==false)
        {
            l_connexion.Text = "Erreur communication";
            b_envoi.Enabled = false;
            b_ouvrir.Enabled = false;
        }

}
```

Nous devrions avoir ce résultat similaire :

Désormais, nous pouvons entrer un texte à envoyer au récepteur dans la zone de texte, puis l'envoyer grâce au bouton « Envoyer » qui appelle la méthode « TransmissionSerie » :

```
public void TransmissionSerie(string data)
{
    my_serie.WriteLine(data); //ecris les données sous forme d'une chaîne de caractère sur le Xbee
}
```

Code du bouton « envoyer » :

```
private void b_envoi_Click(object sender, EventArgs e)
{
    string Texte = textBox1.Text;
    Xbee.TransmissionSerie(Texte);
}
```

Nous pouvons également envoyer le contenu d'un fichier texte (.txt) grâce au bouton « Ouvrir un fichier texte », dont le code est ci-dessous :

```
private void b_ouvrir_Click(object sender, EventArgs e)
{
    string myFile = "";//le nom du fichier
    string data = "";


    //choisir le fichier
    OpenFileDialog myofd = new OpenFileDialog();
    myofd.InitialDirectory = @"d:\";
    if (myofd.ShowDialog() == DialogResult.OK)
    {
        myFile = myofd.FileName;
    }

    //ouvrir, lire le fichier et le fermer
    if (myFile != "")
    {
        StreamReader mystr = new StreamReader(myFile);
        data = mystr.ReadToEnd();
        mystr.Close();
    }

    //afficher le contenu lu
    textBox1.Text = data;
}
```

Du côté de l'application réceptrice, le contenu du xbee est analysé en permanence grâce à une boucle while(true) et grace à la méthode ReceptionSerie, avec un intervalle de 1 seconde :

```
try
{
    C_Serie.Connexion("COM20", 9600, 8);
}
catch
{
    Console.WriteLine("Echec de la connexion, vérifiez la connexion puis relancez le programme");
}
```

Méthode ReceptionSerie :

```
public string ReceptionSerie()
{
        Thread.Sleep(100);//attente pour les gros paquets de données
        string data = my_serie.ReadExisting(); //Lis les données existantes sur le périphérique du port série
        return data; //Retourne ces données
```

Remarque : Le Xbee récepteur, malgré le fait qu'il soit connecté en USB, est traité de la même manière qu'un module série, grâce à un pilote qui permet de générer un port série correspondant au port USB utilisé par ce module (Digi USB Driver) . Pour tous les programmes, et pour le poste de l'apiculteur, le port affilié au Xbee récepteur sera toujours le port COM20. Le numéro de port est paramétrable dans le gestionnaire de périphériques du PC.

Les modules communiquent parfaitement ensemble et quasiment instantanément :

Deuxième test unitaire : Intégration du module émetteur sur le calculateur PANDA

Pour le test précédent, le module Xbee a été directement connecté sur le port série du PC, il sera cependant connecté au calculateur panda lors du déploiement du système.

Pour s'assurer de la bonne communication des deux modules avec le calculateur PANDA pour l'émetteur, j'ai créé un programme sur le calculateur PANDA qui envoie 3 nombres identiques séparés par des espaces, qui pourrait correspondre par exemple au poids de la ruche, à la température de la ruche, et au niveau de batterie. Ces nombres seront compris entre 30 et 40, et augmentent au fur et à mesure des envois.

Cette application sera une application de type MicroFramework.

```
public static void Main()
{
    SerialPort xbee_E = new SerialPort("COM1", 9600, Parity.None, 8, StopBits.One);
    xbee_E.Open();

    UInt16 i = 0; // Valeur à transmettre

    // Création d'une chaîne de caractère et d'un buffer d'émission
    string counter_string;
    byte[] buffer;

    // Envoi de la valeur toutes les 2s
    while (true)
    {
        for (i = 30; i <= 40; i++) // Nombre de 30 à 40
        {
            counter_string = i.ToString() + " " + i.ToString() + " "+ i.ToString() ; //Création d'une chaine sous la forme : "nombre nombre nombre"
            buffer = Encoding.UTF8.GetBytes(counter_string);
            xbee_E.Write(buffer, 0, buffer.Length); //Envoie de la chaîne sur le Xbee
            Thread.Sleep(2000); //Envoie toutes les 2 secondes
        }
    }
}
```

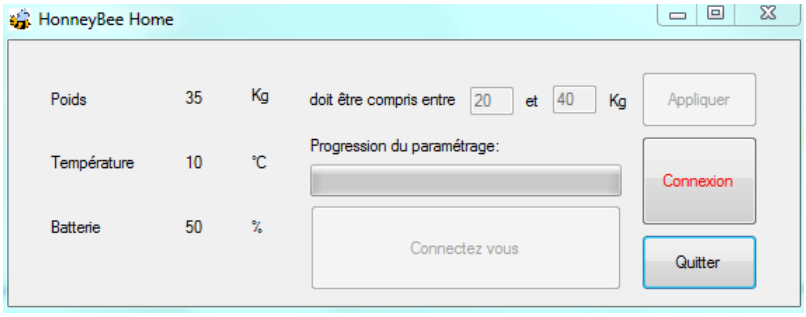
Apres exécution du programme, coté application réceptrice, on obtient ceci :

```
40 40 40

30 30 30

31 31 31
32 32 32

33 33 33

34 34 34

35 35 35

36 36 36
37 37 37

38 38 38

39 39 39

40 40 40

30 30 30
```

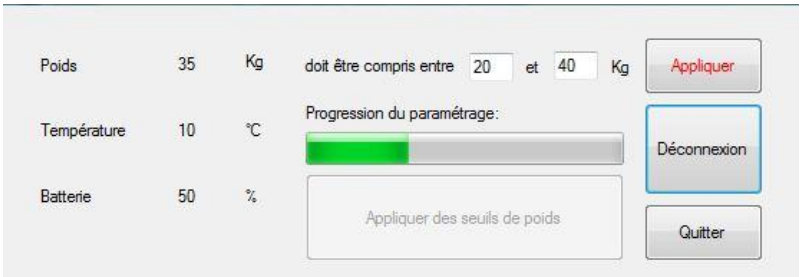
Troisième test unitaire : Créer une interface graphique pour le PC de l'apiculteur

L'apiculteur, n'étant pas informaticien, doit avoir sur son PC une application simple à utiliser, possédant toutes les fonctionnalités que l'apiculteur recherche. J'ai donc mis au point une application graphique pour le poste du client, permettant la lecture des données en temps réel, une gestion de seuils de poids, et une gestion d'alerte en cas de batterie faible. L'application a été conçue dans le but de ne jamais pouvoir faire de mauvaises manipulations de la part de l'apiculteur, et de pouvoir le guider afin qu'il puisse obtenir les données qu'il désire recevoir.

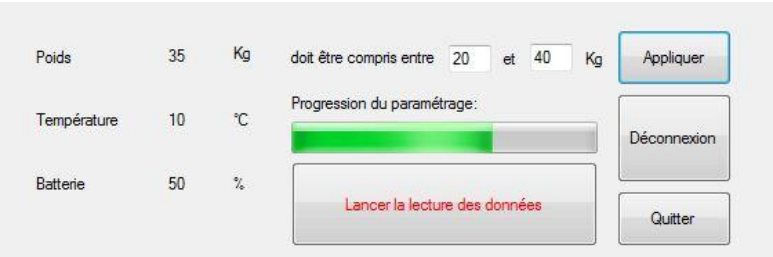
Lors du lancement de l'application, le client voit ceci :

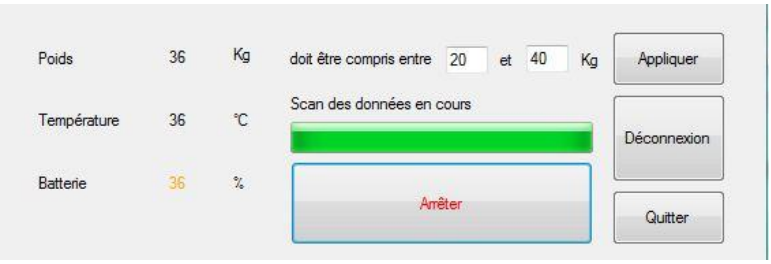


Lorsque le client se connecte, il peut désormais entrer des seuils de poids maximum et minimum pour sa ruche :

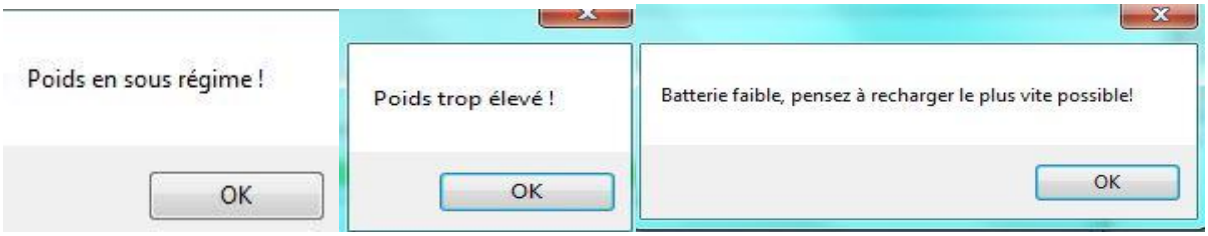


Apres avoir choisi les limites de poids, le client peut enfin lancer la lecture des données, et peut changer les seuils de poids à tout moment :





Cependant, lorsque le seuil de poids maximum est dépassé, le poids minimum est dépassé, ou la batterie est à 20% ou moins, l'utilisateur reçoit ces alertes :



Le code de ce programme contient une classe avec les méthodes identiques des programmes précédents. Cependant, la lecture et l'analyse des données reçues se font grâce à un « timer » qui se lance lors du clic du bouton « lancer la lecture des données », et s'arrête en cliquant à nouveau sur ce même bouton. Ce timer lit et analyse les données toutes les 500 millisecondes, mais cet intervalle peut être modifié.

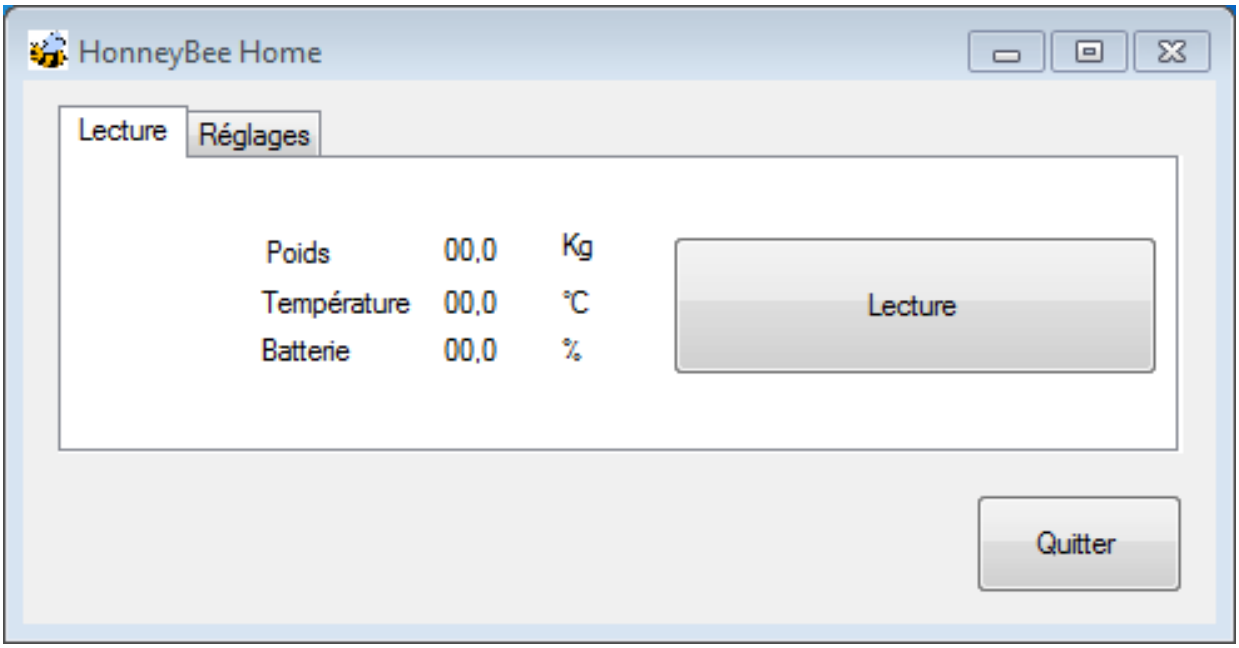
Code du bouton :

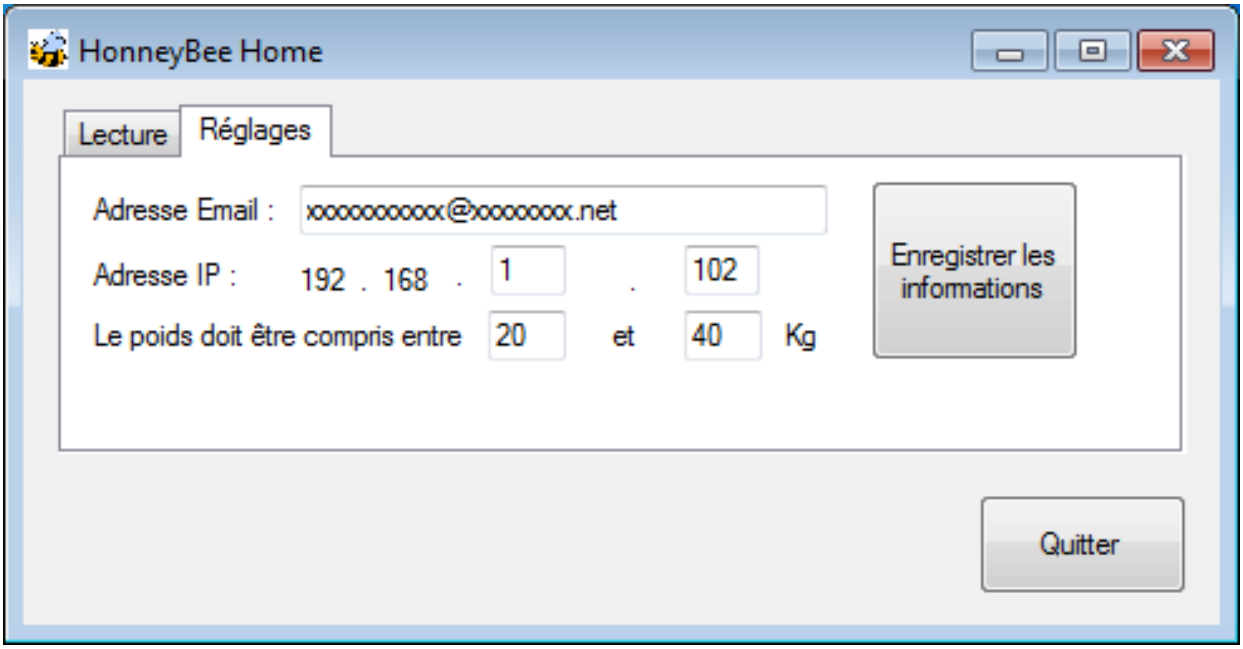
```
if (b_lire.Text == "Lancer la lecture des données")
{
    progressBar1.Value = 100;
    timer1.Start(); //Demarre le timer qui va executer en continu la lecture des données dans le buffer
    b_lire.Text = "Arrêter";
    b_lire.ForeColor = System.Drawing.Color.Red;
}
else
{
    timer1.Stop(); //Arrete le timer et donc arrete la lecture des données du buffer
    b_lire.Text = "Lancer la lecture des données";
    b_lire.ForeColor = System.Drawing.Color.Red;
    progressBar1.Value = 75;
    l_etapes.Text = "Progression du paramétrage";
}
```

Code du timer :

```
private void timer1_Tick(object sender, EventArgs e)
{
    int i;
    l_etapes.Text = "Scan des données en cours";
    string donnees = Xbee.ReceptionSerie();//Récupère les données sous forme "poids temperature batterie"
    if (donnees != "")
    {

        string[] decoupe = donnees.Split(' '); //découpe la trame pour séparer le poids, la température et la batterie
        l_poids.Text = decoupe[0]; // dispose les différents éléments dans les labels qui leur sont affectés
        l_temp.Text = decoupe[1];
        l_batterie.Text = decoupe[2];
        Int32 batteriefaible = 20;
        Int32 batteriemoyenne = 51;
        l_batterie.ForeColor = System.Drawing.Color.Green;
        Int32 poidsmin = Convert.ToInt32(t_poids_min.Text); //Conversion du poids max en entier pour le comparer au poids réel
        Int32 poidsmax = Convert.ToInt32(t_poids_max.Text);//Conversion du poids min en entier pour le comparer au poids réel
        Int32 batterie = Convert.ToInt32(l_batterie.Text); //Conversion de la battrie en entier pour traiter sa valeur
        Int32 poids = Convert.ToInt32(l_poids.Text); //Conversion du poids réel en entier pour le comparer aux seuils
        if (batterie < batteriemoyenne) { l_batterie.ForeColor = System.Drawing.Color.Orange; }
        if (poids < poidsmin) { MessageBox.Show("Poids en sous régime !"); l_poids.ForeColor = System.Drawing.Color.Red; } //Si le poids est en dessous du seuil de poids
        else if (poids > poidsmax) { MessageBox.Show("Poids trop élevé !"); l_poids.ForeColor = System.Drawing.Color.Red; } //Si le poids est au dessus du seuil de poids
        if (batteriefaible > batterie) //Avertissement si la batterie est à 20% ou inférieure
        {
            MessageBox.Show("Batterie faible, pensez à recharger le plus vite possible!");
            l_batterie.ForeColor = System.Drawing.Color.Red;
        }
        for (i = 0; i < 3; i++)
        {
            decoupe[i] = null; //Initialisation du tableau de données
        }


            }
            else
            {
                timer1.Stop();
                MessageBox.Show("Communication impossible entre les deux modules, vérifiez que les connexions soient effectuées.");
                b_lire.Text = "Lancer la lecture des données";
                l_etapes.Text = "Progression du paramétrage";
                b_lire.ForeColor = System.Drawing.Color.Red;
                progressBar1.Value = 75;


            }
        }
    }
}
```

Application finale





L'application finale contient toutes les fonctionnalités imposées dans le cahier des charges. Elle permet à l'apiculteur de recevoir les mails d'alerte depuis son adresse mail, recevoir les données depuis son smartphone deux fois par jour (12 h et 21h).

Contrairement à l'application du troisième test unitaire, cette application peut enregistrer les modifications des réglages (mail, IP, seuils de poids) afin que l'apiculteur puisse rentrer les informations qu'une seule fois. Les dernières modifications seront appliquées et affichés lors des prochains démarrages de l'application.

En appuyant sur le bouton « Enregistrer les informations », les informations précédentes seront effacées du ficher app.config du programme et remplacées par les nouvelles. Bien évidemment, si les informations sont incorrects comme des seuils de poids invalides ou une faute de frappe dans l'adresse ip ou mail, les modifications ne seront pas enregistrées.

Code du bouton « Enregistrer les informations » :

```
adresseipcomplete = "192.168." + t_ip3.Text + "." + t_ip4.Text;
if (VerifMail(t_mail.Text) == true) //Si le format de l'adresse mail est respecté
{
    adressemail = t_mail.Text;
    //on charge la configuration
    System.Configuration.Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
    //On efface la clé voulue dans app.conf
    config.AppSettings.Settings.Remove("mail");
    //On la rajoute dans config.App
    config.AppSettings.Settings.Add("mail", adressemail);
    //On la resauvegarde
    config.Save();
    ConfigurationManager.RefreshSection("appSettings");
    b_reglages.ForeColor = System.Drawing.Color.Black;
}
else
{
    MessageBox.Show("Adresse Email invalide, vérifiez l'orthographe de celle-çi !");
}
if (VerifIP(t_ip3.Text) == true && VerifIP(t_ip4.Text) == true) //si le format des octets de l'IP sont bons
{

    //on charge la configuration
    System.Configuration.Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
    //On efface la clé voulue dans app.conf
    config.AppSettings.Settings.Remove("adresseip");
    //On la rajoute dans config.App
    config.AppSettings.Settings.Add("adresseip", adresseipcomplete);
    //On la resauvegarde
    config.Save();
    ConfigurationManager.RefreshSection("appSettings");
    b_reglages.ForeColor = System.Drawing.Color.Black;
}
```

```
else
{
    MessageBox.Show("Adresse IP invalide, vérifiez la saisie de celle-çi !");
}
try
{
    //Convertion du texte de la boite de texte t_poids_max en entier signé 32 bits
    poids = Convert.ToDouble(l_poids.Text); //Convertion du texte du label l_poids en entier signé 32 bits
    if (Convert.ToInt32(t_poids_min.Text) > Convert.ToInt32(t_poids_max.Text))
    {
        MessageBox.Show("Seuils de poids invalides, le seuil de poids maximum ne peut pas être inférieur au seuil de poids minimum");
    }
    else if (Convert.ToInt32(t_poids_min.Text) == Convert.ToInt32(t_poids_max.Text))
    {
        MessageBox.Show("Seuils de poids invalides, le seuil de poids maximum ne peut pas être égal au seuil de poids minimum");
    }
    else if (Convert.ToInt32(t_poids_max.Text) > 150 || Convert.ToInt32(t_poids_min.Text) < 0)
    {
        MessageBox.Show("Seuils de poids invalides, les seuils de poids ne peuvent être inférieur à 0kg ou supérieur à 150 kg");
    }
    else if (isnumeric(t_poids_max.Text) == false || isnumeric(t_poids_max.Text) == false)
    {
        MessageBox.Show("Seuils de poids invalides, veuillez entrer des nombres et UNIQUEMENT des nombres s'il vous plait !");
    }
    else
    {

        System.Configuration.Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
        //On efface la clé voulue dans app.conf
        config.AppSettings.Settings.Remove("poidsmin");
        //On la rajoute dans config.App
        config.AppSettings.Settings.Add("poidsmin", t_poids_min.Text);
        //On efface la clé voulue dans app.conf
        config.AppSettings.Settings.Remove("poidsmax");
        //On la rajoute dans config.App
        config.AppSettings.Settings.Add("poidsmax", t_poids_max.Text);
        //On la resauvegarde
```

```
                config.Save();
                ConfigurationManager.RefreshSection("appSettings");
                poidsmin = Convert.ToInt32(ConfigurationManager.AppSettings["poidsmin"]);
                poidsmax = Convert.ToInt32(ConfigurationManager.AppSettings["poidsmax"]);
            }
        }
        catch
        {
            MessageBox.Show("Seuils de poids invalides"); //Message d'erreur en cas de saisie invalide des seuils de poids
        }

    }
```

De plus, le nombre de boutons a été réduit. Le bouton « Lecture » permet au programme de s'assurer de la bonne connexion entre le module et le PC, et permet de lire les données si la connexion est effectuée. Ce bouton exécute les tâches du bouton « lecture des données » et « Connexion » de l'application du test unitaire :

```
if (b_connect.Text == "Lecture")
{
    try
    {
        connect = Xbee.ConnecterPortSerie("COM20", "9600"); //Connexion au port COM20 paramétré au préalable avec une vitesse de 9600 bauds
    }
    catch
    {
        MessageBox.Show("Module sans fil récepteur non détecté, vérifiez que la connexion soit effectuée.");
    }
    if (connect == true)
    {
        timer1.Start(); //Demarre le timer qui va executer en continu la lecture des données dans le buffer
        b_connect.Text = "Arrêter";
    }
}
else
{
    connect = Xbee.DeconnecterPortSerie();
    timer1.Stop(); //Arrete le timer et donc arrete la lecture des données du buffer
    if (connect == false)
    {
        b_connect.Text = "Lecture";
    }
}
```

Les mails sont envoyés à l'adresse mail rentrée par l'utilisateur et les données à l'adresse IP du téléphone de l'utilisateur. Un mail est envoyé en cas de seuil de poids dépassé ou de batterie faible par session, afin que l'utilisateur ne se fasse pas spammer de mails. Les données sont envoyées à 12h et 21h (heure du PC).

Une classe C_Mail a été créée afin de gérer l'envoi des mails. Cette classe contient un constructeur et une méthode « Envoi ».

Constructeur :

```
public C_Mail(string mailexp, string serversmtp, string portsmtp, string usersmtp, string pwdsmtp, bool sslsmtp)
{
    _mailexp = mailexp;
    _serversmtp = serversmtp;
    _portsmtp = Convert.ToInt32(portsmtp);
    _usersmtp = usersmtp;
    _pwdsmtp = pwdsmtp;
    _smtpssl = sslsmtp;
}
#endregion
```

mailexp correspond à l'adresse mail de l'expéditeur.

serversmtp correspond au serveur SMTP utilisé pour envoyer notre mail (dans notre cas, nous utiliserons celui de gmail).

portsmtp correspond au port du serveur SMTP utilisé (port 587).

usersmtp correspond au nom d'utilisateur SMTP (identique à l'adresse mail de l'expediteur).

pwdsmtp correspond au mot de passe de l'adresse mail de l'expéditeur.

Sslsmtp correspond à l'état (activé ou désactivé) du SSL, un protocole de sécurisation des échanges sur Internet. Il sera désactivé dans notre cas.

Méthode « Envoi » :

```
public bool Envoi(string destinataire, string usern, string userp, string body)
{
    try
    {

        MailMessage mail = new MailMessage();
        mail.From = new MailAddress(_mailexp);
        mail.To.Add(new MailAddress(destinataire));
        mail.Subject = "Alerte Honey Bee";
        mail.Body = body;
        SmtpClient SmtpServer = new SmtpClient();
        SmtpServer.Host = _serversmtp;
        SmtpServer.Port = _portsmtp;
        SmtpServer.UseDefaultCredentials = false;
        SmtpServer.DeliveryMethod = SmtpDeliveryMethod.Network;
        SmtpServer.Credentials = new NetworkCredential(_usersmtp, _pwdsmtp);
        SmtpServer.EnableSsl = _smtpssl;
        SmtpServer.Send(mail);
        return true;
    }
    catch { return false; }
}
```

Destinataire correspond à l'adresse mail du destinataire.

Usern correspond à l'adresse mail de l'expéditeur.

Userp correspond au mot de passe de l'adresse mail de l'expéditeur.

Body correspond au contenu texte du mail.

Le Credential permet de définir les informations d'authentification à utiliser.

Exemple d'envoi de Mail en cas de poids inférieur au seuil de poids minimum :

```
if (poids < poidsmin)
{
    l_poidsalerte.Text = "TROP BAS"; //Si le poids est en dessous du seuil de poids minimum, alors message d'erreur
    l_poidsalerte.Visible = true;
    l_poids.ForeColor = System.Drawing.Color.Red;
    if (mailpoids == false) //si aucun mail n'a été envoyé depuis le lancement du programme
    {
        try { email.Envoi(ConfigurationManager.AppSettings["mail"],
            "honeybeeparticulier@gmail.com",
            "honeybee201557",
            "Aujourd'hui, le " +DateTime.Now.ToString("dd/M/yyyy hh:mm") + ", le poids est en sous régime (" + poids + " kg)");
            mailpoids = true; } //envoie un mail d'alerte à l'adresse enregistrée
        catch { MessageBox.Show("Echec de l'envoi du mail, vérifiez votre connexion ou l'adresse Email entrée"); }
    }
}
```

Pour cette application, nous utilisons désormais des Regex. Regex est une classe qui permet de vérifier qu'une chaine de caractères correspond à un modèle prédéfini, et est indispensable pour s'assurer que les réglages rentrés par l'utilisateur, et les données reçues par le module récepteur soient corrects et correspondent à ce que le programme exige.

Méthode de vérification de l'adresse mail :

```
public bool VerifMail(string adresse)
{
    System.Text.RegularExpressions.Regex myRegex = new Regex(@"^([\S][^@]+)@([a-zA-Z]+)\.([a-zA-Z]+)$");
    // verifie que l'adresse est sous format: tout caractere alpha numerique sauf @ + @ + chaine de caractere + point + chaine de caractere
    return myRegex.IsMatch(adresse); // retourne true ou false selon la vérification
}
```

La méthode retourne « True » si le format et respecté, et « False » si le format n'est pas respecté.

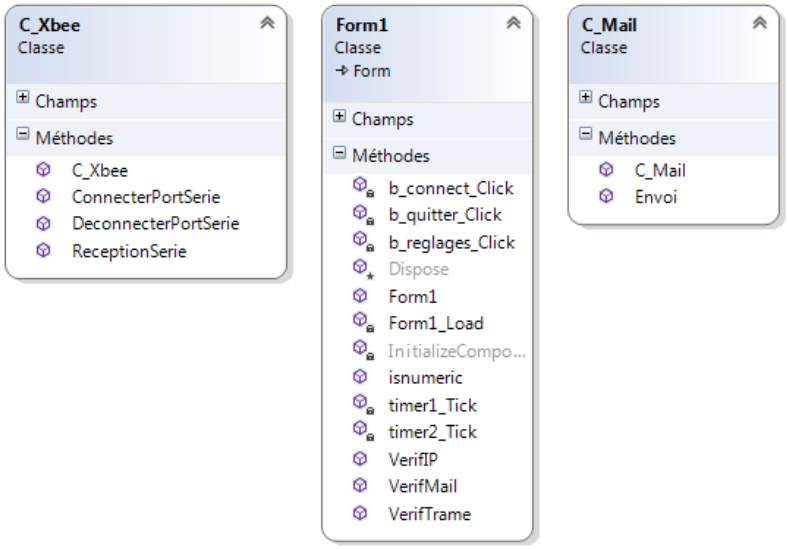
Méthode de vérification d'un octet d'adresse IP :

```
{
    System.Text.RegularExpressions.Regex myRegex = new Regex(@"^\d{3}|\d{2}|\d{1}$");
    //Verifie que l'octet de l'adresse ip est un, doux ou trois chiffres
    if (myRegex.IsMatch(data)) //si le format est respecté
    {
        if (isnumeric(data) == true) //Si la chaine est bien un nombre
        {
            if (Convert.ToInt16(data) > 0 && Convert.ToInt16(data) < 255) // si le nombre < 255
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
```

Vérification du bon format des données reçues :

```
{
    System.Text.RegularExpressions.Regex myRegex = new Regex(@"^\d{2}\.\d\s\d{2}\.\d\s\d{2}\.\d$");
    //verifie si la trame est sous la forme: deux chiffres + point + chiffre + espace + deux chiffres
    //+ point + chiffre + espace + deux chiffres + point + chiffre
    return myRegex.IsMatch(data); // retourne true ou false selon la vérification
}
```

Nous obtenons donc au final ce diagramme de classe :



Conclusion

Ce projet a été pour moi très bénéfique. Il m'a permis de travailler en équipe et de pouvoir gérer mon temps, afin que le projet puisse être déployé dans les temps impartis. Ce projet m'a permis d'enrichir mes connaissances dans le domaine de la programmation en c#, ou j'ai dû à de nombreuses reprises chercher les solutions par moi-même et m'aider à devenir de plus en plus autonome.