Rapport d'avancement

BMONS

Beehive Monitoring System

Rédigé par :

Alice Danckaers
Benoît Raymond
Etiene Dalcol
Nicolas Van-Nhân Nguyen
Tao Zheng
Armand Sellier

Sous la direction de :

Olivier Reynet

ENSTA Bretagne

Option Systèmes Perception Information Décision

Première impression, décembre 2014

Sommaire

Remerciements

> La gratitude est non seulement la plus grande des vertus, mais c'est également la mère de tous les autres.
>
> Emil Cioran

L'équipe du projet BMONS (BeeHive Monitoring System) souhaiterait remercier monsieur O.Reynet, responsable de l'UV 3.4 et auteur de notre sujet ainsi que notre encadrant IS, monsieur B.CLEMENT pour leurs conseils, leur soutient et leur disponibilité. Ils ont su nous guider tout au long du projet notamment sur le plan technique et architectural de notre système.

Nous tenons également à remercier monsieur F.Singhoff, professeur à l'université de Bretagne Occidentale à Brest dans le domaine des systèmes embarqués et apiculteur, qui a accepté de prendre part à notre projet en tant que "client" en partageant ainsi son expérience de l'apiculture. Il a aussi mis à notre disposition une de ses ruches afin que nous puissions nous rendre compte des dimensions et de l'espace disponible pour installer notre système.

Première partie

Introduction au projet

Chapitre 1

Formulation initiale du projet

1.1 Contexte

BeeHive Monitoring System (BMONS) est un projet qui a pour but d'aider les apiculteurs. Il s'agit de leur proposer un système de surveillance et de détection peu onéreux afin de prodiguer les meilleurs soins au meilleur moment aux ruches qui en ont besoin et d'éviter les vols.

En effet, les abeilles sont vitales à l'équilibre écologique. Einstein avait même dit : " Si l'abeille disparaît, l'humanité en a pour quatre ans à vivre ". Sans elles 84 % des espèces végétales cultivées pour l'alimentation disparaitraient. Or les abeilles sauvages sont aujourd'hui rares et l'espèce ne survivrai pas sans l'aide des apiculteurs. Ainsi le travail de ces derniers est crucial non seulement pour assurer la production de miel mais aussi pour la sauvegarde de l'environnement. Cependant, ces dernières années, les apiculteurs ont été confrontés à de nombreux problèmes et nous sommes aujourd'hui face à une diminution du nombre d'abeilles telle que la production annuelle européenne de miel est quatre fois moindre que celle qu'il y a vingt ans.

Pour aider à la résolution de ce problème, nous voulons donc créer un système capable d'aider l'apiculteur dans son travail et de ce fait combattre la disparition des abeilles.

1.2 Expression initiale du besoin

Après avoir discuté avec plusieurs apiculteurs, nous avons pu identifier leurs besoins et déterminer de quelle manière nous pouvons les aider. Ainsi l'objectif de ce système est tout d'abord de donner accès à l'apiculteur à des informations clés sur la ruche sans que celui-ci n'ai à se déplacer, ni à ouvrir les ruches. En effet l'ouverture de la ruche perturbe les abeilles et elle n'est pas possible en hiver à cause des températures trop basses. De plus les ruches sont souvent disposées dans des ruchers éloignés les uns des autres, ce qui complique le travail de l'apiculteur. Les informations nécessaires seraient : la température dans et en dehors de la ruche, le poids, l'humidité et les sons de la ruche. Mais le système devra aussi alerter l'apiculteur quand la sécurité de la ruche est compromise, pour permettre une action rapide destinée à sauver la colonie.

Le système BMONS est donc composé de deux parties distinctes. La première consiste en un élément embarqué dans la ruche qui consomme un minimum d'énergie et qui mesure les paramètres clés. Les données de cet élément embarqué sont transmises via un transmetteur sans fils à un serveur qui constitue la deuxième partie du système. Il donne accès à l'apiculteur aux différentes mesures effectuées dans et autour des ruches. Il envoie également des alertes de sécurités à l'apiculteur si besoin.

Chapitre 2

État de l'art

En effectuant nos recherches sur le sujet nous avons trouvé beaucoup d'informations sur les abeilles et le travail des apiculteurs en général, ainsi que des systèmes "maison" développés par des particuliers pour surveiller un peu mieux leurs ruches. Cependant nous avons également découvert l'existence de quatre projets similaires au notre : trois projets en cours ayant une approche OpenSource et un projet commercial déjà développé. Ce dernier appartient à la société anglaise Arnia. Ce système est décrit [?] comme permettant à l'utilisateur d'avoir des informations sur une ou plusieurs ruches telles que la température, l'humidité et l'intensité acoustique dans la ruche ainsi que la température du couvain. Les apiculteurs peuvent ensuite visualiser ces informations sur une partie sécurisée du site internet d'arnia. Ils peuvent également comparer les informations et évolution d'une ou plusieurs ruches, comme on peut le voir sur la figure 2.1.

FIGURE 2.1 – Interface du système d'arnia : comparaison des données d'une ruche

L'un des projets OpenSource est développé par Ken Meyer sur le site hackaday [?] et consiste à mesurer la température, l'humidité et le poids d'une ruche. Ce projet est encore en développement et plusieurs prototypes ont déjà été testés.

Il existe également un autre projet OpenSource sur le sujet. Il s'agit de Bzzz [?], développé par le Fablab de Lannion. Ce système propose une supervision de la température intérieure, de la luminosité extérieure et la masse d'une seule ruche via un envoi de données périodique par SMS et par visualisation des données sur

un portail en ligne. L'utilisateur pourra également configurer des alertes via le portail.

Enfin le dernier système existant que nous avons trouvé a été développé conjointement par le Fablab de Barcelone et Open Tech Collaborative, Denver, USA [?]. Ce projet OpenSource, appelé Open Source BeeHive, ne s'adapte pas aux ruches classiques mais propose une architecture simple qui permet de construire sa propre ruche entièrement, comme on peut le voir sur les figures 2.3 et 2.2.

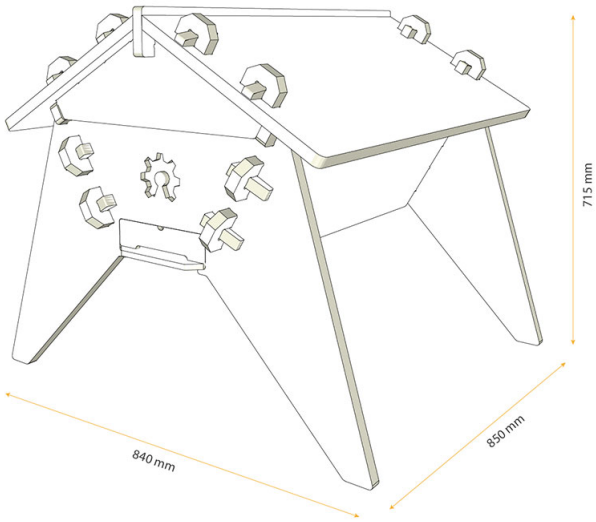

FIGURE 2.2 – Modèle de ruche Open Source Beehive

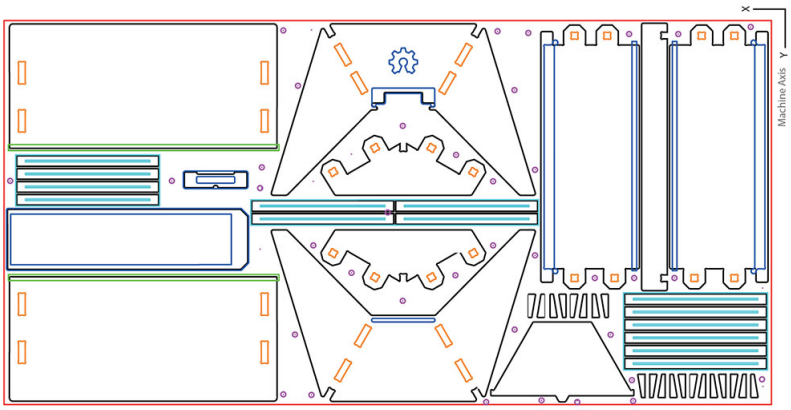

FIGURE 2.3 – Plan de la ruche Open Source Beehive

Ensuite un kit de capteurs à installer permet de mesurer la température, l'humidité, l'intensité acoustique et le nombre d'abeilles via un capteur infrarouge. Les données seront ensuite visibles de tous sur la plateforme Smart Citizen.

Nous n'avons pas détaillé ici tous les projets que nous avons trouvés du fait de leur grand nombre. Cependant nous nous sommes intéressés à ceux qui présentaient un intérêt pour le système que nous voulons développer.

Concernant le traitement sonore, le site internet Beesource comporte une description complète du système Apidictor [?]. Ce système permet de filtrer le son

émit par une ruche pour prévoir un essaimage. Un schéma du montage électrique ainsi qu'un texte expliquant quelles fréquences sont surveillées et quelles méthodes ont été utilisées pour vérifier le bon fonctionnement de ce système sont également présents. Nous ne réutiliserons pas le montage proposé, car le traitement du signal sonore se fera de manière informatisée, en revanche le travail effectué pour savoir quelles fréquences sont à surveiller nous sera utile.

A propos de la transmission de l'information à l'apiculteur, deux moyens sont très souvent utilisés : un site internet sécurisé et une application pour smart-phone. Concernant le site internet, celui de la société Arnia nous a semblé complet et clair. Pour l'application smart-phone, l'entreprise américaine B-Ware en a mis une au point mettant bien en avant le tableau de bord, l'affichage de l'historique des paramètres de la ruche et la personnalisation des alertes en laissant même l'apiculteur définir lui même les seuils de déclenchement des avertissements comme nous pouvons le voir sur la figure 2.4.

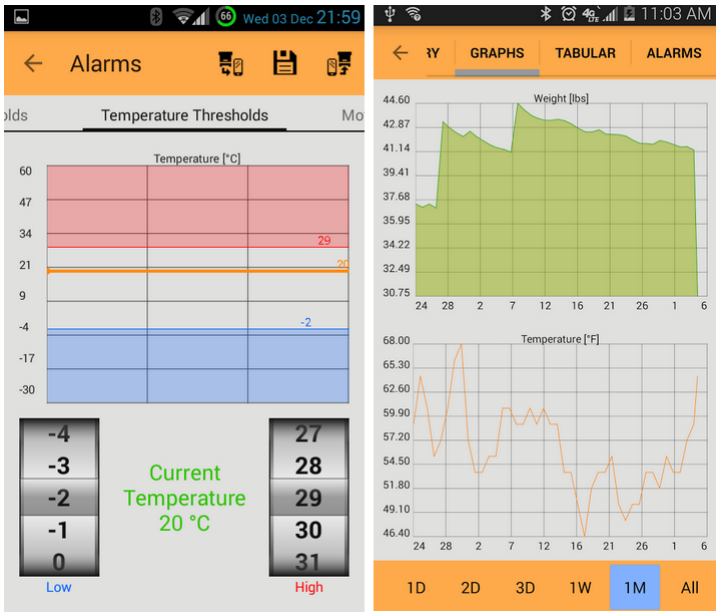

FIGURE 2.4 – Capture d'écran de l'application B-Ware

La société BeeWise d'origine française a développé un système d'alimentation écologique à l'aide de panneaux solaires connectés au boitier regroupant les cartes électroniques pour le traitement des données et de transmission [**?**]. Il suffirai juste de la fixer sur le couvercle de la ruche en prenant soin d'étudier l'orientation adéquate au préalable comme sur la figure 2.5. Cette solution semble être la plus répandue car elle est également utilisée par les développeurs de l'ApiScan, un système de comptage de d'abeilles installé sur la planche d'envol pour minimiser la gène occasionnée.

Les recherches sur l'exploitation des ruchers nous a mené à comprendre que l'un des principaux risques pour les apiculteurs est le vol de ruche. Cette pratique s'est très largement répandue ces dernières années. C'est pour cela que la société Apimiel met à disposition un système GPS pour permettre à l'apiculteur

FIGURE 2.5 – Système de panneau solaire installé sur une ruche

de connaitre en temps réel la position de ses ruches [**?**]. Au moindre déplacement, l'apiculteur recevra une alerte SMS. Compte tenu de sa taille, le traceur GPS devra se trouver à l'extérieur de la ruche mais il existe aussi des versions plus petites que l'on peut directement placer dans la ruche comme sur la figure 2.6. C'est ce que Michel BOCQUET, ingénieur agronome, propose sur son blog. Néanmoins, son objectif final est d'étudier le comportement des abeilles pour mieux comprendre celui de notre environnement et non commercialiser son projet aux apiculteurs. Le coût du matériel n'est donc pas pris en compte.

FIGURE 2.6 – Système GPS placé dans la ruche

Deuxième partie

Cahier des charges fonctionnel

Chapitre 3

Ingénierie des exigences

3.1 Approche Top-Down

Dans cette partie nous allons analyser notre système avec une approche Top-Down. Cela signifie que nous adopterons une démarche de conception descendante. Pour cela nous avons tracé le diagramme "bête à cornes", que l'on peut voir sur la figure 3.1. Il permet de représenter graphiquement l'expression du besoin. Comme on peut le voir sur le diagramme, le système BMONS rend service aux apiculteurs en agissant sur une ou plusieurs ruches. Il a pour but d'aider la surveillance d'un rucher et d'avertir l'apiculteur en cas de problème.

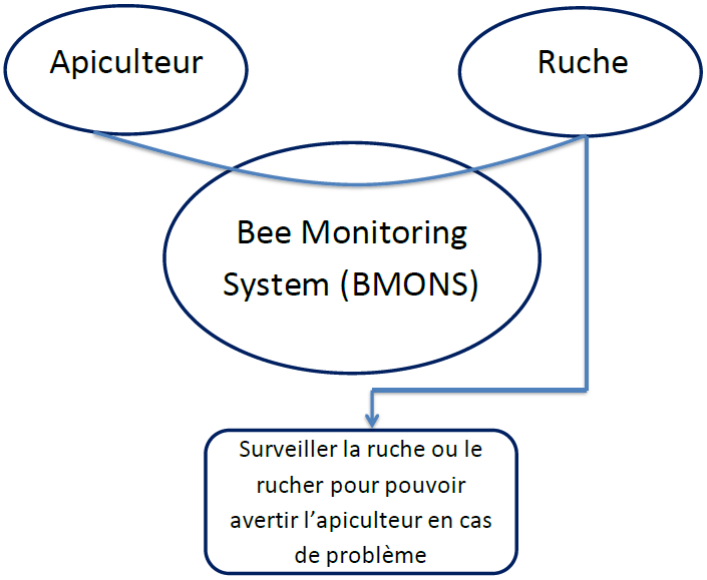

FIGURE 3.1 – Diagramme "bête à cornes" du système BMONS

Le diagramme pieuvre, 3.2 et 3.3, nous permet ensuite de faire apparaître les fonctions principales du système. On peut aussi y retrouver les fonctions de services et de contraintes.

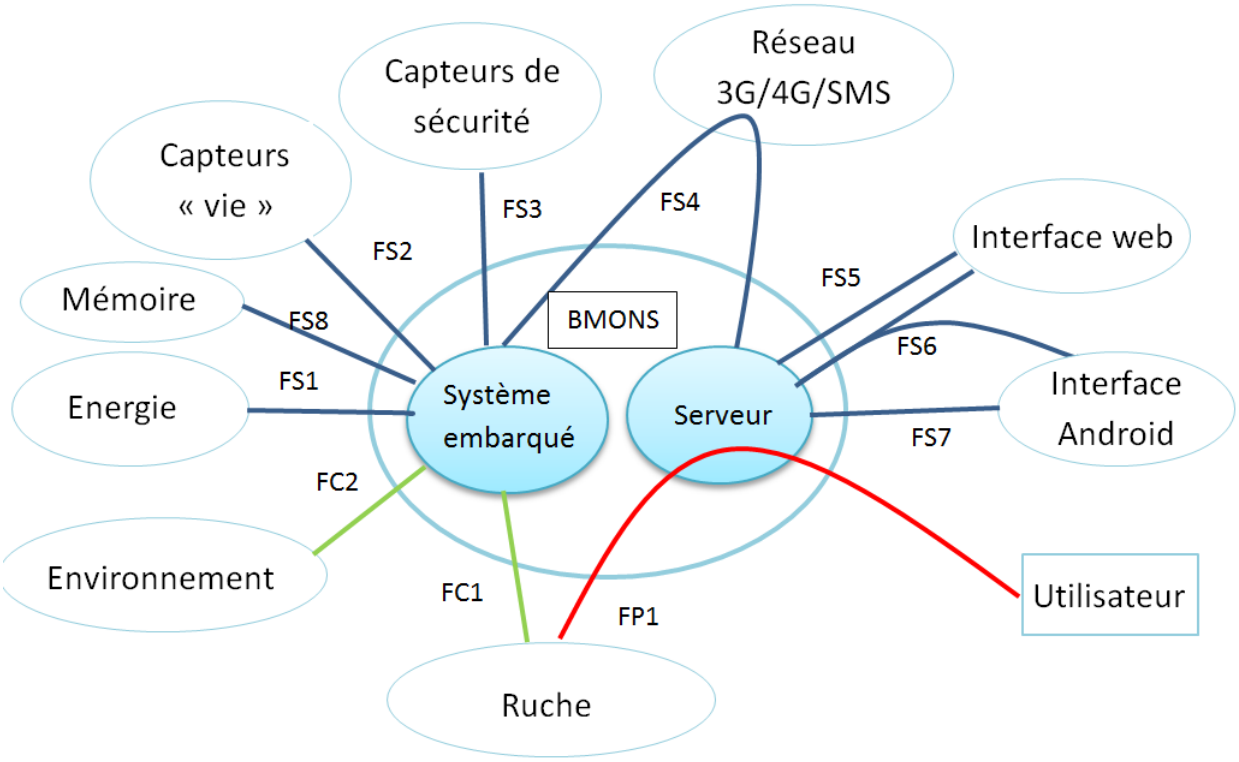

FIGURE 3.2 – Diagramme pieuvre du système BMONS

FP1 : Informer l'apiculteur de l'état de la ruche

FC1 : Etre adapté aux dimensions de la ruche
FC2 : Résister à l'environnement de la ruche

FS1 : Etre autonome en énergie
FS2 : Acquérir les données relatives à la vie de la ruche
FS3 : Acquérir les données relatives à l'intégrité de la ruche
FS4 : Communiquer avec le serveur via le réseau 3G/4G/SMS
FS5 : Proposer une IHM web
FS6 : Proposer un accès via l'API
FS7 : Proposer une IHM Android
FS8 : Conserver les données localement

FIGURE 3.3 – Légende diagramme pieuvre du système BMONS

3.2 Approche Bottom-Up

Nous allons maintenant adopter la démarche inverse, mais néanmoins complémentaire, de l'approche Top-Down. Il s'agit de l'approche Bottom-Up. C'est une démarche de conception ascendante qui va nous permettre d'avoir une vision plus globale du système. On peut voir sur 3.4 et 3.5 les exigences issues de cette analyse.

Identifiant	Type	Expression de l'exigence	Performance	Test	Groupe logique	Fonction
exi_01	contrainte	Supporter les variations de température	de -15 à 60°C		GLE_Capteur_info	Résister au milieu ambiant de la ruche
exi_02	contrainte	Ne pas être endommagé par la cire et la propolis		Plonger dans le miel		
exi_03	contrainte	Ne pas être endommagé par les abeilles		Mise en situation		
exi_04	contrainte	Ne pas être endommagé par l'apiculteur lors de ses interventions sur la ruche				
exi_05	contrainte	Supporter les variations d'humidité	de 0 à 100 %			
exi_06	contrainte	Ne pas nuire aux abeilles				Perturber au minimum
exi_07	contrainte	Etre adapté aux dimensions		implantation dans un prototype		
exi_08	contrainte	Respecter la reglementation alimentaire				Adapter à la legislation
exi_09	service	Mesurer le poids de la ruche	[0 ; 150] résolution 100g fréquence 2/jour			Mesurer les paramètres propres relatifs au fonctionnement de la ruche
exi_10	service	Relever le bruit dans la ruche				
exi_11	service	Mesurer la température dans la ruche	[-5 ; 50] résolution 1°C fréquence 1/heure			
exi_12	service	Mesurer l'humidité dans la ruche	[0 ; 100] résolution 1% fréquence 1/heure			
exi_13	service	Vérifier la présence d'abeilles	fréquence 1/heure			
exi_14	service	Relever la température extérieure	[-15 ; 50] résolution 1°C fréquence 1/heure			
exi_15	service	Détecter un choc			GLE_Capteur_sécu	Mesurer les paramètres liés à la sécurité et à l'intégrité de la ruche
exi_16	service	Détecter l'ouverture de la ruche				
exi_17	contrainte	Supporter les variations de température	[-15 ; 50] (° C)			Résister au milieu extérieur
exi_18	contrainte	Supporter les variations d'humidité	[0 - 100] %			
exi_19	contrainte	Résister aux chocs				
exi_20	contrainte	Etre étanche		plonger dans l' eau		

FIGURE 3.4 – Exigences issues de l'approche Bottom-Up (1/2)

Identifiant	Type	Expression de l'exigence	Performance	Test	Groupe logique	Fonction
exi_21	service	Transmettre les données par ondes radio	3G ou 4G		GLE_Boitier_Exterieur	Gérer les données
exi_22	contrainte	pouvoir stocker les données localement				
exi_23	contrainte	Etre relié aux capteurs extérieurs				Récupérer les données
exi_24	contrainte	Etre relié aux capteurs intérieurs				
exi_25	contrainte	Supporter les variations de température	[-15 ; 50] (° C)			Résister au milieu extérieur
exi_26	contrainte	Supporter les variations d'humidité	[0 - 100] %			
exi_27	contrainte	Résister aux chocs				
exi_28	contrainte	Etre étanche		plonger dans l' eau		
exi_29	service	Avoir 2 modes de fonctionnement (été, hivers)				S'adapter au besoins de l'apiculteur
exi_30	contrainte	Déclancher des mesures à intervalles réguliers				
exi_31	contrainte	être autonome en énergie				S'alimenter en énergie
exi_32	service	Avertir l'apiculteur en cas de diminution significative du poids de la ruche			GLE_software	Informer l'apiculteur
exi_33	service	Avertir l'apiculteur en cas de choc				
exi_34	service	Avertir l'apiculteur en cas de détection d'essaimage				
exi_35	service	Afficher les données (graphique)				
exi_36	service	Informer l'apiculteur sur l'alimentation électrique et l'état des capteurs				
exi_37	service	Donner la possibilité aux utilisateurs de configurer les alertes				
exi_38	service	Donner la possibilité aux utilisateurs de configurer les informations à afficher				
exi_39	service	Donner la possibilité aux utilisateurs de configurer le niveau de confidentialité				
exi_40	service	Pouvoir accéder rapidement à l'information				
exi_41	service	Déterminer la position de la grappe				Traiter les donnees reçues
exi_42	service	Reconnaitre le chant des futurs reines				
exi_43	service	Identifier les prémices de l'essaimage				
exi_44	service	Identifier les bruits caractéristiques dans la ruche				
exi_45	contrainte	Restreindre l'accès aux données				Gérer les données
exi_46	service	Extraire les données				
exi_47	service	Conserver un historique des données				

FIGURE 3.5 – Exigences issues de l'approche Bottom-Up (2/2)

3.3 Fonctions métiers du système

Après avoir réalisé une approche du point de vue "concepteur" du système, nous allons maintenant nous intéresser à la formulation des fonctions qu'un apiculteur souhaiterai avoir pour pouvoir suivre l'évolution de son rucher. Ces fonctions métiers ont été discutées avec notre client, monsieur Singhoff. Elles sont représentées sur les figures **??** en annexe dont la figure 3.6 est un extrait.

	expresion de la fonction metier	exigences	Performance
exi met 1	suivre la miellée	Mesurer le poids de la ruche	[0 ; 150] résolution 100g fréquence 2/jour
		Avertir l'apiculteur en cas de diminution significative du poids	
		Transmettre les données par ondes radio	3G ou 4G
		Pouvoir stocker les données localement	
		Déclancher des mesures à intervalles réguliers	
exi met 2	détecter la presence d'abeille dans la ruche	Relever le bruit dans la ruche	
		Pouvoir stocker les données localement	
		Déclancher des mesures à intervalles réguliers	
		Transmettre les données par ondes radio	3G ou 4G
exi met 3	suivre le traffic	Mesurer le poids de la ruche	
		Relever le bruit dans la ruche	
		Pouvoir stocker les données localement	
		Déclancher des mesures à intervalles réguliers	
		Transmettre les données par ondes radio	3G ou 4G

FIGURE 3.6 – Fonctions métiers du système (1/4)

Chapitre 4

Architecture fonctionnelle

L'étude de la spécification fonctionnelle trois axes a permis d'établir l'architecture fonctionnelle du système qui est représentée sur la figure 4.1. Ce schéma résume les interactions entre chaque partie : Bee Monitor qui regroupe l'ensemble des capteurs, la carte Arduino ainsi que la carte SSD pour l'enregistrement local des données, le module de transmission et le système d'alimentation rendant notre projet autonome en énergie et le serveur. Chaque acteur interagissant avec le système est également représenté : Les abeilles/ruche, l'apiculteur et l'environnement.

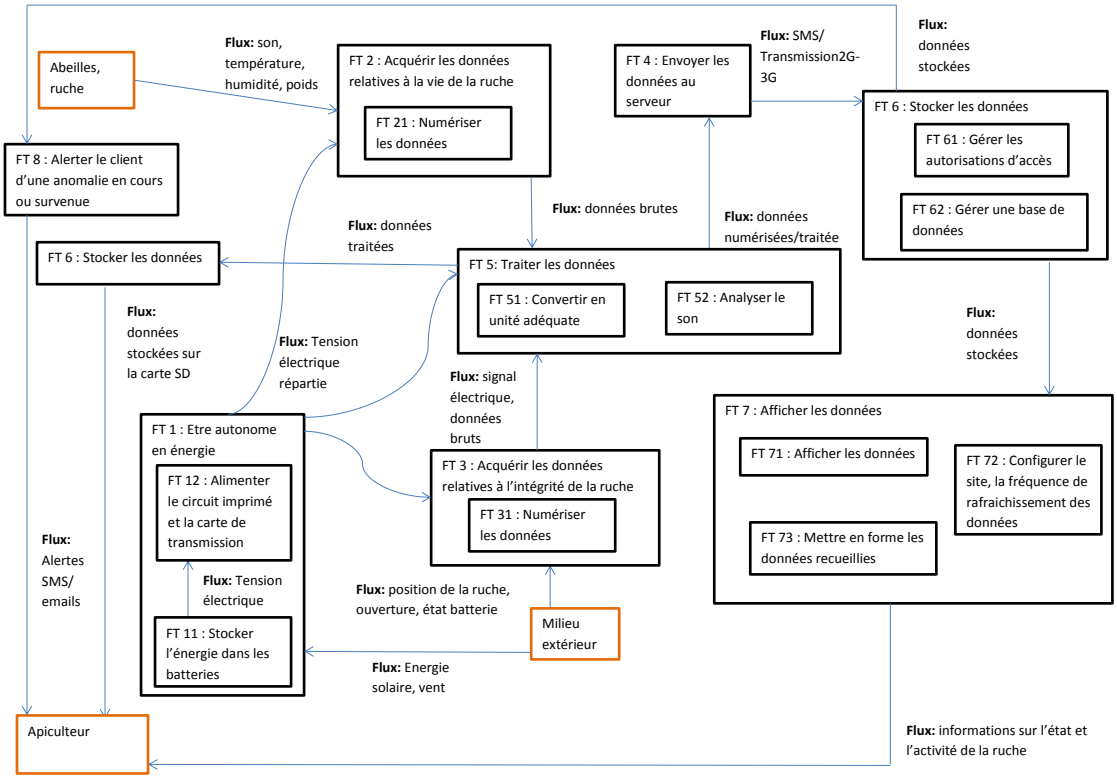

FIGURE 4.1 – Architecture fonctionnelle

Chapitre 5

Architecture physique

5.1 Architecture physique matérielle

La solution technique que nous avons choisie se présente sous la forme d'un cadre en bois léger (type contreplaqué) dans lequel les capteurs sont incrustés. On peut voir sur la vue du dessus du cadre en coupe,figure 5.3, la position des différents capteurs.Sur deux cotés opposés il y aura 4 capteurs de pression et 3 capteurs de température, voir figure 5.1. Puis sur un autre coté le capteur d'humidité, le microphone et la sortie des fils, voir figure 5.2. Enfin sur le dernier côté il y aura uniquement un microphone. Les capteurs seront placés entre deux épaisseurs de bois préalablement travaillées et dépasseront si nécessaire du cadre. Les fils de connexion seront rassemblés et sortiront à un seul endroit du cadre. Ils seront placés dans une gaine protectrice et se connecteront au boitier extérieur.

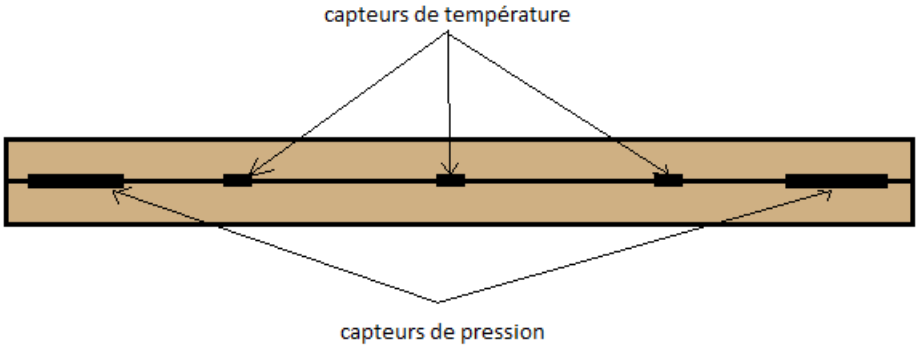

FIGURE 5.1 – Schéma du cadre de mesure. Coupe vue de coté

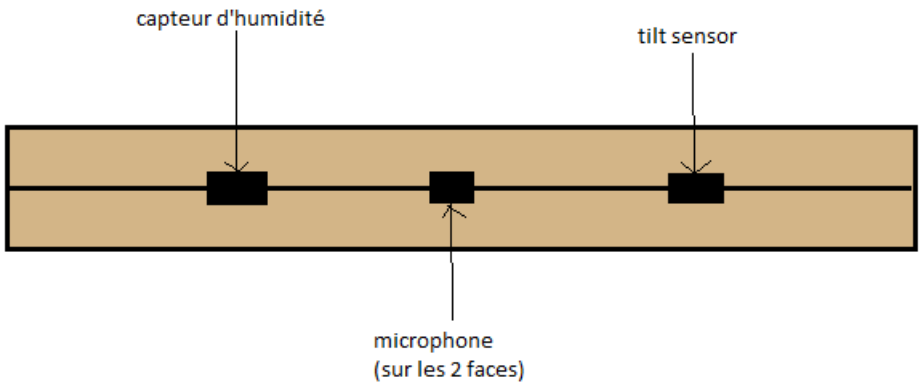

FIGURE 5.2 – Schéma du cadre de mesure. Coupe vue de coté

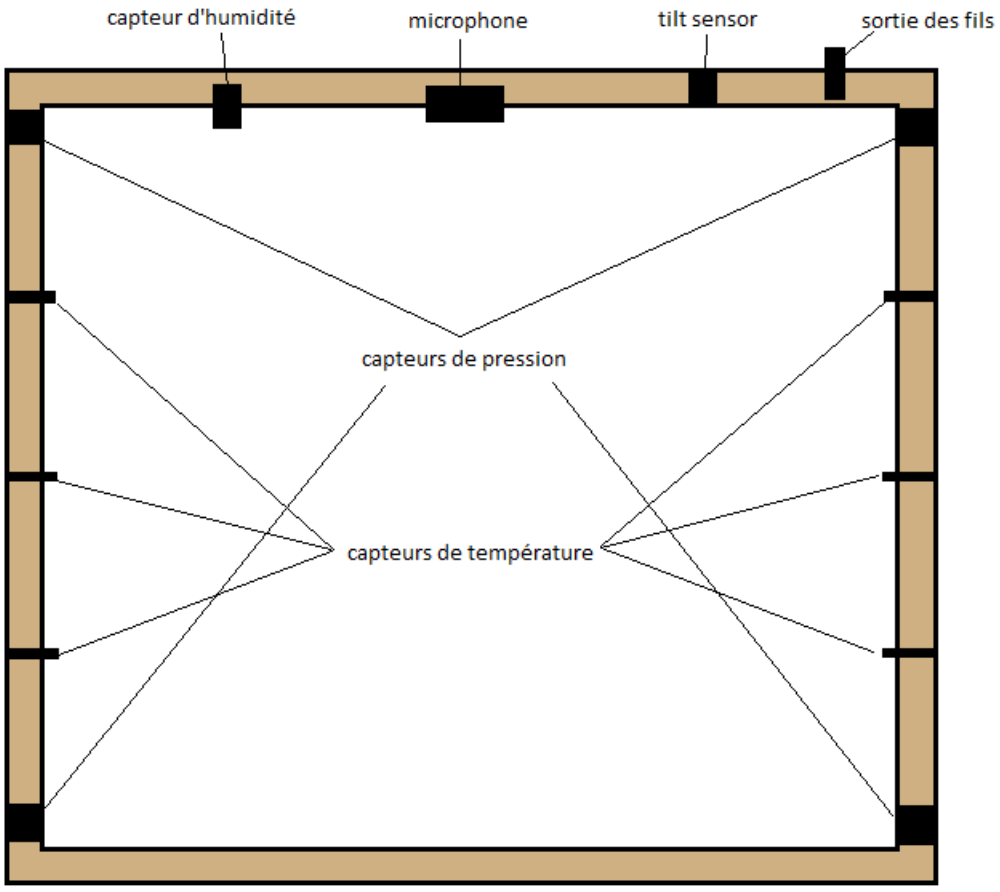

FIGURE 5.3 – Schéma du cadre de mesure. Coupe vue du dessus

5.2 Architecture physique logicielle

Dans cette partie nous allons analyser l'architecture logicielle côté logiciel. Elle est divisée en deux parties : Serveur et Arduino. La partie serveur comprend tout qu'est lié au développement du site et les scripts qui contrôlent les logiciels, les backups et l'obtention de données. La partie arduino comprend tous les capteurs et modules, l'énergie et les scripts de contrôle des capteurs et manipulation des données. Ceci est résumé dans le diagramme ci-dessous.

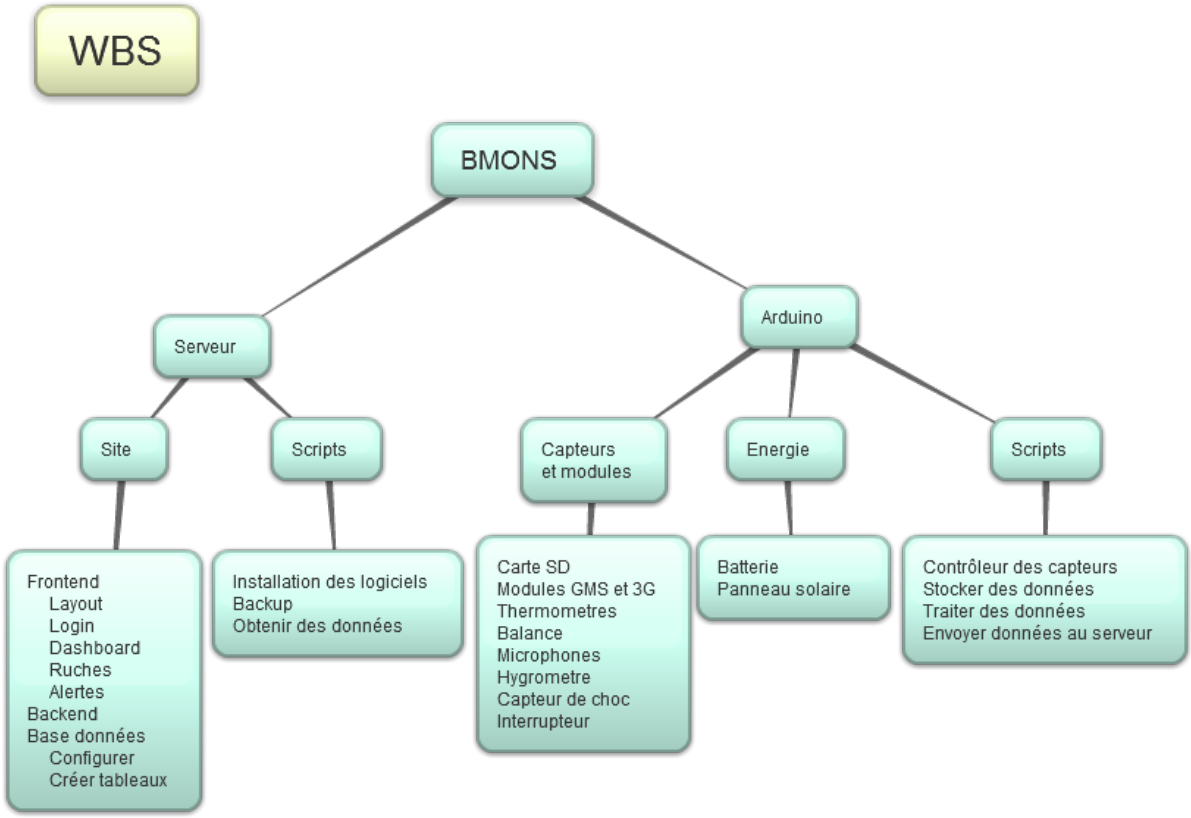

FIGURE 5.4 – Architecture physique logicielle du système BMONS

5.2.1 Base de données

L'organisation de ma base de données se fait comme expliqué sur la figure 5.5.

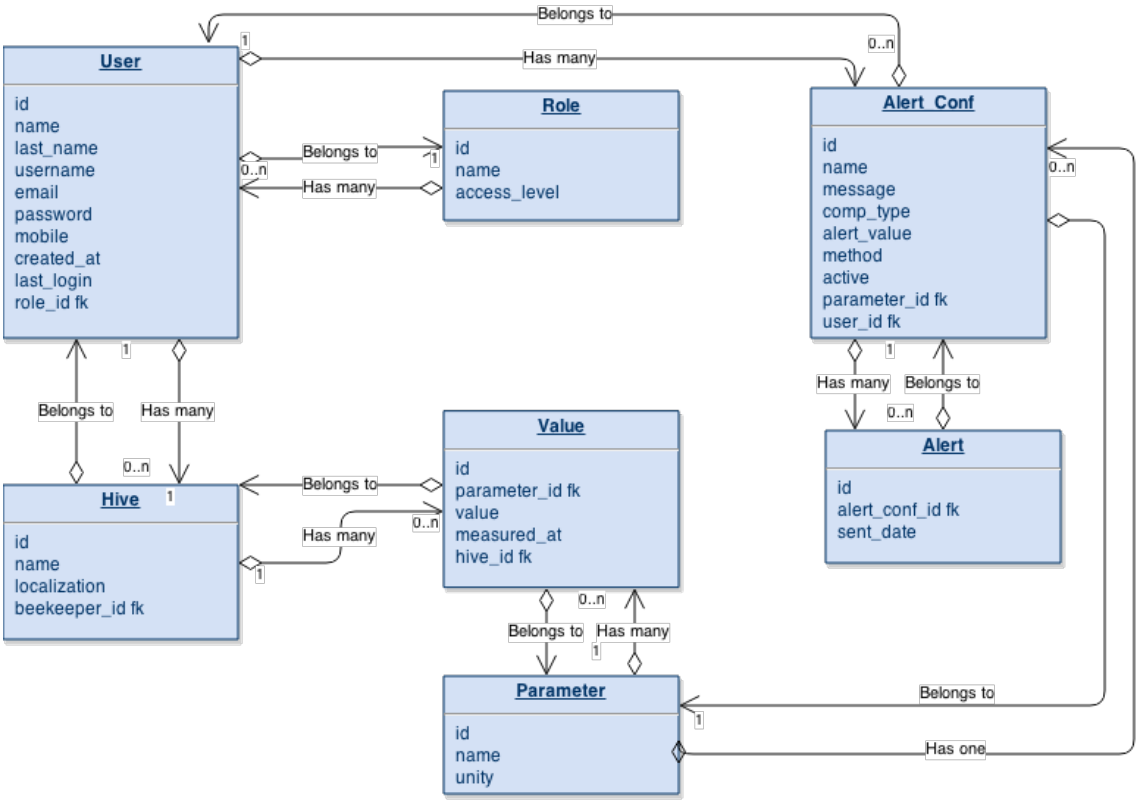

FIGURE 5.5 – Modèle de données pour le système BMONS

5.2.2 Carte Arduino et transmission des données

Au niveau de la carte Arduino, nous avons commencé à prendre en main les capteurs dès que nous les avions reçus notamment les capteurs de température et de pression. Nous avons commencé à coder un programme qui permettrai de récupérer la température en degrés. Pour la pression, nous récupérons une certaine valeur de résistance. A partir de cela, nous comptons effectuer un étalonnage afin d'en tirer une information sur la masse de la ruche ou des hausses selon la saison et l'envie de l'apiculteur. Cette étalonnage se fera en fonction des informations présentes sur la documentation constructeur du capteur de pression.

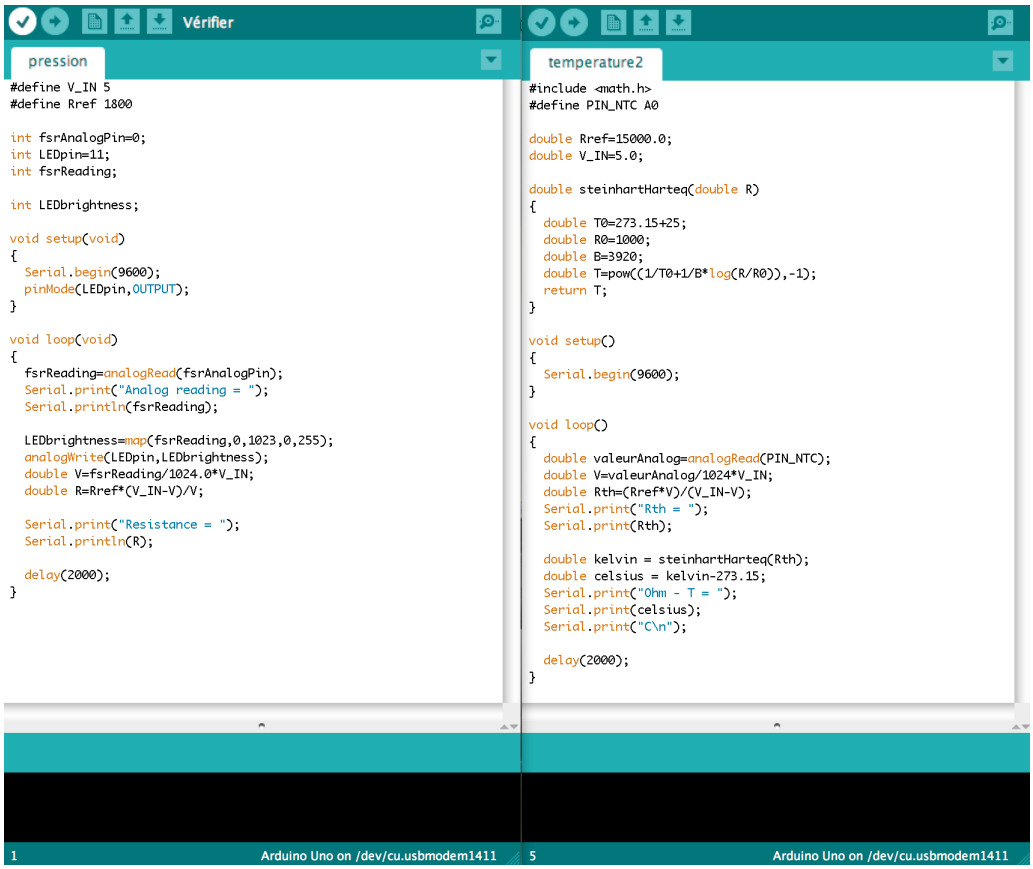

FIGURE 5.6 – Programmation de la carte Arduino

Troisième partie

Organisation

5.2.3 Identification et affectation des tâches

L'analyse effectuée grâce aux diagrammes WBS et GANTT met en avant deux directions principales :

— création d'un cadre de mesure comportant l'électronique embarqué
— traitement des différents flux d'informations pour afficher les données des ruches sur un site web

Nous n'avons pas souhaité scinder totalement le groupe selon ces deux directions pour ne pas perdre totalement contact avec une partie du projet. Certains se sont, cependant, spécialisés dans un domaine. Etiene s'est intéressée à la base de donnée et au site web. Nicolas s'est spécialisé dans les fonctionnalités de la carte Arduino. Benoit s'est spécialisé dans la confection du cadre de mesure. Les autres membres ont gardés des fonctions transverses, pouvant ainsi s'adapter à la situation et travailler ce qui était nécessaire.

5.2.4 Méthodes de travail

A chaque début de séance, nous nous retrouvons pour une petite réunion (10 à 15 minutes). Celle-ci a pour but d'informer tous les membres du groupe de l'avancée du travail et des problèmes rencontrés par chacun. Nous déterminons également les éléments qui doivent être effectués lors de cette séance et les membres qui en sont chargés. Enfin c'est une occasion de maintenir la cohésion dans le groupe.

Quatrième partie

Présentation des réalisations

Chapitre 6

Le cadre de mesure

Le cadre de mesure a été réalisé en bois. Les dimensions ont été choisies afin de limiter au maximum l'encombrement et en concertation avec notre client, M. SINGHOFF, afin que l'environnement des abeilles ne soit pas perturbé. Sa hauteur maximale a été fixée à 1 cm. Le cadre a été fabriqué grâce à la fraiseuse de l'ENSTA Bretagne en deux parties qui peuvent s'emboîter. Nous avons ensuite intégré tous les capteurs de mesure au cadre, comme nous le détaillerons dans la partie suivante. La structure à enfin été refermée par de fine plaques de métal car leur rigidité permet aux capteurs de pression d'effectuer des mesures correctes. Ceci garanti également l'étanchéité de la structure.

6.1 Intégration des capteurs

Avant d'être intégrés au cadre, les capteurs ont été testé individuellement. Les capteurs de pression qui permettront de déterminer le poids de la ruche ont notamment dû être étalonnés. Nous avons réalisé cette manipulation dans un des laboratoires de mécanique de l'école. Comme nous pouvons le voir sur la figure 6.1, nous avons placé chaque capteur de pression sur une machine capable d'exercer une gamme prédéfinie de pressions sur la zone de test. Nous avons relevé la réponse du capteur, une tension entre 0 et 5 volts, pour chaque échantillon lors de trois séries de mesures. Ces séries étaient composées d'un ensemble d'échantillons répartis comme suit : 20 valeurs croissantes de force de 0 à 100 Newtons puis 20 valeurs décroissantes de 100 à 0 Newton.

L'ensemble des valeurs obtenues nous a permis de tracer la courbe qui nous donne une mesure de masse en fonction de la tension en sortie du capteur, voir figure 6.2. Nous avons ensuite obtenu une équation de cette courbe de tendance. L'équation de cette courbe est de type polynomiale de degré 6 et $R = 0.96$. Comme les calculs de conversion ne seront pas effectués par l'Arduino mais par le serveur, la forme de cette équations n'est pas problématique. Nous avons également vérifié que tous les capteurs de pression suivent bien la même loi. Nous avons donc une formule unique de conversion de la tension de sortie en masse appliquée.

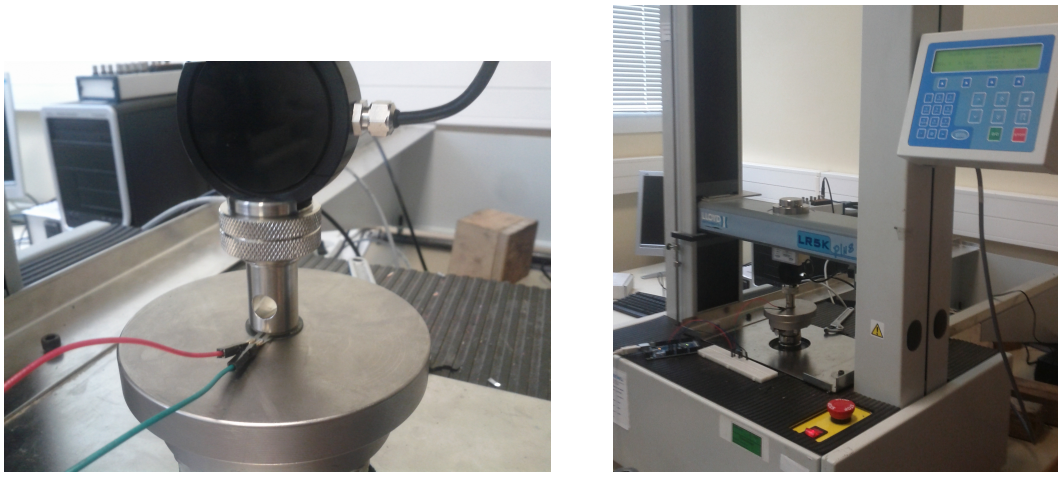

FIGURE 6.1 – Étalonnage des capteurs de pression dans un laboratoire de mécanique de l'ENSTA Bretagne

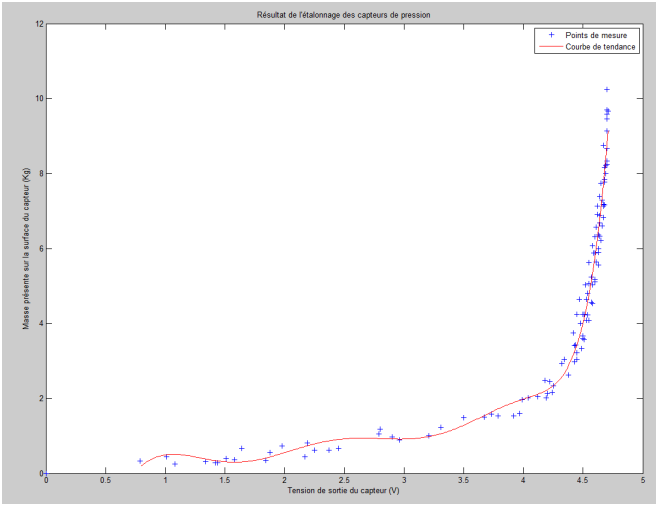

FIGURE 6.2 – Courbe de tendance obtenue par l"étalonnage des capteurs de pression

6.2 Gestion des données par Arduino

Chapitre 7

Server

7.1 Virtual Machine

During the development of the BMONS systems, we opted for the utilization of a Virtual Machine. A virtual machine is a way to emulate a particular computer system from another system. Considering that most of the development would take place during class and inside the school's labs, we are faced with one issue : not having full administrator powers and full control of the software we are allowed to install. Having a complete virtual machine also allows a full portability of our development environment since it's common that the actual physical machines used for development vary between classes. Other advantage include reducing the stress of deployment and developing in a safe environment. Once the project is done, it is possible to just copy the virtual machine to its final destination instead of doing a full reinstall of the system. Should any problems occur during the development, like corruption of the essential packages, for example, it will not affect the main system from which the virtual machine loaded. Multiple virtual machines can co-exist in a hard drive, each one having its own full environment, using different softwares installed and eliminating possible conflits. Even though a virtual is somewhat "closed", it is possible to share files between the main system (host) and a virtual machine (guest) using specific partitions.

By using the software called Oracle VM VirtualBox, we were able to create a full development environment in a virtual machine with the Ubuntu operational system version 14.04. Ubuntu was our choice of operational system because it combines the power and possibilities of a Unix system, useful packages installed out of the box and a friendly Graphical User Interface. All that while still being completely free to use, since Ubuntu is an open source software.

Once the operational system is installed on the virtual machine, it might be necessary to configure a network proxy so it is possible to access the internet from within. That is the case when using computers from the school's labs. Using Ubuntu, however, this is not such a difficult task. A proxy can be added by accessing System's Settings > Network > Proxy.

7.2 Basic packages and installation

To install BMONS on the virtual machine, it is firstly necessary either to download BMONS from the github repository (https ://github.com/Etiene/sailor/archive/master.zip) or to download at least a file called install.sh, located at the directory /scripts/-server

Even though a virtual machine allows a portable environment, eliminating the need to install all software needed everytime a developer changes computers, sometimes errors might occur and a fresh install of the system might be necessary. This is why BMONS is provided with a install script for the server side. A bash script, extension .sh, runs multiple commands that normally would be necessary to be ran in the command line terminal and is very useful for automating tasks or a repeated sequence of commands. The install.sh file provided with BMONS will, then, automatically install most of the required software to run the BMONS server and modify it. Please note that this script should be run by the root user or using the privileges of the root user by using sudo.

cd <location of BMONS at the filesystem>/scripts/server sudo ./install.sh

Among the essential packages installed by this script you will find : Git, the version control software used to keep track of BMONS ; Ruby, the server-side programming language used on BMONS's website ; Rails - the framework over which BMONS's website is developed ; PostgreSQL - the database used to store BMONS's data ; and pgAdmin III - a GUI (Graphical User Interface) software to administrate PostgreSQL.

An IDE (Integrated Development Environment) is not provided with the install script. Not only some IDEs are needlessly heavy, we believe that each developer has the right to choose their preferred way to write their code. The choice of our team was to use regular text editors, like gedit or Sublime Text (recommended).

7.3 Webserver

BMONS's website's server files are located under the directory bmons-site of our files.

Before the first use of BMONS, it is necessary to navigate to this directory and run 'bundle install' and 'rake db :schema :load' at the command line. Please note that this step is not necessary if you are running BMONS from a previously configured virtual machine. This is required only if it is a fresh install of BMONS and only before the first time of running the webserver.

cd <location of BMONS at the filesystem>/bmons-site bundle install rake db :schema :load

BMON's website uses some third-party softwares that made the development phase more productive, such as Devise, for example. Devise is a full-featured authentication solution which handles all of the controller logic and form views. This was used to implement the login system for BMONS. Running 'bundle install' will install all the necessary gems (plugins for the Ruby language) required to run Rails and the other gems that we chose to use during development.

When installing the server for the first time, the database will be empty. It will not contain the tables that store our data. This is why the command 'rake db :schema :load' is necessary. It will load the schema of the entities used by the website and generate the queries that will create those tables.

To start the server, it is necessary to run the command 'rails server'.

cd <location of BMONS at the filesystem>/bmons-site rails server

Once that is done, the website will be accessible via the url configured.

7.4 Database and Backup

The choice of database our team made for BMONS was PostgreSQL. Many options were possible but there are some advantages to using this specific one. While NoSQL databases are interesting and can be very fast in terms of performance, they are not very easy to deal in terms of development. Also, NoSQL is usually very fast for accessing and reading data but not so fast for writing and inserting data. Normally, BMONS should insert data way more frequently than reading. That is because one single user that is not connected to the website the whole time can have multiple beehives that will be sending data to the server on a regular basis, making us to reject this option right away. In addition to this, the archirecture of BMONS's data is already compatible with a regular SQL relational data.

There are many possible options of SQL databases, however, among those, PostgreSQL was favored because our team already had some knowledge of it and because it is a very good, reliable and stable open source software that is capable of dealing with very high volumes of data if necessary.

It is recommended to do database backups regularly. In case data is corrupted, it will be possible to restore the original data from recent a backup. BMONS is provided with a script that will automatically dump the database data into a dated file in the filesystem regularly (every 3 hours). It is recommended that that these generated files are copied to a safe location, such as another computer or an external hard drive from time to time.

Cron is an automatic scheduler for Unix systems that will run command line commands in specific times or time intervals according to the following pattern : comment * * * * * command to execute day of week (0 - 6) month (1 - 12) day of month (1 - 31) hour (0 - 23) min (0 - 59)

To activate the automatic backups, it is necessary to navigate to the directory /scripts/server of BMONS files. Under this directory is located a file called cron.txt. When making a fresh install of BMONS, it might be necessary to edit the line 6 of this file and correct the path to the backup bash script. It is also possible to change the frequency of the backups if desired.

0 */3 * * * /home/bmons/BMONS/scripts/server/db$_b$*ackup.sh*

Then you can run the following command to incorporate this cron job into the schedule :

crontab cron.txt

7.5 The web interface

The web interface of BMONS was built using Ruby on Rails. Rails is an open source web application development framework written in the Ruby language. It's used specially for developing database-driven websites and allows the creation of applications using pre defined structures. Rails is a Model-View-Controller (MVC) framework. MVC is a software design pattern for implementing user interfaces. It divides the software into three interconnected parts. Models : They store the data of the application, its business rules and methods. It's often regarded as a "mirror" of the database. Views : They're the output of a model data representation, such as a table or a diagram, for the user. Controllers : They make the mediation between the models and the views, sending commands to update models' states and sending commands to the view to alter the presentation of the models.

The Rails philosophy includes two major guiding principles :

"Don't Repeat Yourself : DRY is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." By not writing the same information over and over again, our code is more maintainable, more extensible, and less buggy. Convention Over Configuration : Rails has opinions about the best way to do many things in a web application, and defaults to this set of conventions, rather than require that you specify every minutiae through endless configuration files." (http ://guides.rubyonrails.org/getting$_started.html$)

Rails philosophy and features allowed our team to develop an interesting application in a feasible time. This web application allows a beekeeper to register, login, change password, see a dashboard with visual data from their hives and manage hives and alerts. The application will also receive content from the Arduino located at the hive and update its data.

The authentication system was developed with the use of Devise, which is a plugin that fully handles the authentication through the MVC pattern. The pages to manage beehives, sensors, measurements, alerts and alert logs were firstly created with the use of scaffolding. A scaffold in Rails is a full set of model, database migration for that model, controller to manipulate it, views to view and manipulate the data, and a test suite for each of the above. This scaffold can be created through the command line in a very easy manner.

rails generate scaffold Beehive name :string user :references position :string mode :string

However, scaffolding alone is not enough and after the creation it is necessary to rearrange and modify the code to make it suit our needs. One example is modifying the beehive scaffold for showing only the beehives that belong to the user currently logged in.

One important aspect of the BMONS's website is the design. The design was developed with the help of Bootstrap. Bootstrap is a free and open source front-end framework that uses HTML, CSS, and JavaScript to provide re-usable components and other interface elements. It allows the development of responsive and mobile first projects on the web. With some customization Bootstrap, we were able to obtain pages that display our system's contents beautifully.

Bootstrap, however, is not enough for displaying our data. The dashboard requires the data to be easily grasped and even though tables are really useful, they are not sufficient to provide a quick visual analysis. Our team's choice of optimal display was a graphical one. The development of the graphical dashboard was made with NVD3, a JavaScript library that extends another library called D3 that allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. The advantage of using NVD3 over D3 is that it is an easier to use library with a reduced collection of components that satisfies our needs without taking away the power of D3. The data is then passed from the controller backend in ruby to the frontend view in a JavaScript piece of code that is then used and manipulated by the NVD3 library, producing beautiful results. The graphic allows for the visualization of measures from sensors in time of each individual beehive, allowing to switch between hives without refreshing the page.

7.6 The API and receipt of data from external source

BMONS contains an API for receiving the captured data by the Arduino and storing it on a database for later displaying to the beekeeper. By default a Ruby on Rails CRUD (create, read, update and delete actions for an entity) is already a REST (Representational State Transfer) API that will perform the necessary actions using the right HTTP protocol request method and with full JSON capability. A PATCH or PUT will access the update action, a GET will obtain the data in a JSON object as a response, a DELETE will access the delete action and destroy the desired data and, finally, a POST will receive a JSON object and store new data. For BMONS needs, it was necessary to use the POST request to receive new measurements periodically from an external source, the Arduino, instead of from a form in a view used by an authenticated user. To make sure the data coming in is still safe, for the measurement create action, instead of a user authentication, a Basic HTTP Authentication is used. That means that the POST request sent by the Arduino must also send an authentication token on its header and the data will only be stored if the details are correct. In addition to the authentication, the Arduino must also sent, of course, the data. The content-type of this request should be application/JSON and a JSON object containing the necessary information for creating a new measurement should be included in the body of the request. To test BMONS's RESTful API, a browser extension named Postman was used as a client.

Cinquième partie

Perspectives d'amélioration

Chapitre 8

Conclusion

Le projet BMONS a pour ambition de réaliser un système embarqué sur une ruche pour récolter les paramètres physiques de la ruche dans l'optique de détecter les anomalies de comportement des abeilles. Notre équipe s'est donc penché sur cette problématique avec des notions d'apiculture et de conduite de projet plutôt minces, voici un premier bilan.

Après cette première partie du projet nous pouvons enfin dire que nous comprenons les besoins et les attentes des apiculteurs, par exemple des fonctions auxquelles nous avions pensé, comme la détection du taux d'humidité dans la ruche ne semblent finalement pas primordiale au projet. En revanche des options comme le comptage des abeilles seraient appréciées par les apiculteurs, mais des compromis doivent êtres faits car certaines de ces options entrainent un surcoût trop conséquent pour que le projet satisfasse les exigences initiales.

Nous devons une grande partie de ces connaissances en apiculture aux réponses précises de monsieur Franck Singhoff, qui nous a également prêté une ruche pour que nous puissions travailler sur l'implémentation des capteurs sur notre prototype.

Les difficultés que nous avons rencontrées, par exemple lors de la compréhension et de la création des diagrammes nécessaires à l'établissement de la spécification fonctionnelle sur les trois axes, ont été surmontées grâce à l'aide de nos encadrants de projet, mais aussi grâce au dialogue entre les membres du groupe. Les choix des divers composants qui constitueront le prototype ont aussi été une source de problèmes, car même si nous avions une idée de quel type de composant nous avions besoin, faire le tri entre tous ceux qui existent nécessite de faire des choix entre par exemple le prix et la précision d'un capteur, et cela influe sur les exigences du projet.

Ces quelques mois de travail en commun nous ont permis d'avoir un aperçu d'ensemble de la conduite d'un projet. Nous nous sommes rendu compte de la quantité de travail que représente la partie ingénierie système sur un projet comme celui-ci. Et bien que nous ne puissions pas prévoir tous les aléas de la seconde partie du projet, ce travail d'anticipation nous permet d'être plus sereins face au

travail qu'il reste à fournir.

Sixième partie

Annexes