Département Génie Électrique et Informatique Industrielle

INTERNET DES OBJETS

SUJET DE TRAVAUX PRATIQUES

21 février 2019 - 20:13:32

Michel GRIMALDI

Table des matières

Index des illustrations

Index des tableaux

OBJECTIFS

Il s'agit d'illustrer une technologie proche de l'internet des objets en utilisant des composants pré-existants et connus par les étudiants du département GEII.

Les objets, basés sur une carte Arduino Nano, sont connectés à un concentrateur, Raspberry PI, par le biais d'une liaison hertzienne à 2,4 GHz, transmetteurs NRF24.

Le concentrateur est connecté au réseau local, internet en vrai grandeur, par le biais d'une interface Ethernet. Un serveur MQTT assure la distribution des informations à l'ensemble des machines du réseau.

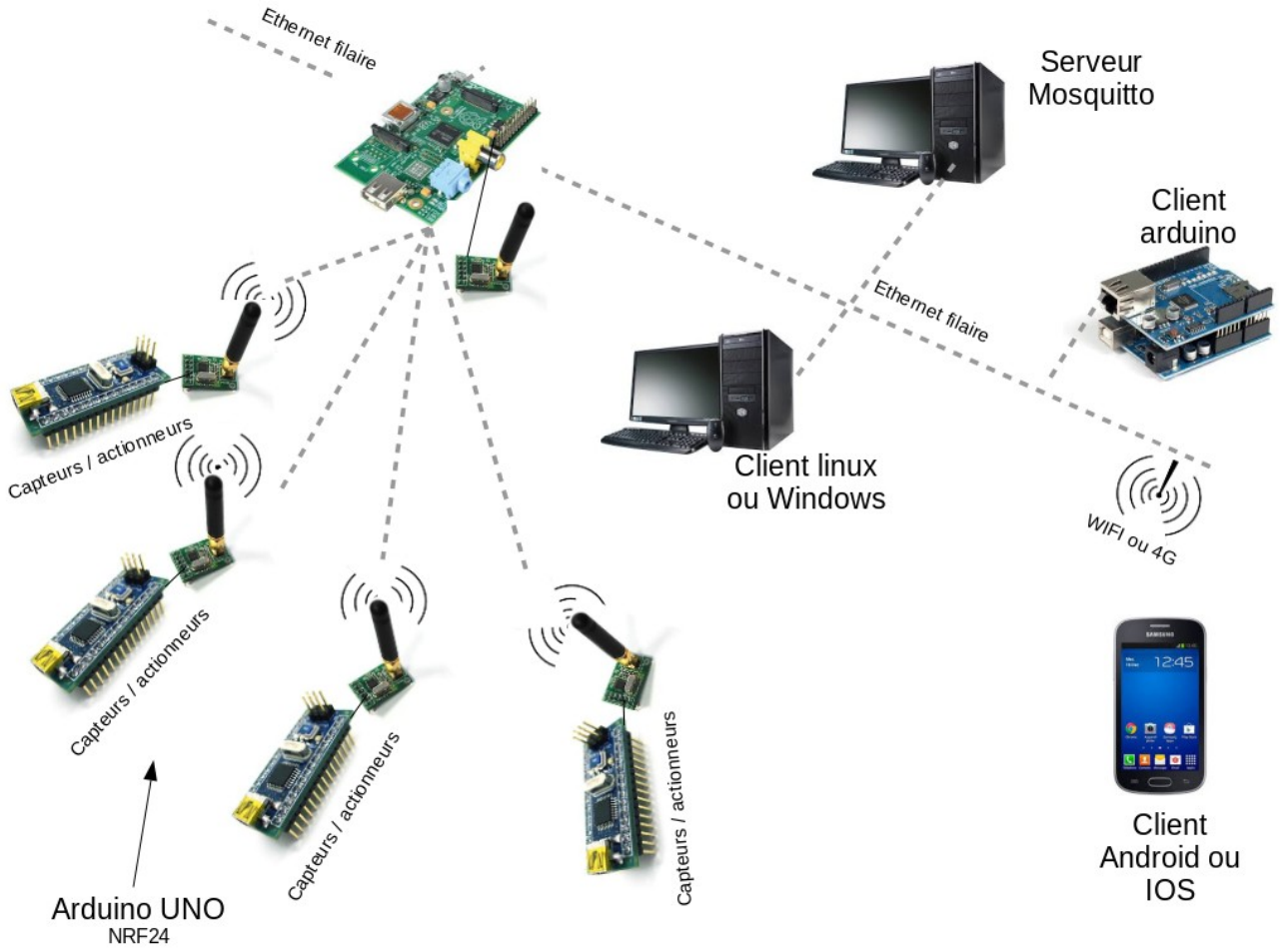


Figure 1 - Architecture fonctionnelle

PARTIE MATÉRIELLE

OBJETS COMMUNICANTS BASES SUR UN ARDUINO NANO

L'Arduino Nano est une petite carte complète à micro-contrôleur, basée sur la puce ATmega328. Il a plus ou moins les mêmes fonctionnalités que l'Arduino UNO, mais se présente dans un format différent ; il fonctionne avec un câble USB Mini-B.

L'Arduino Nano dispose d'un certain nombre de moyens pour communiquer avec un ordinateur, un autre Arduino, ou d'autres microcontrôleurs. L'ATmega328 fournit un port UART TTL (5V), qui est

disponible sur les broches numériques 0 (RX) et 1 (TX). Un FT232RL FTDI permet une communication série via le port USB en utilisant les pilotes FTDI (inclus avec le logiciel Arduino). Le RX et TX LED sur la carte clignote lorsque des données sont transmises via la puce FTDI et la connexion USB (mais pas pour la communication série sur les broches 0 et 1).

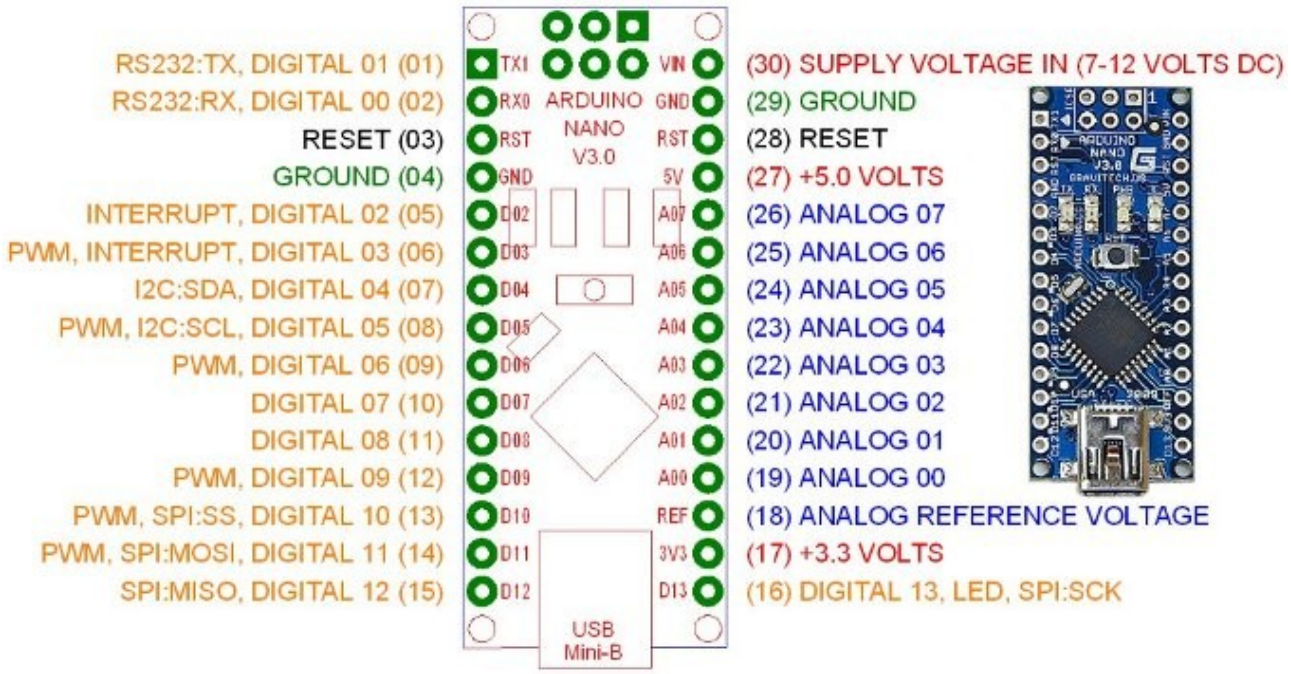


Figure 2 - brochage de la carte Arduino Nano

FONCTIONNEMENT DES OBJETS

Capteurs

Chaque objet connecté, équipé d'un ou plusieurs capteurs/actionneurs, émettra régulièrement ou lors d'événements de déclenchement spécifiques des trames sur le *tranceiver* NRF24. Ces trames contiennent les données qu'il est susceptible de délivrer (la température, la lumière, .etc..).
Chaque trame est constituée d'une suite de caractères (32 au maximum) ayant la structure suivante :

`tttt/ii;ddd,...,ddd`

où :

`tttt`	est un mnémonique de 4 caractères représentant le type de capteur (SENSOR_TYPE) ou d'actionneur (ACTUATOR_TYPE) connecté à l'objet émetteur de la trame (*cf.* tableau 1, ci-dessous).
`/`	Le caractère délimiteur /
`ii`	est un identifiant unique de 2 digits hexadécimal représentant l'objet émetteur de la trame (DEV_NUMBER), *cf.* tableau 3, page 11. Il est compris entre 0x01 et 0xFE (1 à 254).
`;`	Le caractère délimiteur ;
`ddd,...,ddd`	les données spécifiques du capteur, séparées par des virgules (ex : température, lumière, *.etc.*) 24 caractères maxi pour ne pas dépasser les 32 caractères de la trame. (*cf.* tableau 1, ci-dessous).

Exemples :

text/03;22.3,19.7,25.0

L'objet n°3 envoie une information de température extérieure de 22,3°, min=19,7°, max=25.0°

joys/04;345,200,off

L'objet n°4 envoie une position de joystick : x=345, y=200, bouton non appuyé.

ligt/01;145

L'objet n°1 envoie une information de lumière de 145 lux

Un même objet peut envoyer des informations de plusieurs types (ex : température et lumière).

Actionneurs

Tous les objets sont équipés d'un ou plusieurs actionneurs, dont une led RGB connectée au microcontrôleur. L'état de ces actionneurs devra être modifié à la réception d'une trame équivalente à la précédente, dans laquelle le champs de données (ddd,...ddd) représente cet état.les trois niveaux de couleur (rouge, vert, bleu) à l'action à effectuer.

Exemple :

sund/03;1 jouer la mélodie n°1 sur l'objet n° 3

lrgb/01;255,0,0 jallume la led en rouge sur l'objet n° 1

Tableau 1 : récapitulatifs des fonctions

TYPE[1]	DONNÉES SPÉCIFIQUES	type[2]	DÉCLENCHEMENT
tbrd	Température de la carte (°C) température	c	Toutes les 10 minutes
lrgb	Allumage de la led RGB rouge,vert,bleu	a	A la demande
text	Température extérieure (°C) T instantanée ; T mini ;T maxi	c	Au changement, mais pas plus de 5 fois par secondes
joys	Joystick pos X , pos Y , switch (au repos, X=Y=0)	c	Au changement, mais pas plus de 5 fois par secondes
ligt	Capteur de lumière BH1750FVI valeurLux , mini60s , maxi60s	c	Au changement, mais pas plus de 5 fois par secondes
incl	inclinomètre basé sur un accéléromètre/gyroscope MPU-6050 - angles en degrés angleX, angleY	c	Au changement, mais pas plus de 5 fois par secondes
irdt	Détecteur de présence IR on/off	c	Au changement
irrc	Récepteur de télécommande IR code-touche[3]	c	Au changement
irtr	émetteur de télécommande IR code à émettre (SONY)	a	A la demande

1 **SENSOR_TYPE** ou **ACTUATOR_TYPE**
2 'c' = capteur, 'a' = actionneur
3 un caractère parmi: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #, L (left), R(right), U(up), D(down), \r(ok)

hygs	Capteur d'humidité du sol hygrométrie (%), trigger=on/off, mini60s , maxi60s	c	Au changement ou 60 minutes
bous	Boussole - angle en degrés angle/nord	c	Au changement, mais pas plus de 5 fois par secondes
sund	Actionneur d'alarme sonore n° mélodie (0, 1, 2)	a	A la demande
rfid	Lecteur de badge RFID n° du badge	c	Au changement
auth	Autorisation d'accès à un badge RFID n° du badge, 0|1	a	Après la réception d'une lecture RFID comme ci-dessus

Gestion des communications NRF24 :

Pour les transmissions sans fil entre les différents objets (Arduino nano) et le concentrateur (Raspberry PI), nous utiliserons des transmetteurs Radio 2,4 GHz.

NRF24L01+ 2.4GHz Antenna Wireless Transceiver Module
Data sheet complète : https://www.sparkfun.com/datasheets/RF/nRF2401rev1_1.pdf

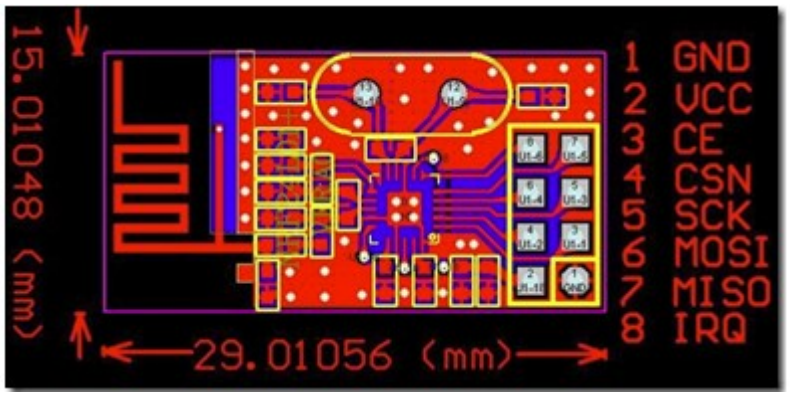


Figure 3 - brochage du *transceiver* radio 2,4 GHz

Le câblage des transmetteurs NRF24 et donné au tableau 2, ci-dessous.

Attention: les repères entre parenthèses correspondent aux numéros de broche de la carte Nano, – *cf.* figure 2, page 5, pour avoir la correspondance avec les broches Arduino.

Tableau 2 : câblage des NRF24 sur Arduino Nano

Broche Module	Broche Arduino
(1) GND	(4) GND
(2) VCC	(17) 3,3v
(3) CE	(9) D06
(4) CSN	(10) D07
(5) SCK	(16) SPI :SCK
(6) MOSI	(14) SPI:MOSI
(7) MISO	(15) SPI:MISO
(8) IRQ	

Pour la partie logicielle, vous utiliserez la bibliothèques C++ RF24, téléchargeable sur le WEB, (lien 2 , page 21). Elles est constituée d'une classe **RF24** définie en UML comme suit:

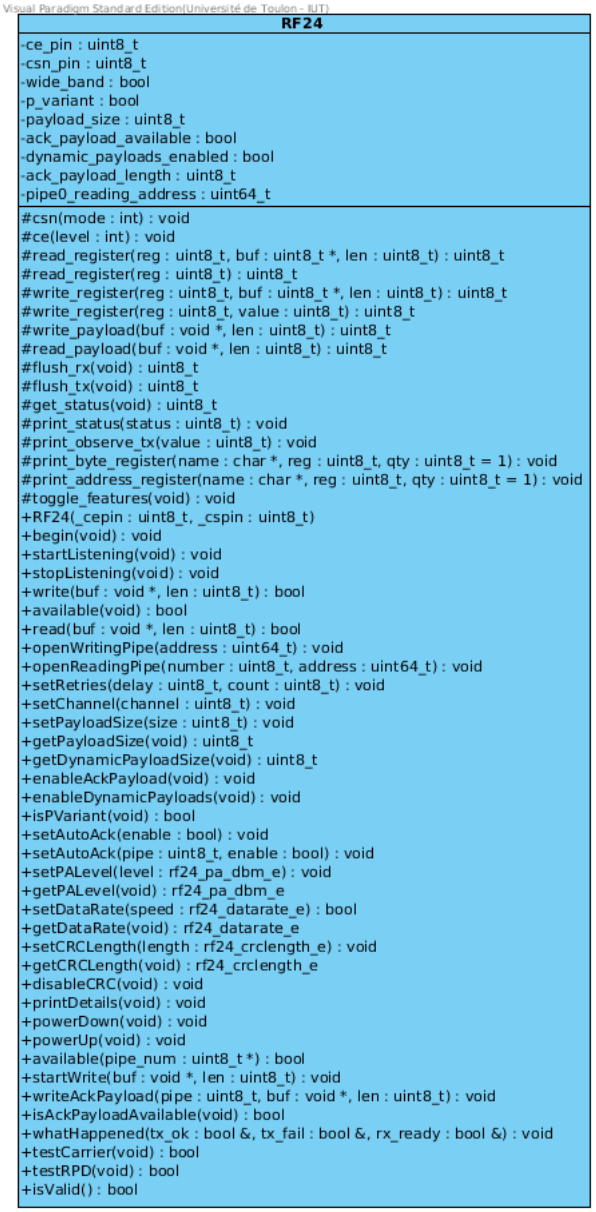


Figure 4 - représentation UML de la classe RF24

Les transmetteurs NRF24 permettent des communications point à point entre deux nœuds par le biais de canaux logiques de transmission appelés ***pipes***. Les communications se font sous la forme de trames de 32 octets utiles au maximum (***payload***).
Deux paramètres sont à prendre en compte pour une transmission entre deux points : le canal physique, c'est à dire la fréquence d'émission, et le canal logique, ***pipe*** *;* lorsqu'un émetteur envoie une trame sur un ***pipe***, seuls le, ou les récepteurs, « écoutant » ce ***pipe*** la reçoivent - sur la même fréquence bien entendu.

Logiquement, les *pipes* sont représentés par un mot de 5 octets. Dans le programme c++, une valeur du type entier long non signé sur 64 bits (**`uint64_t`**) est utilisée pour cela.

Chaque transmetteur NRF24 possède un *pipe* d'émission (*tx_pipe*) et 6 *pipes* de réception (*rx_pipe*), ce qui veut dire qu'il ne peut recevoir des données uniquement de 6 autres transmetteurs – sur un

canal (une fréquence) donné. Dans l'exemple de la figure 20, ci-dessous, le transmetteur noté PRX communique avec 6 autres transmetteurs. Vous pourrez remarquer une affectation typique des *pipes*.

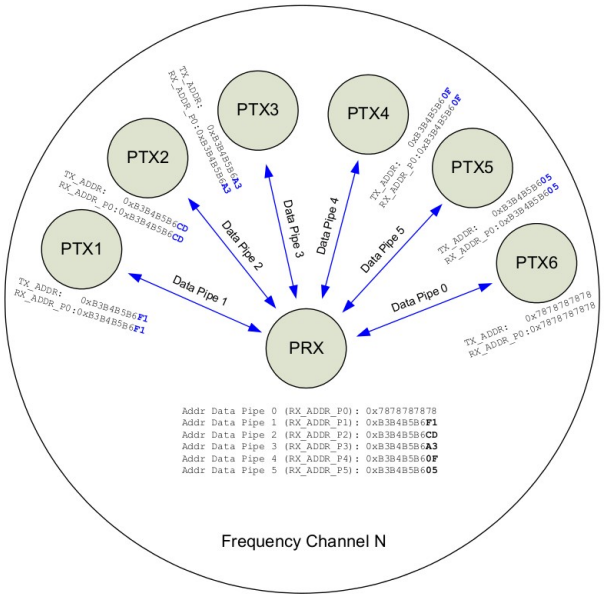


Figure 5 - exemple de choix des *pipes* RF24

Pour ce sujet de TP, si nous nous limitons à 6 objets connectés simultanément, nous vous proposons d'utiliser la répartition suivante, pour les numéros de *device* de 1 à 6 :

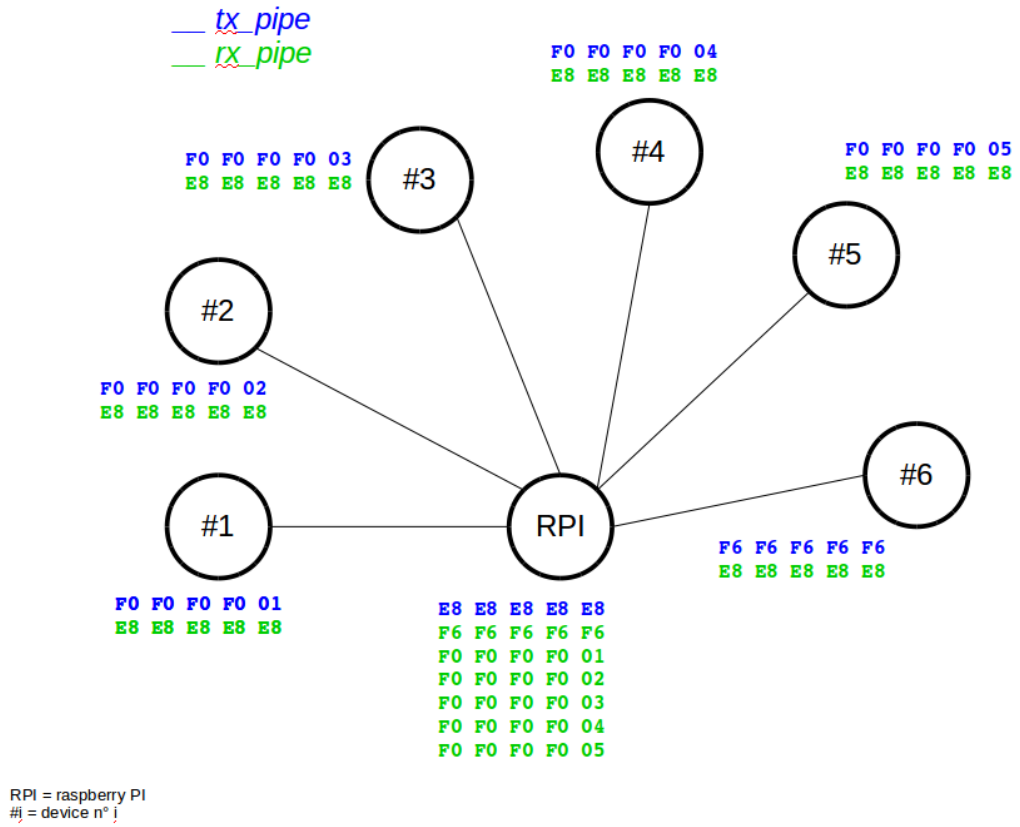


Figure 6 - choix des *pipes* RF24 pour le TP (< 6 objets connectés)

Dans le cas où l'on veut faire communiquer plus de 6 objets[4] il est nécessaire de mettre un transmetteur en « relais » (*meshing*). La répartition suivante sera alors utilisée.

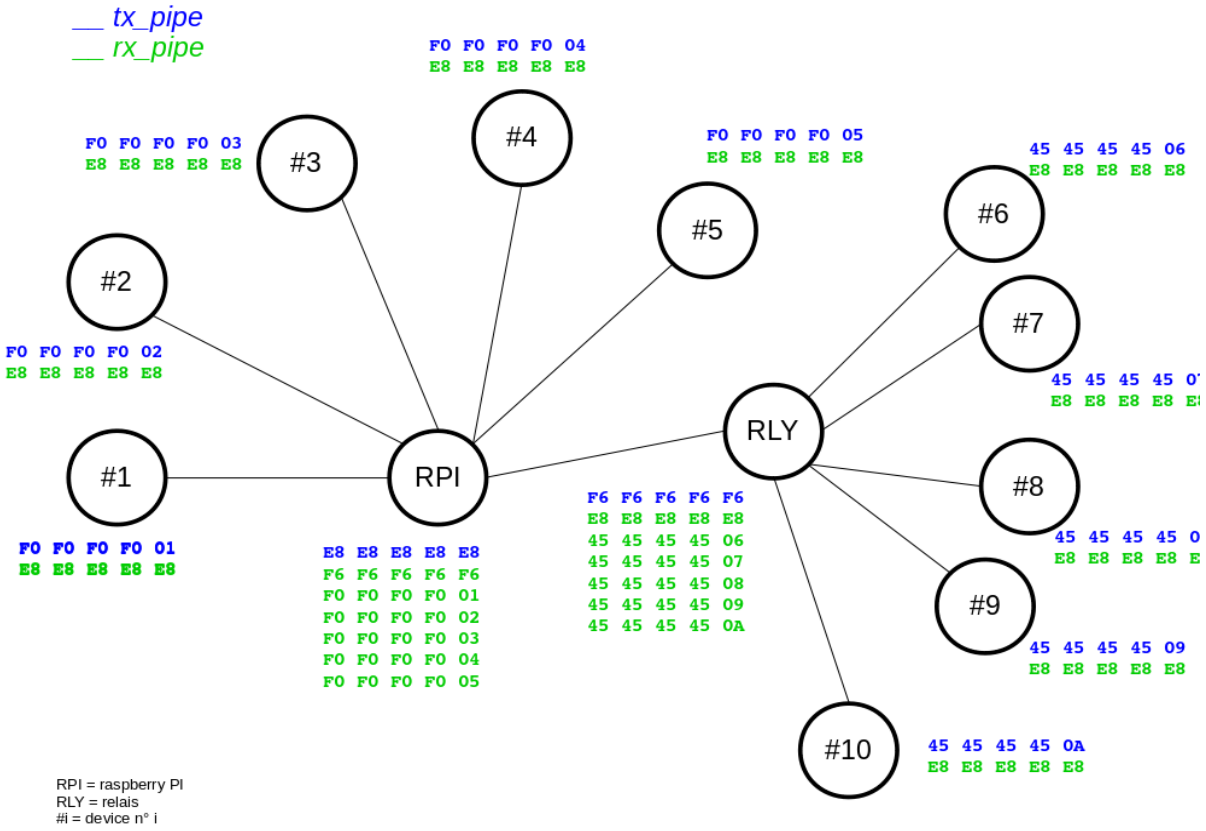


Figure 7 - choix des *pipes* RF24 pour le TP (> 6 objets connectés)

Dans le programme c++, ces deux configurations seront représentées par les lignes suivantes :

```
const uint64_t rxPipe = 0xE8E8E8E8E8LL;
const uint64_t txPipe = (DEV_NUMBER <6 ? 0xF0F0F0F000LL | (DEV_NUMBER & 0xff) :
                                         0xF6F6F6F6F6LL);
```

ou, si >6 devices,

```
const uint64_t rxPipe = 0xE8E8E8E8E8LL;
const uint64_t txPipe = (DEV_NUMBER <6 ? 0xF0F0F0F000LL |
                            (DEV_NUMBER & 0xff) :
                                         0x4545454500LL | (DEV_NUMBER & 0xff));
```

Avec, pour chaque objet connecté, le numéro de *device* (DEV_NUMBER), conforme aux tableau 3 ci-dessous :

4 - ceci peut être notre cas s'il y a plus de 12 étudiants par groupe, soit plus de 6 binômes, en même temps.

Tableau 3 : affection des numéros de *device* : DEV_NUMBER

DeviceNumber	Objet
1	Soil Hygrometer
2	Joystick
3	Accelerometer
4	IR detector + sound
5	IR remote
6	Light BH1750FVI
7	Boussole
8	RFID Reader

Une exemple de programme type permettant d'envoyer et de recevoir des données via le NRF4 a été présenté en TD.

Nota: tous les objets connectés devront envoyer une trame avec leur température interne toutes les 10 minutes. La fonction de lecture de la température du cœur est disponible sur le WEB, voir lien 1, annexe page 21

LES CAPTEURS/ACTIONNEURS

Attention: les repères entre parenthèses correspondent aux numéros de broche de la carte Nano et non aux numéros Arduino – *cf.* figure 2, page 5, pour avoir la correspondance avec les broches Arduino.

LA LED RGB

La led RGB équipant tous les objets est connectée à trois ports PWM de la carte Arduino nano. Le câblage peut être différents d'un objets à l'autre, voir les tableaux ci-dessous.

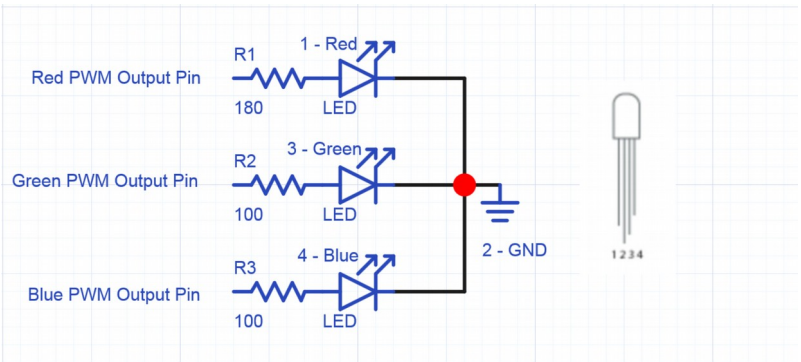


Figure 8 - Câblage de la led RGB

JOYSTICK

La lecture du joystick se résume à deux lectures analogiques des valeurs X et Y, et une lecture digitale pour le bouton - le bouton étant directement câblé sur l'entrée, pensez à activer la résistance de *pullup* interne de l'Arduino,

Figure 9 - joystick analogique

Tableau 4 : joystick

Broche Module	Broche Arduino
(1) GND	(4) GND
(2) VCC	(17) 3,3v
(3) VRx	(26) A07
(4) VRy	(25) A06
(5) SW	(5) D02
Led RGB	(13)D10, (12)D09, (6)D03

CAPTEUR DE LUMIÈRE

BH1750FVI Digital Light Intensity Sensor Module. Une classe spécifique est disponible sur le WEB, (lien 5, page 21)

Figure 10 - capteur de lumière

Tableau 5 : lumière

Broche Module	Broche Arduino
(1) GND	(4) GND
(2) VCC	(17) 3,3v
(3) VRx	(26) A07
(4) VRy	(25) A06
(5) SW	(5) D02
Led RGB	(13)D10, (12)D09, (6)D03

ACCÉLÉROMÈTRE/GYROSCOPE

6DOF MPU-6050 3 Axis Gyro With Accelerometer Sensor. Une classe spécifique est disponible sur le WEB, (lien 4, page 21).

Figure 11 - accéléromètre gyroscope numérique

Tableau 6 : accéléromètre

Broche Module	Broche Arduino
VCC	(17) 3,3v
GND	(4) GND
SCL	(24) A05
SDA	(23) A04
XDA	
XCL	
ADD	
INT	(5) D02
Led RGB	(13)D10, (12)D09, (6)D03

DÉTECTEUR DE PRÉSENCE PIR

La lecture du détecteur de présence se résume en une simple lecture digitale. Le détecteur comporte deux potentiomètres de réglage du seuil de détection et du temps de maintien - ne pas modifier.

Figure 12 - détecteur de présence infra-rouge

Tableau 7 : détecteur de présence

Broche Module	Broche Arduino
VCC	(27) 5v
OUT	(05) D02
GND	(29) GND
Led RGB	(13)D10, (12)D09, (8)D05

CAPTEUR D'HUMIDITÉ DANS LE SOL

La lecture du capteur d'humidité se résume en une simple lecture analogique. En fonction d'un seuil réglable par un potentiomètre, on peut également avoir une information digitale du type **en dessous** ou **en dessus** du seuil.

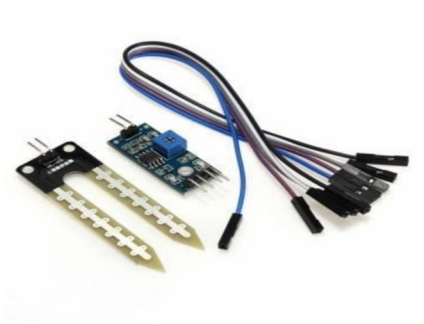


Figure 13 - Soil Hygrometer Humidity Detection Module Moisture Sensor

Tableau 8 : hygrométrie du sol

Broche Module	Broche Arduino
VCC	(27) 5v
GND	(29) GND
Digital	(05) D02
Analog	(26) A7
Led RGB	(13)D10, (12)D09, (6)D03

Description :
Soil Hygrometer Detection Module Soil Moisture Sensor :
when the soil is dry, the module outputs a high level, whereas output low.
Operating voltage: 3.3V~5V.
Adjustable sensitivity (shown in blue digital potentiometer adjustment)
Dual output mode,analog output more accurate.
A fixed bolt hole for easy installation.
With power indicator (red) and digital switching output indicator (green).
Having LM393 comparator chip, stable.
Panel PCB Dimension: 3cm x 1.5cm. Soil Probe Dimension: 6cm x 2cm.
Cable Length: 21cm. VCC: 3.3V-5V. GND: GND.
DO: digital output interface(0 and 1).
AO: analog output interface.

TRANSDUCTEUR PIÉZOÉLECTRIQUE

L'actionneur qui permet d'émettre des sons ou des mélodies sur le transducteur, utilisera une classe spécifique, **Tone**, disponible sur le WEB (lien 8, page 21).

Figure 14 - *Piezo electric transducer*

Tableau 9 : buzzer

Broche Module	Broche Arduino
PWM	(12) D09
GND	(29) GND

LECTEUR RFID MFRC522 - pas encore implémenté!

Une bibliothèque spécifique contenant une classe RFID est disponible sur le WEB (lien 10, page 21). Le lecteur utilisant l'interface SPI, il faudra vérifier que celle-ci est compatible avec la gestion du RF24.

Figure 15 - RFID reader

Tableau 10 : RFID

Broche Module MFRC522	Broche Arduino
SDA	(13) SPI:SS
SDK	(16) SPI:SCK
MOSI	(14) SPI:MOSI
MISO	(15) SPI:MISO
IRQ	
GND	(4) GND
RST	(5) D02
3,3 V	(17) 3,3v
Led RGB	(8) D5, (12) D09, (6) D03

MIFARE NXP card reader and the tags communicate using a 13.56MHz electromagnetic field. (ISO 14443A standard tags)
http://idehack.com/dl/rfid_lib.rar

BOUSSOLE

Digital Magnetic Compass HMC5883L. Une classe spécifique est disponible sur le WEB (lien 7, page 21.

Figure 16 - Boussole

Tableau 11 : boussole

Broche Module	Broche Arduino
3V3	(17) 3,3v
GND	(4) GND
SCL	(24) A05
SDA	(23) A04
DRDY	nc
Led RGB	(13)D10, (12)D09, (6)D03

CAPTEUR DE VIBRATION- pas encore implémenté!

Figure 17 - capteur de vibration

Tableau 12 : vibration

Broche Module	Broche Arduino
VCC	(17) 3,3v
GND	(4) GND
D0	(5) D02
Led RGB	(13)D10, (12)D09, (6)D03

TÉLÉCOMMANDE INFRAROUGE

Une bibliothèque spécifique, permettant de décoder les signaux infrarouges reçus et/ou d'émettre des trames sur une LED IR, selon les principaux protocoles existants, IRremote, est disponible sur le WEB (lien 9, page 21).

Figure 18 - le kit de télécommande IR

Tableau 13 : télécommande IR

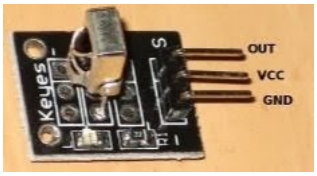
Broche Module	Broche Arduino
VCC	(17) 3,3v
OUT	(05) D02
GND	(29) GND

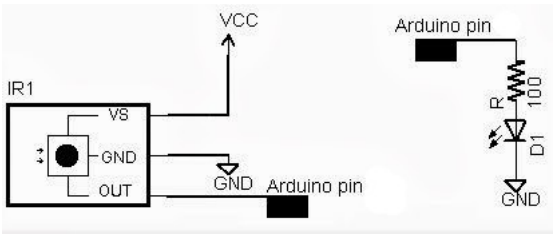
Récepteur IR

Tableau 14 : émetteur IR

Broche Module	Broche Arduino
LED	(06) D04
Led RGB	(13)D10, (12)D09, (6)D03

Émetteur IR





RASPBERRY PI

Le Raspberry Pi est un nano-ordinateur mono-carte à processeur ARM conçu par le créateur de jeux vidéo David Braben, dans le cadre de sa fondation Raspberry Pi.

Cet ordinateur, qui a la taille d'une carte de crédit, est destiné à encourager l'apprentissage de la programmation informatique; il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération.

Le Modèle A original :

- Processeur : ARM1176JZF-S (ARMv6) 700 MHz Broadcom 28351(dispose d'un décodeur Broadcam VideoCore IV, permettant le décodage H.264 FullHD 1080P et d'un VFPv2 pour le calcul des opérations à virgule) ;
- RAM : 256 Mo
- 2 Sorties vidéo : Composite et HDMI ;
- 1 Sortie audio stéréo Jack 3 5 mm (sortie son 5.1 sur la prise HDMI) ;
- Unité de lecture-écriture de carte mémoire : SDHC / MMC / SDIO ;
- 1 Port USB 2.0 ;
- Prise pour alimentation Micro-USB (consommation : 400 mA + périphériques) ;
- Des entrées / sorties supplémentaires sont accessibles directement sur la carte mère via des pins 3v3 55: GPIO, S2C, I2C, SPI ;
- API logicielle vidéo : OpenGL : version embarquée OpenGL ES 2.0 ;
- Décodage vidéo : 1080p30 H.264 high-profile.

Le Modèle B+ - Ce modèle est commercialisé depuis juillet 2014.

Différences par rapport au modèle initial :

- GPIO 40 broches
- 4 ports USB 2.0 et meilleur comportement en cas de surcharge
- micro SD
- réduction de consommation de 3,5 W à 3 W
- meilleur circuit audio

Figure 19 - la carte Raspberry PI rev B+

Dans notre projet, le Raspberry PI est utilisé comme « nœud » pour connecter les objets à internet, pour cela, il est également équipé d'un transmetteur RF24[5].

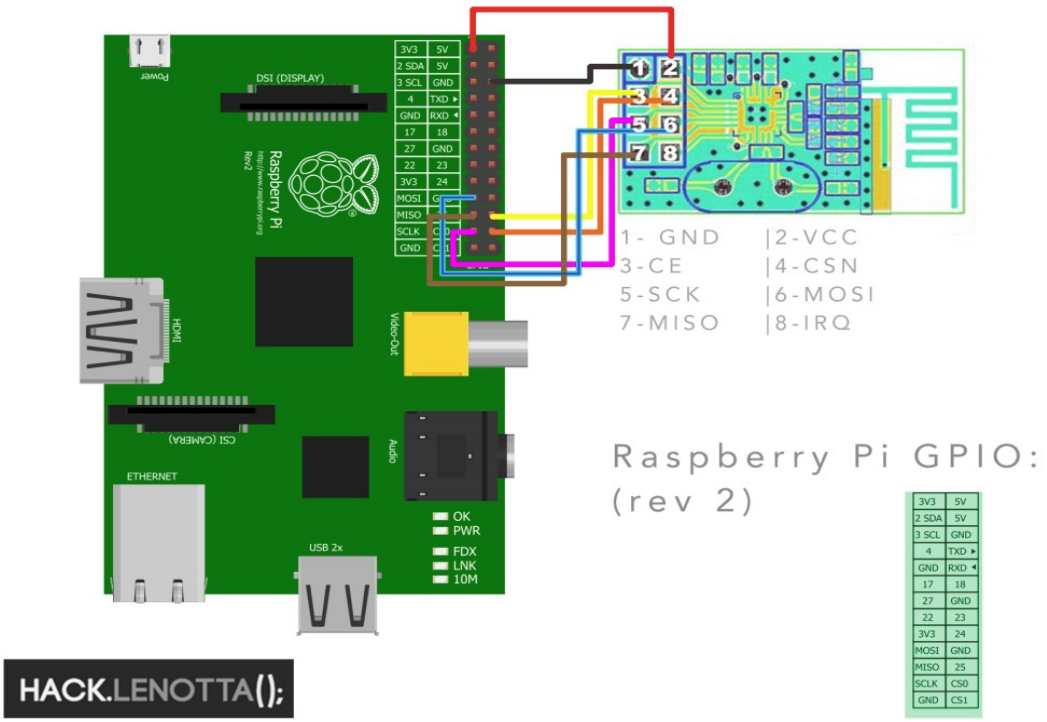
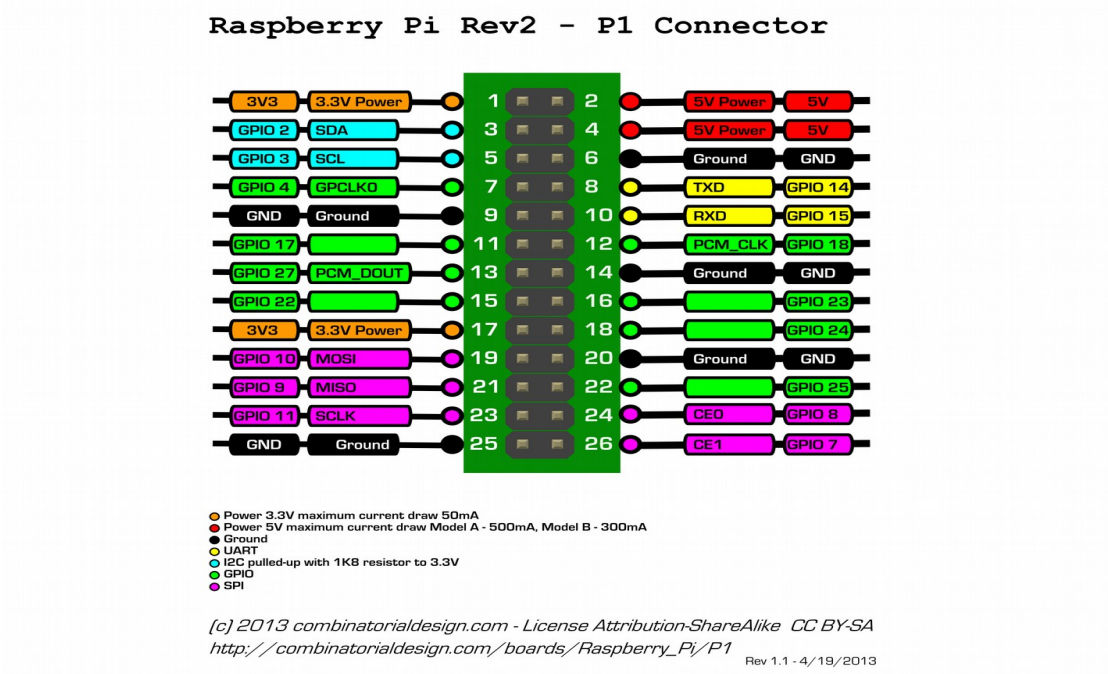


Figure 20 - connexion du NRF24 au Raspberry PI

5 Toute la partie Raspberry sera mise à disposition en TP et ne nécessitera aucune intervention de votre part.



PARTIE LOGICIELLE

Une fois les objets définis au niveau de leur hardware, la communication entre ceux-ci reste l'une des choses essentielles.
[6]Dans les protocoles de communication que l'on retrouve le plus souvent, vous avez bien sûr http (et ses fameuses API Rest (ou pas)), xPL, XAP, XMPP, SNMP. Vous pouvez également développer votre propre protocole sur TCP/IP.

Un protocole devrait pourtant tenir rapidement le « haut du pavé » et devenir le standard de l'i.o.t (comprendre l'internet des objets (ou « internet of things » pour reprendre le terme officiel) : MQTT.

Et pour couronner le tout, MQTT est devenu depuis quelques temps un standard d'échange OASIS. (OASIS (*Organization for the Advancement of Structured Information Standards*) est un consortium chargé d'aider au développement, à la convergence et l'adoption de standards ouverts pour « la société mondiale de l'information » https://www.oasis-open.org/member-sections/

MQTT est un service de messagerie TCP/IP simple et extrêmement léger dans le sens où des messages, limités à 256Mo, sans contrainte de structure peuvent être transmis.

Les messages sont envoyés par des publieurs (*publishers*) sur un canal (une chaîne d'information si vous voulez) appelé *Topic*. Ces messages peuvent être lus par les abonnés (*subscribers*). Les Topics (ou les canaux d'informations) peuvent avoir une hiérarchie qui permet de sélectionner finement les informations que l'on désire.

Par exemple, un topic /sensor/temperature/salle206 ne donnera que les températures de la salle 206

6 Rédaction fort inspirée de l'excellent site: http://blog.guiguiabloc.fr/index.php/2014/11/13/mqtt-faites-communiquer-vos-objets-simplement/

si je m'y abonne, et bien sûr que ma sonde de température dans la salle 206 publie sur ce *topic*, sa valeur régulièrement.
Vous pourrez aussi vous abonner au *topic* **/sensor/temperature/#** pour avoir toutes les températures, ou au *topic* **/sensor/#** pour avoir toutes les remontées de toutes les sondes (plus de détails sur les « wildcards » sur les *topics* sont disponibles ici http://mosquitto.org/man/mqtt-7.html)

Maintenant que nous vous avons mis l'eau à la bouche, je l'espère, vous pouvez continuer cette lecture directement sur le site
http://blog.guiguiabloc.fr/index.php/2014/11/13/mqtt-faites-communiquer-vos-objets-simplement/

Travail préalable:
Installer le serveur MQTT "Mosquitto" :
choisir une machine du réseau, dont l'adresse est 192.168.0.XX, pour y installer le serveur mosquitto - cette adresse sera utilisée par la suite.

```
sudo apt-get install mosquitto
```

Sur les autres machines du réseau, installer les clients de souscription et de publication :
```
sudo apt-get install mosquitto-clients
```

Vérification :
sur une machine ou plusieurs machines du réseau, s'abonner à un service (*topics*) **hello/world** par exemple :
```
mosquitto_sub -d -h 192.168.0.XX -t hello/world
```
Le client s'active !

Sur une autre machine du réseau, publier une donnée sur ce serveur, au même *topic* **hello/world** :
```
mosquitto_pub -d -t hello/world -m "Voici ma première publication." -h 192.168.0.XX
```

Vérifiez la notification sur les clients ayant souscrit au *topic* **hello/world**.

Analysez les échanges entre les diverses machines avec **Wireshark**.

Documentation de référence :
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

Réalisation du projet

Chaque binôme développera un objet spécifique, tiré au sort parmi les objets disponibles.
Le travail à faire consistera à :

1. OBJET CONNECTE: écrire le programme embarqué sur Arduino Nano (objet connecté) conformément aux spécifications ci-dessus. (lecture du ou des capteurs, mise en forme des trames, envoi au concentrateur, gestion des actionneurs le cas échéant, envoi de la température de la carte). Vous pouvez utiliser l'environnement de développement Arduino IDE, mais mon expérience dans ce domaine me conduit à vous **conseiller fortement** d'utiliser un environnement plus évolué du type Eclipse, Netbeans, ou QtCreator permettant la coloration syntaxique, la complétion de code, l'indentation automatique, le *refactoring* .etc.
   Vous trouverez un tutoriel sur mon site (http://grimaldi.univ-tln.fr/developpement-dapplications-arduino-avec-qtcreator.html) pour installer QtCreator avec Arduino.

2. CLIENT MQTT: écrire un programme c++ sur un ordinateur du réseau permettant d'illustrer graphiquement l'utilisation de cet objet à travers le protocole MQTT, sous Qt (voir exemple de programme à: http://grimaldi.univ-tln.fr/category/files/client-MQTT.zip) et mettant en œuvre, si possible des courbes ; voir la bibliothèque QWT.
   Exemple d'interfaçage : déplacer une figure avec le joystick ou de la télécommande IR, afficher une courbe de température ou de lumière, notifier la présence détectée par le capteur de présence IR par un image sur l'interface, afficher la rotation sur une boussole, l'inclinaison sur un horizon artificiel, .etc.
3. CLIENT ANDROID: porter l'application précédente sur Android.

Le planning ci-dessus représente une indication de la répartition des taches à effectuer. Les séances peuvent se chevaucher à condition que l'objectif final soit atteint.

Programme sur Arduino mettant en œuvre : les capteurs de l'objet utilisé la mise en forme des données la communication RF24 vers la Raspberry PI	Programme client mettant en œuvre: le protocole MQTT la récupération de données issues des capteurs un interface graphique illustrant les données ci-dessus	Finalisation et mise au point des programmes Introduction des actionneurs Démonstration Application Android

Évaluation :

- partie Arduino, respectant l'ensemble des spécifications[7] 6 points
- partie application Qt mettant en œuvre le protocole MQTT et une interface graphique illustrant l'utilisation du/des capteurs[7] 5 points
- participation, motivation, autonomie[7] 4 points
- compte rendu : précision de la rédaction, et professionnalisme[8] 5 points

L'ensemble des codes sources, commentés et documentés, sera joint au compte rendu (au maximum 1 semaine après la dernière séance de TP). Prenez vos dispositions ; passé ce délai, et quel qu'en soit le motif, aucun compte rendu ne sera plus accepté, avec la note en conséquence.

Le style de programmation, indentation, choix du nom des variables, respect de la casse, commentaires, *.etc*. entrera en compte dans la notation.

7 parties évaluées durant les séances par l'enseignant

8 cette partie sera évaluée sur le compte rendu ***impérativement 1 semaine après la dernière séance de TP*** ; le non-rendu ou retard sera sanctionné par la note de 0 à cette partie

ANNEXES

Liens de récupération des bibliothèques nécessaires:

1. Fonction d'obtention de la température du cœur: https://playground.arduino.cc/Main/InternalTemperatureSensor
2. La classe RF24 - mise en œuvre du transmetteur NRF24 : https://github.com/nRF24/RF24
3. La classe I2Cdev - accès au bus I2C : https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/I2Cdev
4. La classe MPU6050 - accéléromètre gyroscope :https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
5. La classe BH1750 - capteur de lumière :https://github.com/claws/BH1750
6. La classe RFID - lecture/écriture de cartes à puce :http://idehack.com/dl/rfid_lib.rar
7. La classe HMC5883L - boussole :https://github.com/jarzebski/Arduino-HMC5883L
8. La classe Tone, permettant de jouer un son ou une mélodie: http://downloads.arduino.cc/libraries/github.com/bhagman/Tone-1.0.0.zip
9. La bibliothèque IRremote, contentant entre autres les classes IRrecv et IRsend, permet de décoder les principales télécommandes à infrarouge: http://downloads.arduino.cc/libraries/github.com/z3t0/IRremote-2.2.3.zip
10. La classe RFID, permettant de lire et d'écrire des cartes à puces : http://idehack.com/dl/rfid_lib.rar

Programme type:

Si vous avez du mal à démarrer le programme Arduino, une structure typique est donnée à titre d'exemple ci-dessous[9] - Vous pouvez vous en inspirer partiellement ou totalement sachant que les parties surlignées en jaune seront spécifiques à votre application:

```
//_____________________________________________________________________________
//  IUT DE TOULON - Département GEII
//  TP Internet des objets
//
//  titre de programme - Auteurs
//
//  si le nombre de devices est >6, on peut activer un relais permettant d'étendre
//  le réseau à 10 devices en dé-commentant la ligne suivante:
//#define WITH_RELAY        // avec un noeud de relais (> 6 objets)

//  La fréquence radio (canal de transmission) doit être identique pour tous les objets
//  F0 = 2400 + RF_CH [MHz]
#define RF24_CHANNEL 0x2A             // f = 2400 + 42 = 2,442 GHz
#define TRACE_MODE                    // le mode trace pour mise au point
//_____________________________________________________________________________
//    définition propre à l'objet connecté courant
#define VERSION "objet - 1.0 - Auteurs - 2018"
#define DEV_NUMBER 2                        // exemple objet n° 2
#define SENSOR_TYPE "joys"                  // type de capteur: joystick
#define ACTUATOR_TYPE "rgbl"                // type d'actionneur; led rgb
```

9 Notez le style de programmation (convention d'écriture, indentation, .etc.)

```
//_____________________________________________________________________________________
//    connections hardware de l'objet:
//    définir ici les pin utilisées par les capteurs et actionneurs:
//_____________________________________________________________________________________
//                  partie Arduino
#include <Arduino.h>
#include <string.h>
#include "printf.h"
//                  partie transmetteur RF24
#include "nRF24L01.h"
#include "RF24.h"

//  canaux logiques - voir sujet de TP
#ifdef WITH_RELAY                       // > 6 devices
const uint64_t rxPipe = 0xE8E8E8E8E8LL;                 // rx from raspberry
const uint64_t txPipe = (DEV_NUMBER <6 ? 0xF0F0F0F000LL |  (DEV_NUMBER & 0xff) :
                                         0x4545454500LL | (DEV_NUMBER & 0xff));
#else                                   // <= 6 devices
const uint64_t rxPipe = 0xE8E8E8E8E8LL;                 // rx from raspberry
const uint64_t txPipe = (DEV_NUMBER ==6 ? 0xF6F6F6F6F6LL : 0xF0F0F0F000LL | (DEV_NUMBER & 0xff));
#endif
//                                      connections hardware du RF24
#define RF24_CEPIN ?                                            // pin ce du spi
#define RF24_CSPIN ?                                            // pin cs du spi
//_____________________________________________________________________________________
//                  la Led et la température du coeur
#define LED_PIN  ?                          // nano builtin led
#define LED_BLINK_DELAY 1000UL               // période de clignotement led
#define CORE_TEMP_DELAY 60000UL              // période d'envoi de la température du coeur
//_____________________________________________________________________________________
//   fonction locales diverses
double getCoreTemp(void);                    // température du coeur
//_____________________________________________________________________________________
//                  P R O G R A M M E   P R I N C I P A L
//_____________________________________________________________________________________
int main()
{
    // variables and objets locaux
    bool ledstate = true;                   // état de la led
    unsigned long previousLedMillis = 0;        // l'instant de la dernière mise à jour de la LED
    unsigned long previousTempMillis = 0;       // l'instant de la dernière température du coeur
    unsigned long currentMillis;

    RF24 radio(RF24_CEPIN, RF24_CSPIN);     // instance représentant le rf24
    char payload[32+1] = {0};               // rx/tx buffer --> payload

    // variables liées au capteur
    // mettre ici les variables permettant de lire et de gérer
    // le capteurs/actionneurs
    int valeur, valeurPrecedente;       // par exemple

    // Initialisation des bibliothèques Arduino
    init();
    Serial.begin(115200);
    Serial.println(VERSION);

    // initialisation et préparation des capteurs:
    // affectation des pin en entrée ou sortie, activation des pullup, .etc.
    // pin de la led en sortie
    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, 0);

    // initialisation du RF24
    radio.begin();
    radio.enableDynamicPayloads();      // payloads de tailles variables
    radio.setDataRate(RF24_2MBPS);      // 2MBPS
    radio.setPALevel(RF24_PA_MAX);      // puissance maxi = 0dBm (conso: 11 mA)
    radio.setChannel(RF24_CHANNEL);     // canal physique
    radio.setRetries(15,15);            // delay 1ms, 3 retries, 0 - 250us, 15 - 4000us
    radio.openWritingPipe(txPipe);      // canal logique d'envoi
    radio.openReadingPipe(1,rxPipe);    // canal logique de réception
    radio.setAutoAck(true);             // le RF24 accuse réception automatiquement
    radio.startListening();             // mode réception
    #ifdef TRACE_MODE
    printf_begin();
    radio.printDetails();               // affiche les détails
#endif
```

```
    // boucle infinie -> loop()
    while (1){                          // boucle infinie
        currentMillis = millis();           // on prend le temps
        // lecture du ou des capteurs
        // lire les capteurs ici
        valeur = rand();        // pour simuler un capteur
        // est-ce le moment d'envoyer quelque chose ?
        if (valeur!=valeurPrecedente){    // quelque chose a changé
            // sauvegarde de l'état précédent des valeurs capteurs
            valeurPrecedente=valeur;
            //                              formation du payload
            sprintf(payload,"%s/%02d;%d", SENSOR_TYPE, DEV_NUMBER, valeur);

            // verifie si personne n'émet en ce moment et envoi
            if (!radio.testCarrier()){
                // envoie le payload
                radio.stopListening();              // mode émission
                radio.write((void *)payload, strlen(payload)+1);
                radio.startListening();             // mode réception
#ifdef TRACE_MODE
                Serial.print("send:");
                Serial.println(payload);
#endif
            }
        }  // quelque chose a changé
            // quelque chose est il arrivé entre temps ?
            uint8_t pipe_num;
            if (radio.available(&pipe_num)){
                int pls = radio.getDynamicPayloadSize();        // taille du payload
                if (pls>=1 && pls<32){
                    radio.read((void*)payload, pls);      // lecture
                    payload[pls]=0;
                    // traitement du payload, ex: "joys/02;25,56,off"
                    char devtype[5];    // type du device
                    int devnumber;      // n° du device
                    char args[25];      // arguments
                    sscanf(payload,"%4s/%d;%s", devtype, &devnumber, args);
                    if (devnumber==DEV_NUMBER){      // est-ce pour nous ?
                        // traiter la commande ici!
                    }
#ifdef TRACE_MODE
                    Serial.print("receive from ");
                    Serial.print(pipe_num);
                    Serial.print(" : ");
                    Serial.println(payload);
#endif
                }
            }
        // clignotement de la LED de vie
        if (currentMillis - previousLedMillis >= LED_BLINK_DELAY) {
            // sauve le temps de la dernière mise à jour de la LED
            previousLedMillis = currentMillis;
            digitalWrite(LED_PIN, ledstate? HIGH : LOW);
            ledstate = ! ledstate;
        }
        // envoi de la température du coeur
        if (currentMillis - previousTempMillis >= CORE_TEMP_DELAY){
            // sauve le temps du dernier envoi
             previousTempMillis = currentMillis;
             double temp = getCoreTemp();
             sprintf(payload,"tbrd/%02d;%d.%d",DEV_NUMBER, (int)temp, ((int)temp*10)%10);

             // verifie si personne n'émet en ce moment et envoi
             if (!radio.testCarrier()){
                 // envoie le payload
                 radio.stopListening();               // mode émission
                 radio.write((void *)payload, strlen(payload)+1);
                 radio.startListening();              // mode réception
 #ifdef TRACE_MODE
                 Serial.print("send:");
                 Serial.println(payload);
 #endif
             }
        }
    }       // boucle infinie
}
//_____________________________________________
```

```
// température du coeur
double getCoreTemp(void)
{
  unsigned int wADC;
  double t;
  // The internal temperature has to be used
  // with the internal reference of 1.1V.
  // Channel 8 can not be selected with
  // the analogRead function yet.
  // Set the internal reference and mux.
  ADMUX = (_BV(REFS1) | _BV(REFS0) | _BV(MUX3));
  ADCSRA |= _BV(ADEN);  // enable the ADC
  delay(20);            // wait for voltages to become stable.
  ADCSRA |= _BV(ADSC);  // Start the ADC
  // Detect end-of-conversion
  while (bit_is_set(ADCSRA,ADSC));
  // Reading register "ADCW" takes care of how to read ADCL and ADCH.
  wADC = ADCW;
  // The offset of 324.31 could be wrong. It is just an indication.
  t = (wADC - 324.31 ) / 1.22;
  // The returned temperature is in degrees Celsius.
  return (t);
}
```


SAMSUNG
SONOS
nest
ONKYO
Integra
xfinity
marantz
Honeywell
72°
Pioneer
Panasonic
DENON
YAMAHA
vacmote
SAMSUNG
Dimplex