GEII
Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

Identification Par NFC

RAPPORT DE PROJET TUTORÉ

Licence Professionnelle métier de l'électronique : parcours VEGA

Miller Damien & Bedu Alexandre

Tuteur : LOMBARD Christophe

Sommaire

Table des matières

Remerciements

Nous remercions tout le corps enseignant du département GEII de l'Institut universitaire de technologies de BELFORT, pour la qualité de son enseignement, pour sa disponibilité tout au long du projet mais également pour les moyens mis à notre disposition pour mener à bien notre projet.

Nous remercions plus particulièrement Mr Christophe Lombard pour nous avoir apporté ses connaissances et son expérience dans le domaine de l'informatique.

Présentation du projet

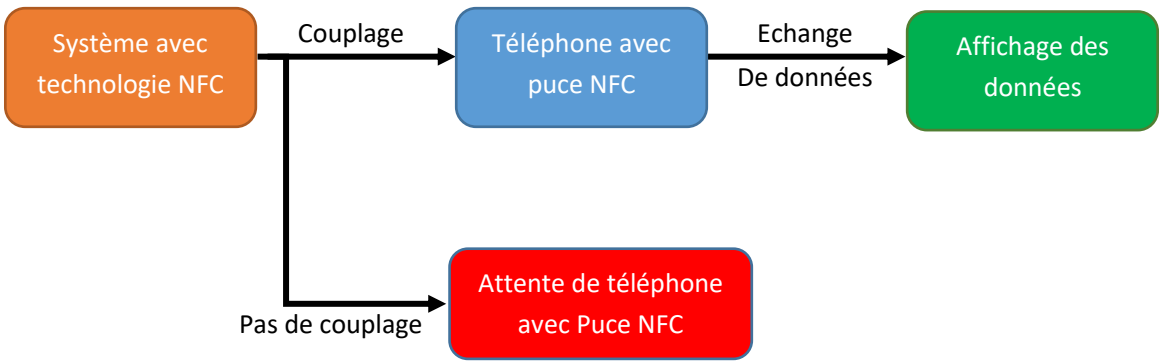
Description du besoin :

On souhaite concevoir un présentoir permettant d'identifier un véhicule par le biais d'une technologie NFC. Lorsque l'on présente un téléphone utilisant une application NFC, le pupitre envoie les informations commerciales ou techniques concernant le véhicule auquel il est rattaché.

On doit pouvoir y retrouver toutes les informations concernant la motorisation, la consommation, la finition, les dimensions du véhicule ...

Le pupitre est pensé dans un but commercial afin d'informer le client des caractéristiques du véhicule (Dans la partie vente d'une concession par exemple).

Analyse du fonctionnement :



Présentation du NFC

Description du NFC :

Le NFC est technologie de communication de proximité (quelques centimètres) lancée par Sony et Philips, le NFC (pour Near Field Communication ou communications en champ proche), permet d'échanger des données entre un lecteur et n'importe quel terminal mobile ou entre les terminaux eux-mêmes et ce, à un débit maximum de 420 Kbits/s.

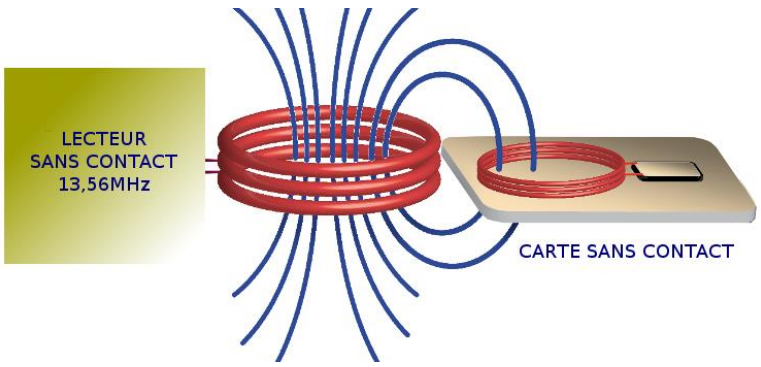
Fonctionnement du NFC

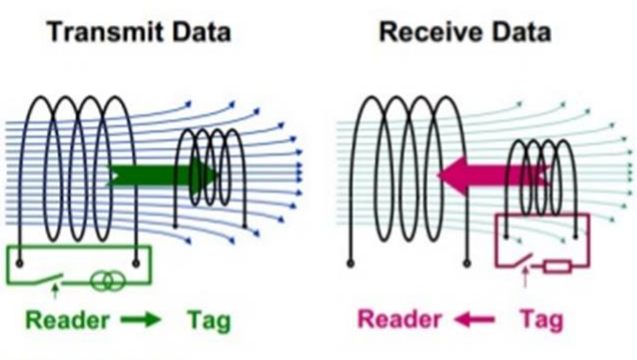
La NFC est une communication basée sur le couplage magnétique de deux bobinages placés à proximité l'un de l'autre : celui du lecteur et celui du récepteur La NFC est un protocole de communication basé sur une fréquence de 13.56MHz et l'interface de visualisation est du UART (Universal Asynchronous Receiver Transmitter).

Pour le synoptique de fonctionnement voir annexe.

Il existe deux modes de communication différents, le mode passif où l'un des deux éléments n'est pas alimenté sur lequel on peut transférer des données ou bien lire celles qui sont déjà écrite dessus.

Le deuxième mode est le mode actif dans lequel les deux parties sont alimentés et peuvent transférer des données, ils jouent tours à tours le rôle de lecteur et de récepteur. Ce mode est également appeler Peer to Peer.



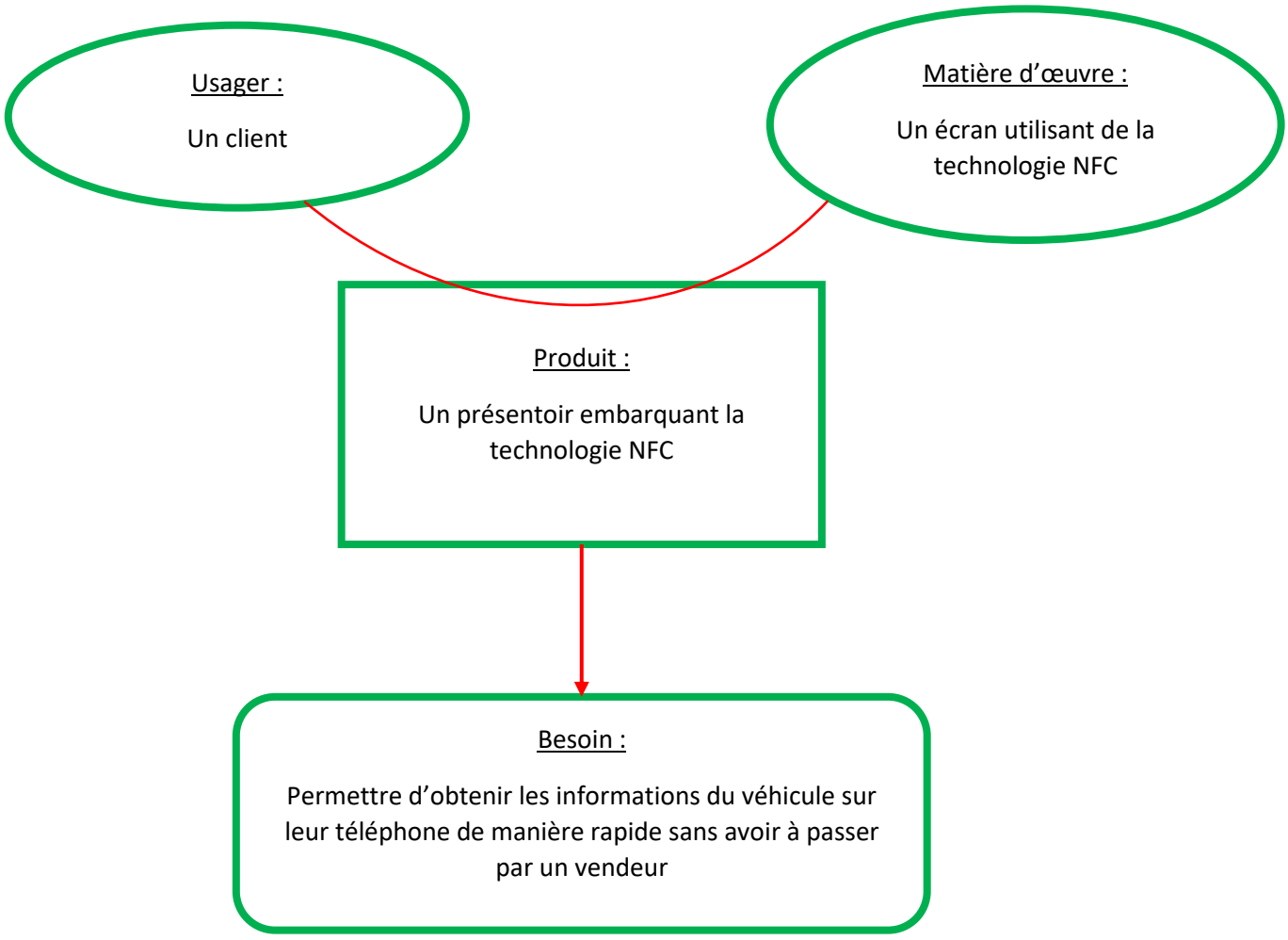


Environnement du produit

Pour un fonctionnement optimal, notre système doit répondre à plusieurs caractéristiques. Pour regrouper un maximum de fonction nécessaire au bon fonctionnement nous avons réalisé plusieurs synoptique.

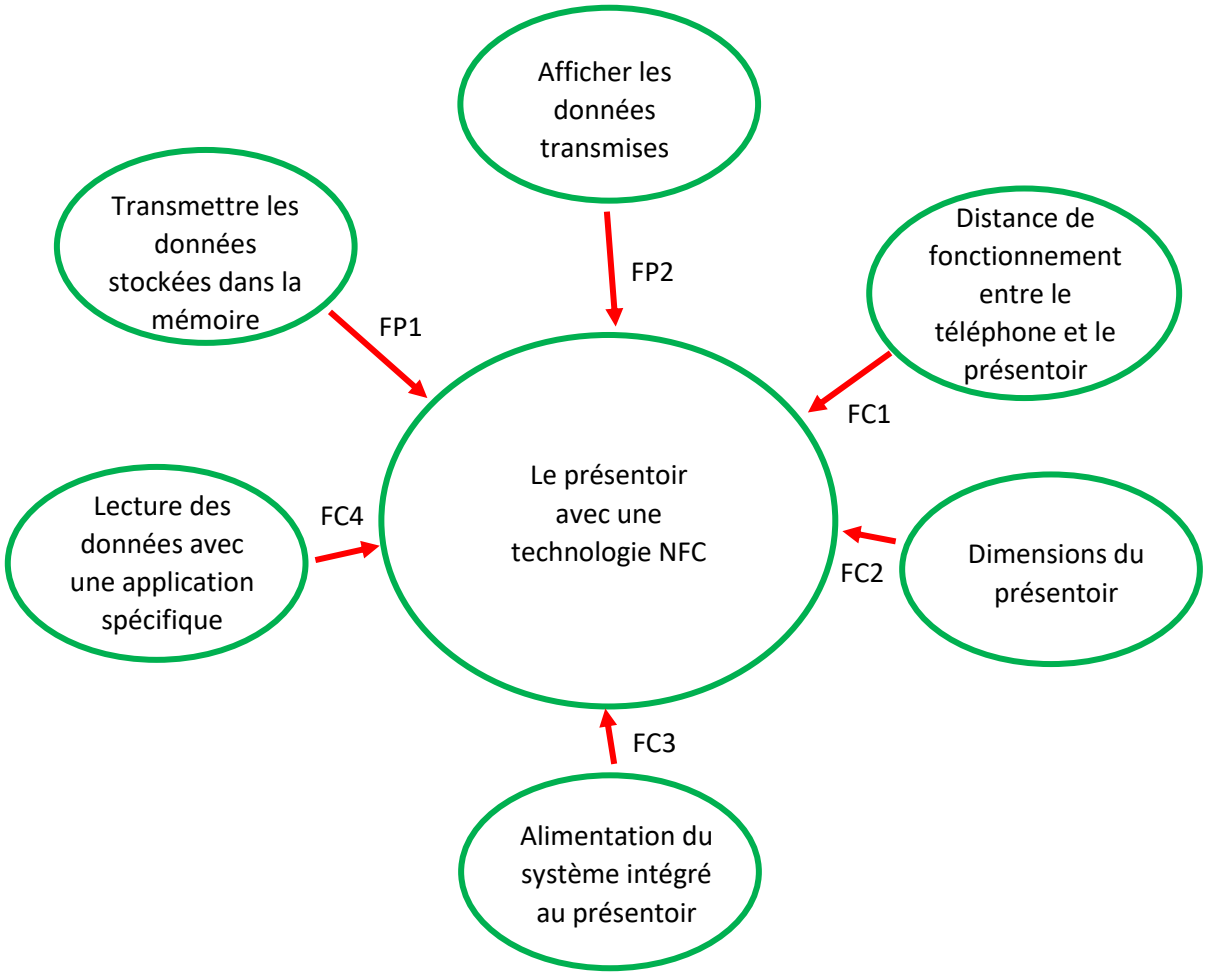
Bête à corne

On utilise une bête à corne pour visualiser le produit en fonction du besoin et des différents éléments avec lesquelles le système va interagir, ici l'usager et la matière d'œuvre.



Analyse fonctionnelle

On utilise une analyse fonctionnelle pour énumérer les différentes fonctions principales et de contrainte auxquelles notre système devra répondre.



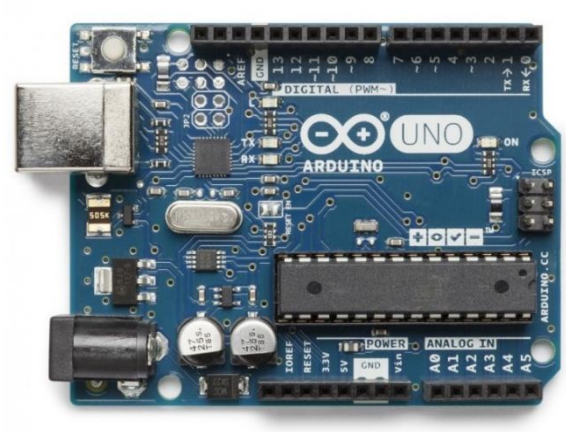
Fonction	Énoncé de la fonction	Exigence
FP1	Transmettre les données	NFC
FP2	Afficher les données	Téléphone
FC1	Distance de fonctionnement	2 à 5 cm
FC2	Dimensions du présentoir (cm)	H:140 L:30 l:21
FC3	Alimentation du présentoir	3.5V ou 5V
FC4	Lecture des données	Application Android

Choix des composants

Durant notre projet nous avons été amener à utiliser plusieurs composants utilisant la technologie NFC et le logiciel Arduino pour écrire le code.

Carte Arduino Uno

Nous avons choisi d'utiliser la carte Arduino Uno qui nous sert de microcontrôleur sur lequel nous connectons notre antenne NFC.



- Microcontrôleur : ATmega 328
- Fréquence d'horloge : 16 Mhz
- Tension de fonctionnement : 5 Volts
- Interface : USB (alimentation et transfert de programme vers la carte)
- Dimensions : 6.86 cm x 5.3 cm

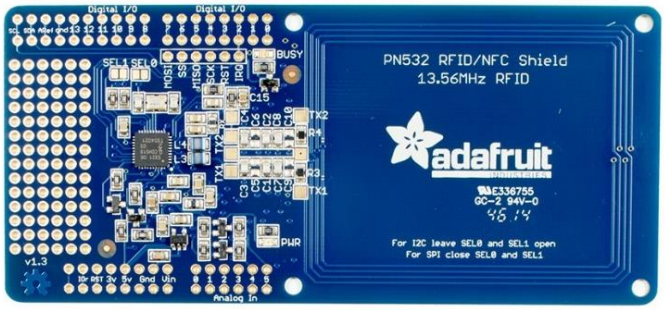
Carte DFRobot 0231

Nous avons tout d'abords utiliser une carte DFRobot 0231 car c'est celle qui nous a été fournie par notre professeur référent.

- Microcontrôleur : PN 532
- Tension d'alimentation : 3.3 ou 5 Volts
- Band rate : 115 200 Bps
- Interface hôte : UART
- Fréquence de communication : 13.56 MHz
- Distance de fonctionnement : 20 à 50 mm
- Dimensions : 11 cm x 5 cm

Carte Adafruit PN 532 Shield

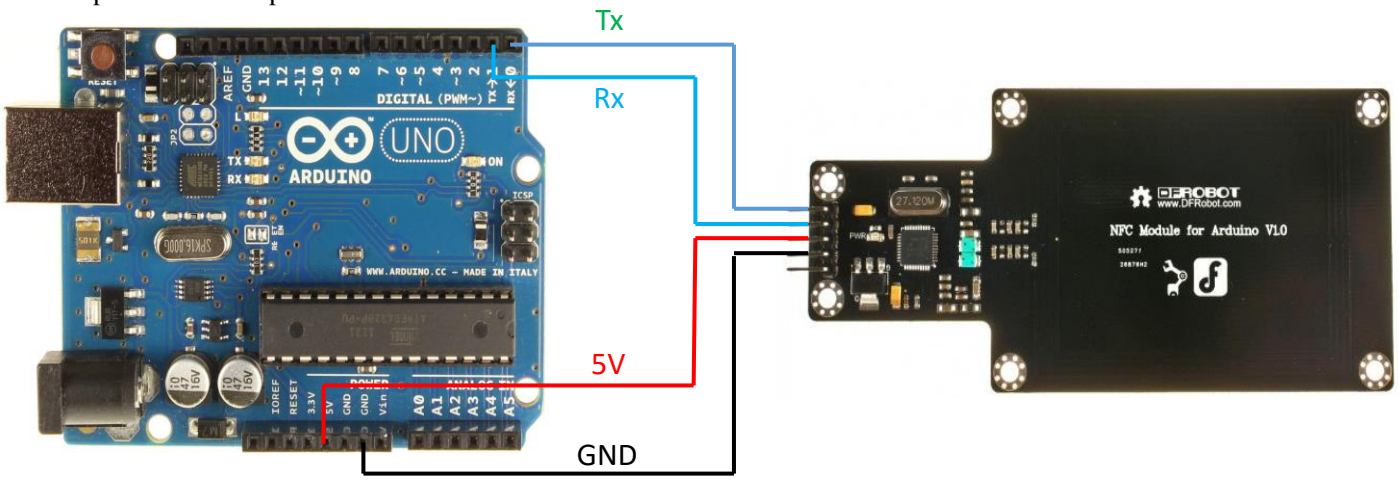
Nous avons utilisé, dans un second temps, le Shield Adafruit car la précédente carte nous restreignait dans l'utilisation de bibliothèque Arduino, en effet nous étions obligés d'utiliser celles de DFRobot nous bloquant dans notre avancement.



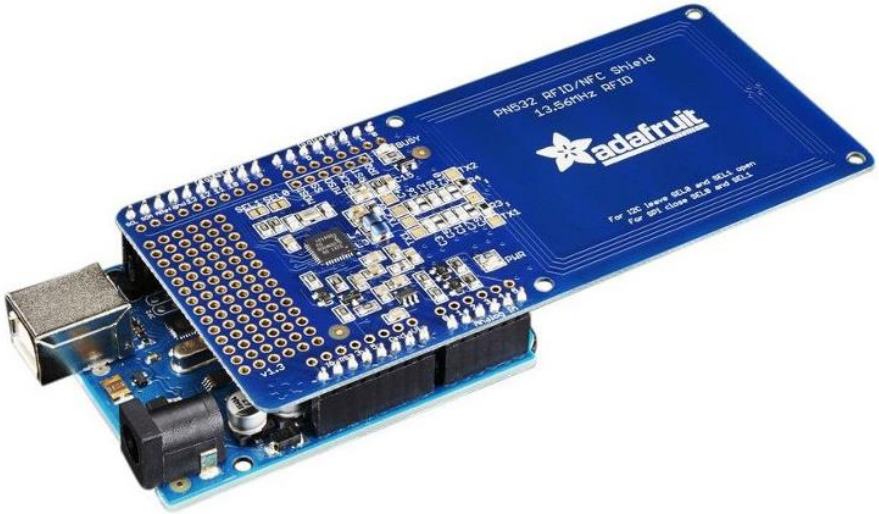
- Microcontrôleur : PN 532
- Tension d'alimentation : 3.3 ou 5 Volts
- Band rate : 115 200 Bps
- Interface hôte : UART
- Fréquence de communication : 13.56 MHz
- Distance de fonctionnement : 20 à 100 mm
- Dimensions : 12 cm x 6 cm

Branchement

Pour pouvoir utiliser la carte DFRobot on reprend l'alimentation en 5 Volts et la masse sur la Arduino Uno puis on relie le port série des deux carte en mettant le Tx de l'une sur le Rx de l'autre et inversement.



Pour la carte Adafruit c'est beaucoup plus simple car celle-ci est un shield.



Présentation Technique

Lecture et écriture de tag avec DFRobot

Notre premier objectif était la lecture d'un tag à l'aide de l'antenne de DFRobot. Pour ce faire nous avons utilisés arduino comme logiciel d'exploitation. Nous avons créé un code qui nous permet de lire le contenu d'un tag et ainsi repérer à partir de quelle adresse se trouve le mot inscrit à l'intérieur. Pour visualiser ces données, nous avons utilisés le moniteur série présent dans le logiciel arduino.

```
  nfc.begin(Serial);
data[i] = nfc.read(i);
```

La première ligne de code nous permet d'initier la communication en NFC. La seconde ligne quant à elle permet de lire le contenu du tag et le placer dans un tableau.

Une fois que nous étions en mesure de faire ceci, nous nous sommes concentrés sur le fait de pouvoir écrire un texte dans un tag. Pour ce faire nous avons dû créer un code.

```
    uint8_t mot[]="Licence vega";

nfc.begin(Serial);

success = nfc.writeBytes(mot,43,20);
```

Ce morceau de code nous permet quant à lui d'écrire dans le tag, pour cela on fait un tableau contenant ce que l'on veut écrire en format 8 octets signés. Nous utilisons par la suite la fonction d'initialisation de la communication. La dernière est utilisée pour écrire le mot, à l'adresse 43, long de 20 caractères maximum.

```
§□§§@□a□§§§§§§§9§§§§□§§@□0□§41
§□§§@□a□§§§§§§§9§§§§□§§@□0□§42
§□§§@□a□§§§§§§§9§§§§□§§@□0□§43    l
§□§§@□a□§§§§§§§9§§§§□§§@□0□§44    i
§□§§@□a□§§§§§§§9§§§§□§§@□0□§45    c
§□§§@□a□§§§§§§§9§§§§□§§@□0□§46    e
§□§§@□a□§§§§§§§9§§§§□§§@□0□§47    n
§□§§@□a□§§§§§§§9§§§§□§§@□0□§48    c
§□§§@□a□§§§§§§§9§§§§□§§@□0□§49    e
§□§§@□a□§§§§§§§9§§§§□§§@□0□§50
§□§§@□a□§§§§§§§9§§§§□§§@□0□§51    v
§□§§@□a□§§§§§§§9§§§§□§§@□0□§52    e
§□§§@□a□§§§§§§§9§§§§□§§@□0□§53    g
§□§§@□a□§§§§§§§9§§§§□§§@□0□§54    a
§□§§@□a□§§§§§§§9§§§§□§§@□0□§55
```

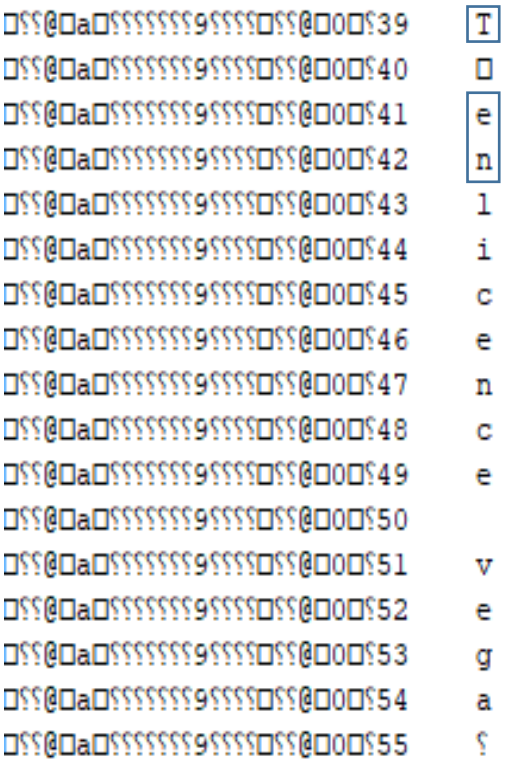
On obtient alors ceci, on peut donc voir que chaque caractère à son adresse spécifique.

On commence à l'adresse 43 car c'est la première adresse dans laquelle nous pouvons écrire et que l'on peut lire avec notre programme de lecture. Les adresses précédentes servent au paramétrage et à l'identification du tag.

Code en annexe 1.

Problème rencontrées avec la carte DFRobot

L'un des problèmes les plus important c'est que nous devions utiliser le programme de lecture que nous avons créé. En effet, pour pouvoir lire le tag sur un téléphone il faut un certain type d'encodage que nous n'arrivions pas à reproduire.



Ici nous pouvons voir le type d'encodage présent sur le tag que nous avons formater avec une application android (NFC tools).

Le T signifie que le contenu du tag est de format texte.

Le en signifie que le texte est en langage anglais.

C'est ce fameux type d'encodage que nous n'arrivons pas à mettre en œuvre.

Nous avons par la suite essayé d'utiliser d'autres bibliothèques nous permettant d'écrire sous ce format mais le problème c'est que la carte n'accepte que la bibliothèque DFRNFC qui est la bibliothèque officielle de DFRobot. Il nous est donc impossible d'utiliser d'autre bibliothèques.

Pour pallier à ce problème, nous avons choisi de nous tourner vers une nouvelle carte nous permettant d'utiliser n'importe qu'elle bibliothèque présentes sur GitHub. Notre choix c'est tourné sur la carte Adafruit NFC PN532 shield.

Lecture et écriture avec la carte adafruit

Grâce à cette carte, nous sommes capables d'écrire sous le bon format à l'intérieur d'un tag. L'encodage se fait grâce à la bibliothèque NDEF. Nous pouvons également lire le contenu de ce tag avec un téléphone qui est l'un de nos objectifs.

```
#include <Wire.h>
#include <PN532_I2C.h>
#include <PN532.h>
#include <NfcAdapter.h>

PN532_I2C pn532_i2c(Wire);
NfcAdapter nfc = NfcAdapter(pn532_i2c);
```

La particularité de cette carte c'est qu'elle peut fonctionner sous de type de format. Le premier est l'I2C et le second le SPI. De base, la carte est en I2C. Pour la passer en mode SPI, il faut souder deux cavalier sur les SEL0 et SEL1.

Pour une communication passive, l'I2C est le moyen le plus simple. En effet, il suffit d'avoir la bibliothèque NfcAdapter pour pouvoir écrire ou lire le tag.

Dans l'exemple du dessus, nous pouvons voir que nous déclarons la communication du NFC en I2C.

```
if (nfc.tagPresent()) {
    NdefMessage message = NdefMessage();
    message.addUriRecord("Licence vega");

    bool success = nfc.write(message);
    if (success) {
      Serial.println("Success. Try reading this tag with your phone.");
    } else {
      Serial.println("Write failed.");
    }
}
```

Dans ce code, nous utilisons une condition if pour déterminer si un tag est présent. Si la condition est vrai, nous écrivons alors le message dans le tag et affichons un message sur le moniteur série disant que l'écriture c'est fait avec succès. Dans le cas contraire rien ne se passe et nous attendons qu'un tag soit présent.

La fonction message.addUriRecord permet d'écrire dans la variable message le texte qui suit cette fonction. Nous pouvons soit écrire une phrase, soit y insérer un URL.

```
void loop(void) {
    Serial.println("\nScan a NFC tag\n");
    if (nfc.tagPresent())
    {
        NfcTag tag = nfc.read();
        tag.print();
    }
```

La fonction de ce code est la même que l'autre. La seule différence c'est que nous utilisons nfc.read pour lire le contenu du tag et ensuite l'afficher sur le moniteur série.

Code en annexe 2.

Problèmes rencontrés avec la carte adafruit

Notre objectif final était de pouvoir lire une base de données directement avec notre téléphone sans passer par un tag physique. Pour ce faire nous devons utiliser la communication active appeler également la communication Peer to Peer. Mais nous avons rencontrés des problèmes car nous n'arrivons pas à faire communiquer c'est deux éléments. Nous pensons que le problème viens soit du type d'encodage qui n'est pas le même pour un téléphone que pour un tag soit à cause de la version des logiciels entre le téléphone et l'adafruit. Une des autres possibilités c'est qu'il se peut que l'adafruit et le téléphone soit tous les deux dans le même mode de fonctionnement c'est-à-dire tous les deux en mode émission, l'un bloquant l'autre.

```
#include "SPI.h"
#include "PN532_SPI.h"
#include "snep.h"
#include "NdefMessage.h"

PN532_SPI pn532spi(SPI, 10);
SNEP nfc(pn532spi);
uint8_t ndefBuf[128];
```

Pour faire fonctionner le code en Peer to Peer nous devons passer la carte adafruit en mode SPI. Pour que cela fonctionne nous devons inclure la bibliothèque SNEP qui inclus les éléments de base pour cette communication. D'autre part, nous devons également déclarer la communication en mode SPI ainsi que la patte correspondante sur la carte arduino qui est ici la patte 10.

Nous avons également déclaré un tableau signé sur 8 octets avec une place de 128 octets.

```
NFC Peer to Peer Example - Send Message
Send a message to Peer
Failed
```

Voici ce que donne le résultat quand nous voulons envoyer le message sur le téléphone. On s'aperçois dans le code qu'il s'agit d'un problème d'encodage mais nous ne savons pas d'où il vient réellement.

Après plusieurs recherches, nous avons effectués des transformations sur la carte Arduino en intervertissant les pins 2-3-4-5 avec les 10-11-12-13. Plusieurs personnes avaient ce même problème et l'on résolu avec ce procédé. Malheureusement pour nous, cela n'a pas fonctionné.

La communication avec de l'Android est très complexe du fait qu'il faut énormément de bibliothèque et de savoir en programmation Android car il y a beaucoup de pare feu à outrepasser.

Code en annexe 3.

Fabrication de notre carte arduino

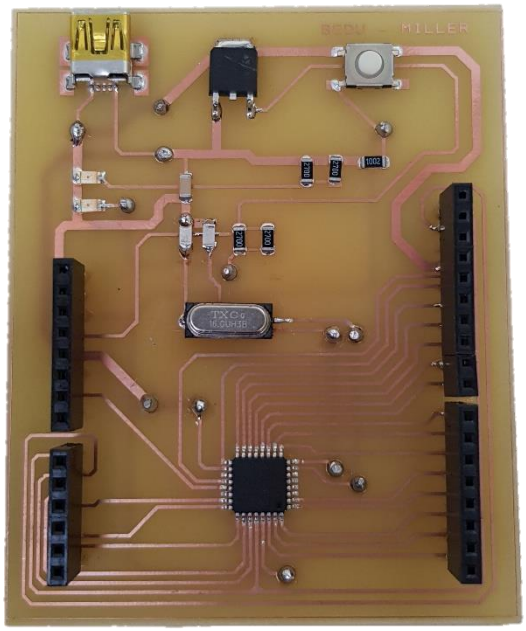
Pour que notre système soit indépendant, nous avons crées une carte dans laquelle nous pouvons envoyer notre programme d'écriture. Pour ce faire, nous nous sommes inspirés du fonctionnement de la carte arduino UNO. Nous avons inséré un ATmega328 qui est un microcontrôleur présent sur la carte UNO. Nous avons mis également un quartz de façon à pouvoir transférer des données à l'ATmega328 avec une cadence d'horloge de 16MHz. Le port mini USB permet d'amener l'alimentation sur la carte en 5V. Comme certain composant fonctionne en 3.3V, nous avons ajouté un régulateur LM323. Nous avons placé un bouton de sorte à pouvoir faire un reset de la carte si nécessaire.

Nous avons placé des connecteurs sur la carte pour pouvoir intégrer la carte Adafruit sur celle-ci.

Pour fabriquer cette carte nous avons utilisé le logiciel DesignSpark qui permet de réaliser des circuits imprimés. Nous avons choisi d'utiliser des composant CMS (Content Management System) pour gagner un maximum de place sur la carte. Notre but final étant de l'intégrer sut un présentoir, celui-ci doit donc être le plus petit possible.

Une fois que nous avons réalisé se schématique, nous sommes passés sur la création du PCB (printed circuit board). Cela consiste à placer les différents composant sur une carte et effectuer le routage (câblage) entre ces différents éléments.

Le PCB et le schématique de la carte se trouve en annexe 4.



Ci-contre, notre carte imprimée et les composants soudés. Pour le soudage, nous avons utilisé une patte abrasive que nous avons déposé aux endroits souhaités, puis on place les composants dessus avant de placer le tout dans un four le tout en respectant le temps de chauffe. On part de 90°c puis on monte à 220°c et on redescend à 170°c pour effectuer un recuit. L'opération dure environ 5 minutes. Une fois que tous les composant sont soudés, nous effectuons un test de continuité pour s'assurer que tous est bien soudé et qu'il n'y ai pas de faux contact.

Problèmes rencontrés avec notre carte

Nous avons rencontré un problème majeur lors de la mise en service de notre carte. En effet lorsque nous avons voulu flasher notre microcontrôleur (ATmega328), la séquence de signature n'était pas la bonne. Après plusieurs tentatives et ayant changé la séquence de signature pour flasher un ATmega au format cms, nous avons pu en déduire un problème matériel.

Effectivement, après avoir regardé l'allure du signal émis par le quartz, nous nous sommes rendu compte que la fréquence d'oscillation n'était pas bonne car il n'y avait aucune fréquence logique.

Nous avons donc retiré le quartz présent sur notre carte et avons tenter d'utiliser le signal émis par le quartz de la carte arduino UNO. Mais un autre problème est survenu car la fréquence d'oscillation de ce quartz descendait à une fréquence de 2 à 4 MHz ce qui est insuffisant et par normal.

Nous avons également changé certains condensateurs pour en mettre des plus petits mais cela n'a rien changé.

Nous avons résolu le problème du flashage de notre carte. Le problème venait d'un micro faux contact présent entre les deux pistes du quartz. Une fois avoir gratté le faux contact nous avons pu lancer le programme du flashage. La carte peut maintenant recevoir notre programme et ainsi être parfaitement autonome.

Évolution possible du projet

Pour pouvoir faire une évolution, il faut réussi à faire communiquer la Adafruit avec un téléphone en fonction Peer to Peer dans le sens de l'émission d'une donnée arduino vers un système android. Une fois cela fait, l'objectif du projet sera atteint.

Lorsque tous ces problèmes seront résolus, il faudra créer une nouvelle carte dans laquelle sera inclus notre carte ainsi que la reproduction de la carte adafruit (antenne et pn532).

Il faudra par la suite inclure le système au sein d'un présentoir de préférence en plexi glace de sorte à ce que la communication ne soit pas perturbée ce qui arrive avec de l'aluminium.

Rendre le présentoir étanche, avec une source d'alimentation portable afin de le mettre à l'extérieur est un objectif final envisageable.

Conclusion

Ce projet nous à permit d'acquérir de nombreuse compétence dans le domaine de l'autonomie, du travail de groupe et la gestion de projet. Il nous a également permis de découvrir le domaine de l'informatique et plus précisément la communication en champ proche ce qui était nouveau pour nous. Le fait de créer une carte fut passionnant et des plus instructifs.

La technologie CMS nous a permis d'obtenir de nouvelle connaissance dans le domaine de l'électronique embarquée. Elle nous a appris à être rigoureux et précis.

Nous nous sommes réellement impliqués dans ce projet qui fût très passionnant. Nous recommandons vivement celui-ci aux futurs étudiants.

Bibliographie

- Carte arduino :
  https://download.tuxfamily.org/sourcetechno/Projet%20robot%201/Carte%20microcontroleur%20Arduino/Arduino%20uno.pdf

- Carte adafruit :
  https://www.adafruit.com/product/789

- Schéma carte arduino :
  https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf
- Fonctionnement NFC :
  http://pedagogie.ac-limoges.fr/sti2d/IMG/pdf/RFID_NFC_SIN_FAVARD.pdf

- Banque de donnée composant Designspark :
  https://componentsearchengine.com/index.html

- Bibliothèque GitHub pour arduino :
  http://don.github.io/slides/2014-04-29-oreilly-nfc/#/32
  https://github.com/don/NDEF/issues/24
  http://don.github.io/slides/2014-05-17-makerfaire-nfc/#/20
  https://github.com/adafruit/Adafruit-PN532
  https://github.com/don/NDEF
  https://github.com/DFRobot/DFRNFC

Annexe

Annexe 1 :

Code Lecture DFRobot :

```
#include "Arduino.h"
#include "DFRNFC.h"
//#include "test.txt"
DFRNFC nfc;

void setup(void)
{
  Serial.begin(115200);
  nfc.begin(Serial);
  Serial.println("recherche du pn532...");
}
void loop()
{
  int i;
  char data[752] = {0};
  int startAdress = 0;
  int dataLenght = 752;
  for ( i = startAdress; i < startAdress + dataLenght - 1; i++)
  {
    data[i] = nfc.read(i);
    Serial.print(i);
    Serial.print("    ");
    Serial.println(data[i]);
  }
}
```

Code Ecriture DFRobot :

```
#include "Arduino.h"
#include "DFRNFC.h"

uint8_t mot[]="Licence vega";
DFRNFC nfc;
void setup(void)
{
  Serial.begin(115200); //PN532 default SerialBaudRate is 115200

  //initialize nfc module
  nfc.begin(Serial);
  Serial.println("Looking for PN532...");
}

void loop()
{
  int address;
  char value;
  uint8_t success;
  uint8_t success2;
  int a = 0;
  for(int i=0;i<752;i++) //only 752 bytes in the data blocks
   {

    // write a byte to each address of the Mifare Classic Card
     success = nfc.writeBytes(mot,43,20);

   Serial.print(address);
  }
}
```

Annexe 2 :

Code lecture Adafruit :

```
#include <Wire.h>
#include <PN532_I2C.h>
#include <PN532.h>
#include <NfcAdapter.h>

PN532_I2C pn532_i2c(Wire);
NfcAdapter nfc = NfcAdapter(pn532_i2c);

void setup(void) {
    Serial.begin(115200);
    Serial.println("NDEF Reader");
    nfc.begin();
}

void loop(void) {
    Serial.println("\nScan a NFC tag\n");
    if (nfc.tagPresent())
    {
        NfcTag tag = nfc.read();
        tag.print();
    }
    delay(5000);
}
```

Code écriture Adafruit :

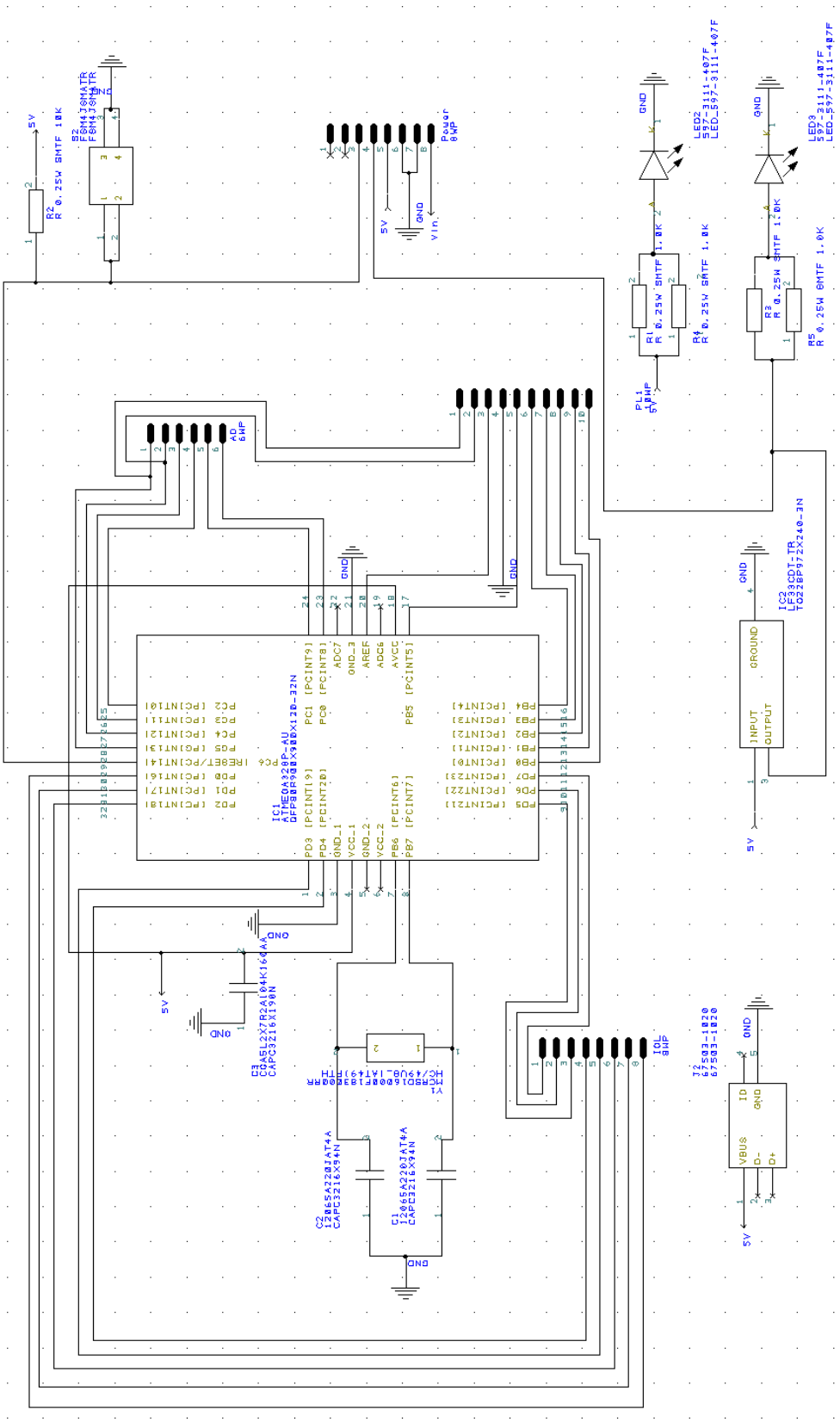
```
#include <Wire.h>
#include <PN532_I2C.h>
#include <PN532.h>
#include <NfcAdapter.h>

PN532_I2C pn532_i2c(Wire);
NfcAdapter nfc = NfcAdapter(pn532_i2c);
void setup() {
      Serial.begin(115200);
      Serial.println("NDEF Writer");
      nfc.begin();
}

void loop() {
    Serial.println("\nPlace a formatted Mifare Classic or Ultralight NFC tag on the reader.");
    if (nfc.tagPresent()) {
        NdefMessage message = NdefMessage();
        message.addUriRecord("Licence VEGA");

        bool success = nfc.write(message);
        if (success) {
          Serial.println("Success. Try reading this tag with your phone.");
        } else {
          Serial.println("Write failed.");
        }
    }
    delay(5000);
}
```

Annexe 3 :

Code Peer to Peer Adafruit

```
#include "SPI.h"
#include "PN532_SPI.h"
#include "snep.h"
#include "NdefMessage.h"

PN532_SPI pn532spi(SPI, 10);
SNEP nfc(pn532spi);
uint8_t ndefBuf[128];

void setup() {
    Serial.begin(115200);
    Serial.println("NFC Peer to Peer Example - Send Message");
}

void loop() {
    Serial.println("Send a message to Peer");

    NdefMessage message = NdefMessage();
    message.addUriRecord("Licence VEGA");

    int messageSize = message.getEncodedSize();
    if (messageSize > sizeof(ndefBuf)) {
        Serial.println("ndefBuf is too small");
        while (1) {
        }
    }

    message.encode(ndefBuf);
    if (0 >= nfc.write(ndefBuf, messageSize)) {
        Serial.println("Failed");
    } else {
        Serial.println("Success");
    }

    delay(3000);
}
```

Annexe 4 : Schéma de notre carte



PCB de notre carte :