INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

ENTRE LES MAILLES
Association

Projet de réalisation technique

SYSTEME DE COMMANDE POUR CINEMA INTERACTIF

Edouard Callens – Gabriel Lepetit-Aimon
INSA LYON | GENIE ELECTRIQUE

Table des matières

Introduction

En tant qu'élèves ingénieurs en 4ième année de Génie Electrique à l'INSA de Lyon, nous sommes amenés à réaliser un Projet de Réalisation Technique durant un semestre. Ce projet est une rampe vers le monde de l'ingénierie puisqu'il permet de travailler sur une réalisation concrète répondant à un réel besoin. Nous sommes donc amenés à utiliser nos compétences acquises lors de notre cursus et d'en acquérir de nouvelles.

Nous avons donc choisi le sujet concernant le boitier de commande pour cinéma interactif. Notre tuteur technique de l'INSA est Mr Florin Hutu, spécialiste en télécommunications du laboratoire CITI. Nous avons également travaillé le cahier des charges avec les instigateurs du projet, à savoir Tiffany Vernet et Swann Meralli.

En effet, ce projet rentre dans le cadre d'une réalisation plus globale, une projection cinématographique un petit peu inhabituelle que nous allons décrire maintenant.

Présentation du projet

Le projet technique en lui-même est commandité par l'association lyonnaise Entre les Mailles dans le cadre d'un projet de long métrage transmedia, interactif et collectif intitulé "Une histoire simple". Ce projet vaste de par ses ambitions et sa complexité, nécessite la création de solutions techniques particulières. C'est ici que nous rentrons en jeu.

Le film, qui sera projeté tout d'abord en plein air, comporte une composante interactive forte. En effet, à certains moments, le film s'arrêtera et des choix déterminant la suite de l'histoire devront être pris par le public, à l'aide d'un dispositif de commande. Des séquences de jeux vidéo seront également insérées dans le long métrage et seront également contrôlées par ce dispositif. Concrètement, des tables seront installées en face de l'écran avec un boîtier de commande à disposition sur chacune d'entre elles (un boîtier pour 4 5 personnes, pour favoriser les échanges).

Notre travail consiste en la réalisation de ces boitiers et du système de commande associé comme nous allons le décrire maintenant.

Le boîtier sera un module comportant 5 boutons (4 directions, 1 bouton d'action) et devra être sans fil, tant en alimentation qu'en communication. Cela implique un dispositif fonctionnant avec pile ou batterie et communiquant en RF. Un dispositif lumineux devra indiquer si le boîtier est actif. Tous les boîtiers pourront être actifs en même temps (cas d'un vote pour choisir la suite du film) ou bien un seul à la fois (cas d'une séquence vidéo ludique). Un ordinateur central en régie récupérera les données des boîtiers et pourra envoyer des instructions grâce une Gateway (passerelle entre le réseau de boîtiers et l'ordinateur). Nous nous intéresserons ici qu'à la partie boîtier de commande et Gateway dans sa problématique réseau et commande. Nous tâcherons de faire en sorte que les données soient mises en forme de manière simple afin de faciliter un futur développement logiciel. 20 boîtiers devront pouvoir être actifs en même temps.

Le cadre technique du projet est ici encadré en rouge sur le schéma de principe de l'application.

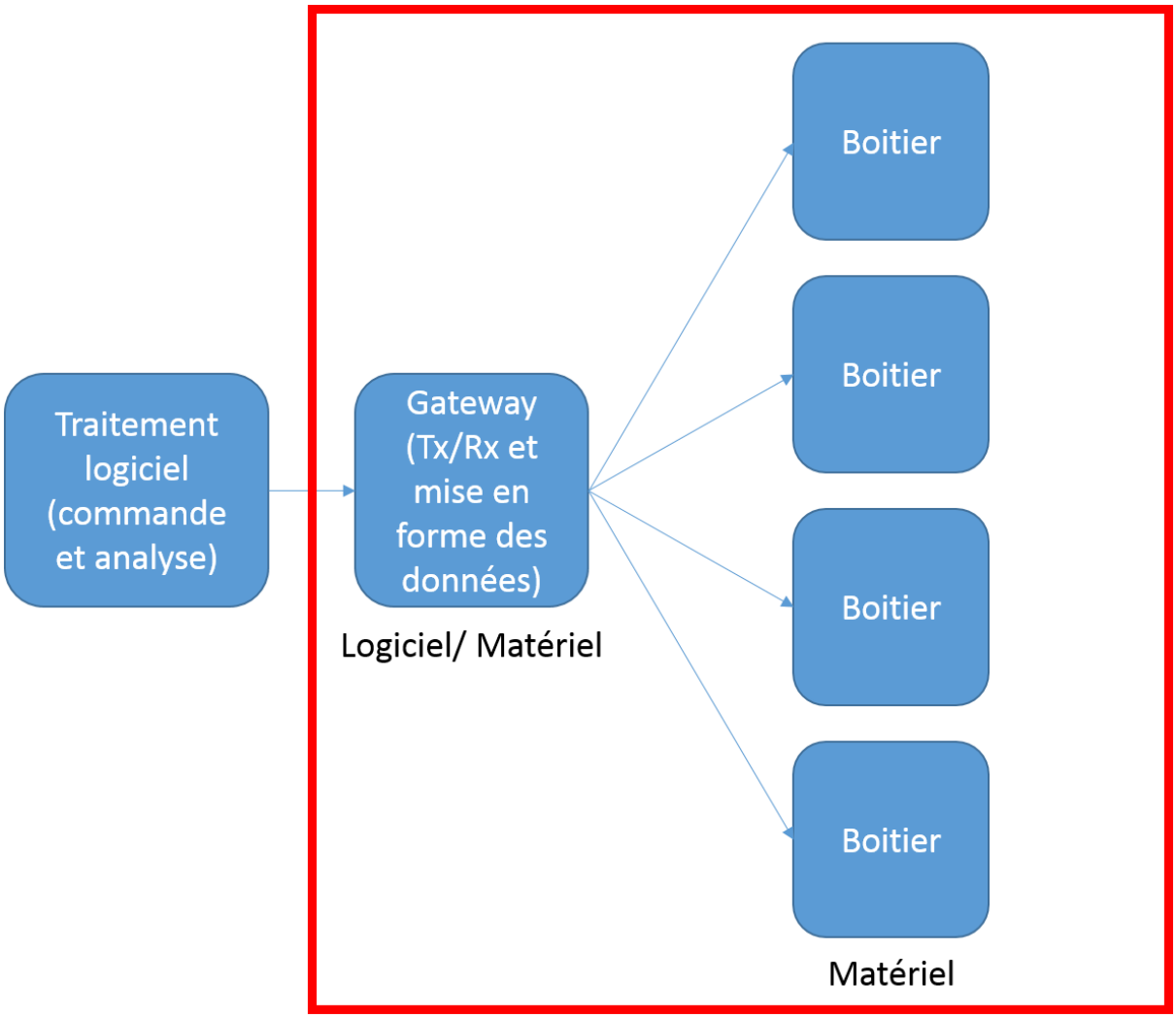


Figure 1: Cadre technique du projet

Le boîtier doit avoir une prise en main simple et rapide et ne doit pas susciter « l'envie » de l'emporter avec soi : la projection étant publique et en plein air, il ne sera pas possible de prévenir un quelconque vol.

Dans un premier temps, il a fallu évaluer quelles solutions étaient les plus adaptés afin de répondre au cahier des charges. Plusieurs choix s'offrent à nous concernant plusieurs mises en œuvre :

- Choix du protocole radio.
- Choix de la solution d'alimentation.
- Choix du de la solution de commande.
- Choix du traitement logiciel.

Concernant le protocole radio, nous avons choisi le protocole Zigbee : celui-ci permet d'avoir une portée suffisante et une consommation réduite, et ce pour une somme réduite. La mise en œuvre est également plus simple qu'un réseau bluetooth. Nous utiliserons plus précisément les modules Xbee S2 basé sur ce protocole. Leur mise en œuvre est simple et leur consommation faible.

Figure 2: un module Xbee S2

Nous avons choisi une alimentation par pile pour les télécommandes, l'alimentation par batterie étant inadaptée à l'utilisation très parcellaire des boîtiers et au nombre de ceux-ci. La mise en œuvre est également plus simple pour l'utilisateur. Nous avons choisi la pile 9V, qui, bien qu'elle ne soit pas à la tension de travail des composants, possède une capacité suffisante pour l'application.

Figure 3 : pile 9V utilisée dans le boitier

La commande se fera à travers un microcontrôleur gérant les entrées et les sorties de la télécommande. La « gateway » branchée à l'ordinateur se chargera d'envoyer des données que le microcontrôleur pourra décoder et inversement. Nous avons choisi la solution arduino, là aussi simple de mise en œuvre, pour piloter nos entrées-sorties et gérer les communications côté télécommande. Afin d'être dans notre cahier des charges en terme de prix et de dimensionner correctement notre système, nous avons choisi la carte Arduino mini pro 3.3V, qui possède juste ce qu'il nous faut en terme d'entrées-sorties ou de capacité de calcul et de mémoire.

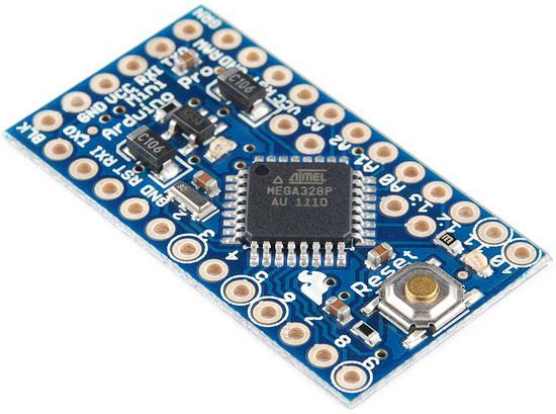


Figure 4: Arduino mini pro distribuée par Sparkfun

Le traitement logiciel fera l'objet d'un chapitre à part et gérera l'interfaçage entre la partie réseau du système et la création de données exploitables pour le commanditaire du projet.

Fonctionnement général

Connaissant ces choix techniques, il peut être intéressant de redéfinir le fonctionnement général de notre système.

Du côté passerelle, nous aurons une carte **Xbee coordinatrice** reliée grâce à un adaptateur USB à un ordinateur. Celui-ci, à travers la solution logicielle, gère la connexion avec les cartes **Xbee** des télécommandes, l'envoie et la réception des données, et la transcription en événements MIDI.

Côté boitier, des LEDs seront pilotés par la carte **Arduino**, à partir des données reçues par la carte **Xbee terminal,** et de même un appui sur un bouton sera traité par l'Arduino et envoyé par la **Xbee** vers la carte **coordinatrice.** Le système peut supporter 20 télécommandes différentes. Le principe de base peut donc se schématiser comme ceci :

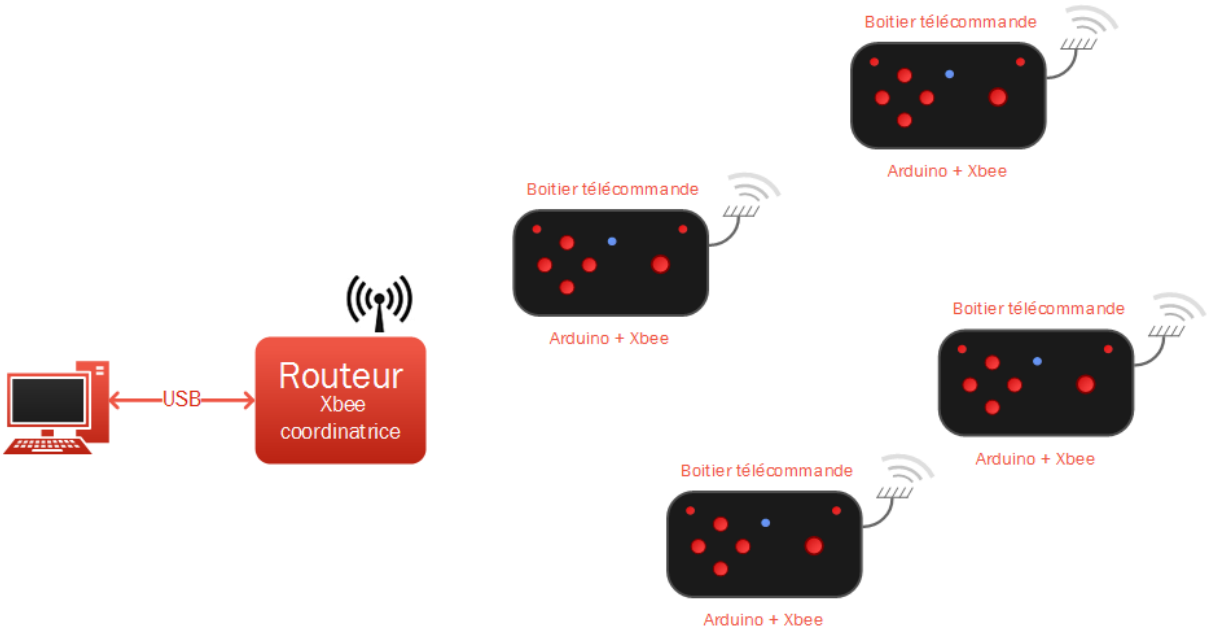


Figure 5 : Une carte coordinatrice gère l'ensemble des boitiers.

Nous allons maintenant présenter notre réalisation, en séparant la partie matérielle (hardware) du système et le développement logiciel (software).

Détails techniques

Ce chapitre présente les principaux aspects techniques utilisés lors de la réalisation de notre projet. C'est un zoom sur certains protocoles ou principes impliqués dans la réalisation.

Réseau Zigbee et carte Xbee

Aspect réseau

Notre carte **Xbee** peut être utilisée dans une configuration réseau de type **Zigbee** basé sur le protocole standardisé IEEE 802.15.4, ce qui veut dire que celui-ci aura les caractéristiques théoriques suivantes :

- Transfert de données maximum à 250 kbps.
- Fréquence de fonctionnement à 2.4 GHz.
- Portée en intérieur de 30m. C'est également la portée théorique en milieu extérieur dit urbain. En effet les bâtiments ou les obstacles divers perturbent la transmission de par les absorptions et réflexions de signaux qu'ils induisent.
- Portée en extérieur (espace dégagé) de 100m.
- Puissance en transmission : 1 mW (0 dBm).
- Adressage avec PAN ID et 64-bit IEEE MAC : le PAN ID (pour Personal Area Network IDenfier) est un numéro d'identification du réseau. Chaque réseau est défini par un unique PAN ID, donc cet identifiant est partagé par tous les membres du même réseau. Ainsi, dans notre cas, toutes les Xbee auront le même PAN ID. Le MAC est une adresse unique à chaque télécommande, non modifiable.
- 1 canal : le réseau ZigBee opérera sur 1 seul canal de communication autour de 2.4 GHz grâce à une modulation à étalement de spectre DSSS

La topologie du réseau ZigBee sera dite « en étoile », c'est-à-dire qu'une seule Xbee dite « coordinator » sera relié à chaque Xbee de boîtier (appelées « end device ») comme ceci :

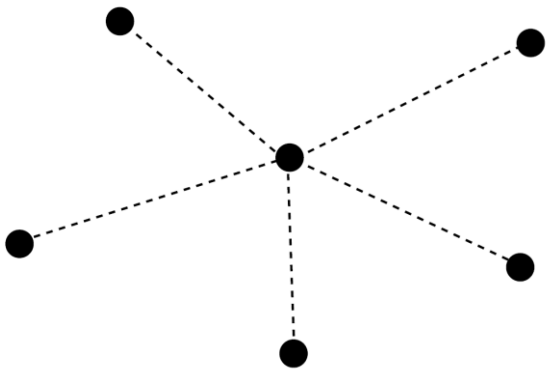


Figure 6: réseau en étoile : le « coordinator » est au centre

On devine aisément que notre Xbee coordinatrice reliée à l'ordinateur jouera le rôle de « coordinator » dans l'implémentation. Le réseau peut également supporter la présence de routeurs, qui sont des nœuds intermédiaires reliés d'un côté au coordinateur et de l'autre à de multiples « end devices ». Ces éléments ne seront toutefois pas utilisés dans notre implémentation.

Communication

Autre point important, la communication de la carte Xbee avec les autres éléments, que sont l'ordinateur et la carte Arduino. Celle-ci se fait grâce à une interface de communication série UART CMOS en 3.3V. C'est pour cela d'ailleurs que nous avons choisi un modèle d'Arduino mini pro communiquant lui aussi en UART 3.3V, assurant la compatibilité. Pour ce qui est de la communication avec l'ordinateur, nous avons donc besoin d'un adaptateur UART/USB pour fonctionner, comme celui-ci :

Figure 7: adaptateur Xbee/USB microbot

Cette communication série permet de transférer des données vers ou de la Xbee pour transmettre ou recevoir sans-fil, ainsi que configurer la carte.

Configuration

Le principal avantage de ces cartes Xbee est l'utilisation d'un logiciel appelé XCTU, permettant de configurer et tester facilement les cartes Xbee.

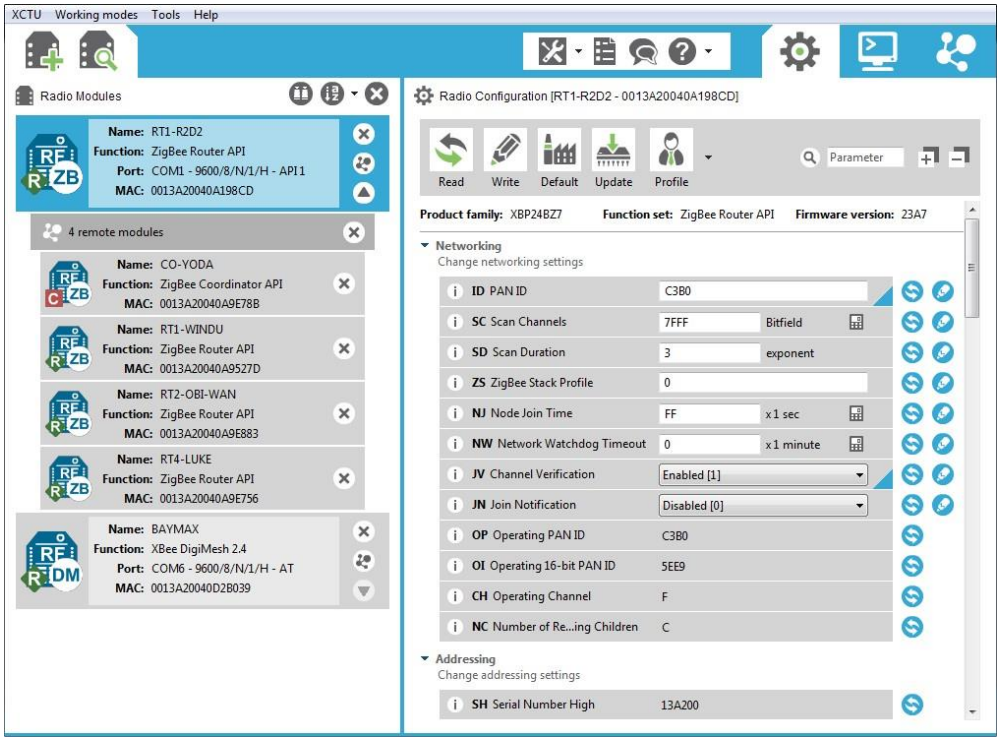


Figure 8 : XCTU en action

D'ici, il est possible de configurer de nombreux paramètres de la carte Xbee, du comportement en veille au PAN ID. Il est également possible d'installer le « firmware » (le logiciel s'exécutant sur la carte) adapté à la fonction de celle-ci.

Dans notre implémentation, il faut donc charger le firmware appelé « Zigbee End Device AT » à toutes les cartes Xbee s'intégrant dans les boitiers et le firmware « Zigbee Coordinator API » à la Xbee coordinatrice. De plus, il faut configurer le même PAN ID à toutes les cartes du système, afin de spécifier que celles-ci sont sur le même réseau.

Nous pouvons également détailler l'utilisation des pins de la Xbee, c'est-à-dire les entrées sorties de celle-ci. Dans le cas de la Xbee coordinatrice, rien à faire, il suffit de la brancher sur l'adaptateur USB. Dans le cas de la carte utilisée pour la télécommande, les pins utilisés sont à relier à l'Arduino. Nous verrons plus tard les détails de l'utilisation de ces pins, mais nous pouvons déjà établir le tableau résumant les fonctions de ceux-ci. On pourra se référer à l'annexe A2 pour connaitre la position de ces pins sur la carte.

PIN	**FONCTION**
VCC	Alimentation
GND	Masse
DOUT	Réception provenant de la Xbee
Din	Transmission vers la Xbee
ON/SLEEP	Indicateur statut carte

Arduino mini pro

La carte Arduino mini pro présente dans le boîtier télécommande gère les entrées/sorties du système. C'est l'unité nécessaire au contrôle de l'application. Elle est programmable à travers l'utilisation de plusieurs éléments :

- Le programme de programmation Arduino, qui permet de créer du code en C simplifié régissant le comportement du microcontrôleur.
- Un câble FTDI, qui permet de relier un ordinateur à la carte pour transférer le code vers celle-ci.

Figure 10 : le programme Arduino

Figure 9 : câble FTDI

Concrètement, il est nécessaire de souder des « pins headers » sur la carte arduino afin de pouvoir avoir accès aux entrées et sorties de la carte. Les pins situés sur la largeur de la carte sont

destinés à la communication avec l'ordinateur à travers le câble FTDI, comme on peut le voir figure 11. Ainsi, le code généré par le logiciel peut être transféré vers la carte.

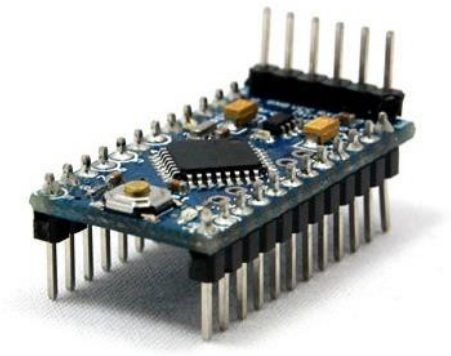


Figure 11 : Arduino mini pro munie de ses headers

Ensuite, les pins restants sont destinés aux entrées sorties du système en lui-même : tous ne peuvent être utilisés de la même façon. Nous verrons plus tard les détails de l'utilisation de ces pins, mais nous pouvons déjà établir le tableau résumant les fonctions de ceux-ci. A savoir seulement ici que chaque pin peut être configuré comme une entrée ou une sortie. On pourra se référer à l'annexe A2 pour connaitre la position de ces pins sur la carte.

PIN	**FONCTION**
GND	Masse
VCC	Alimentation
9	Input : PIN ON/SLEEP Xbee
8	Input : bouton gauche
7	Input : bouton haut
6	Input : bouton droit
5	Input : bouton bas
4	Input : bouton action
10	Output : Led 1 (PWM)
11	Output : Led 2 (PWM)
3	Output : Led 3 (PWM)
RX1	Réception provenant de la Xbee
TX0	Transmission vers la Xbee

Alimentation

Nos composants sont alimentés en 3.3V, or nous alimentons l'ensemble du système en 9V grâce à la pile. Il est donc nécessaire d'abaisser la tension grâce à un régulateur. Pour que celui-ci soit efficace en termes de consommation électrique, nous avons choisi un régulateur de type hacheur buck. Dans la pratique, nous n'avons pas pu tester ce régulateur, celui-ci étant monté sur un boitier impossible à utiliser en prototypage.

Schéma électrique

Notre sous-système boitier s'articule autour d'une implémentation électrique simple que nous pouvons décrire par ce schéma :

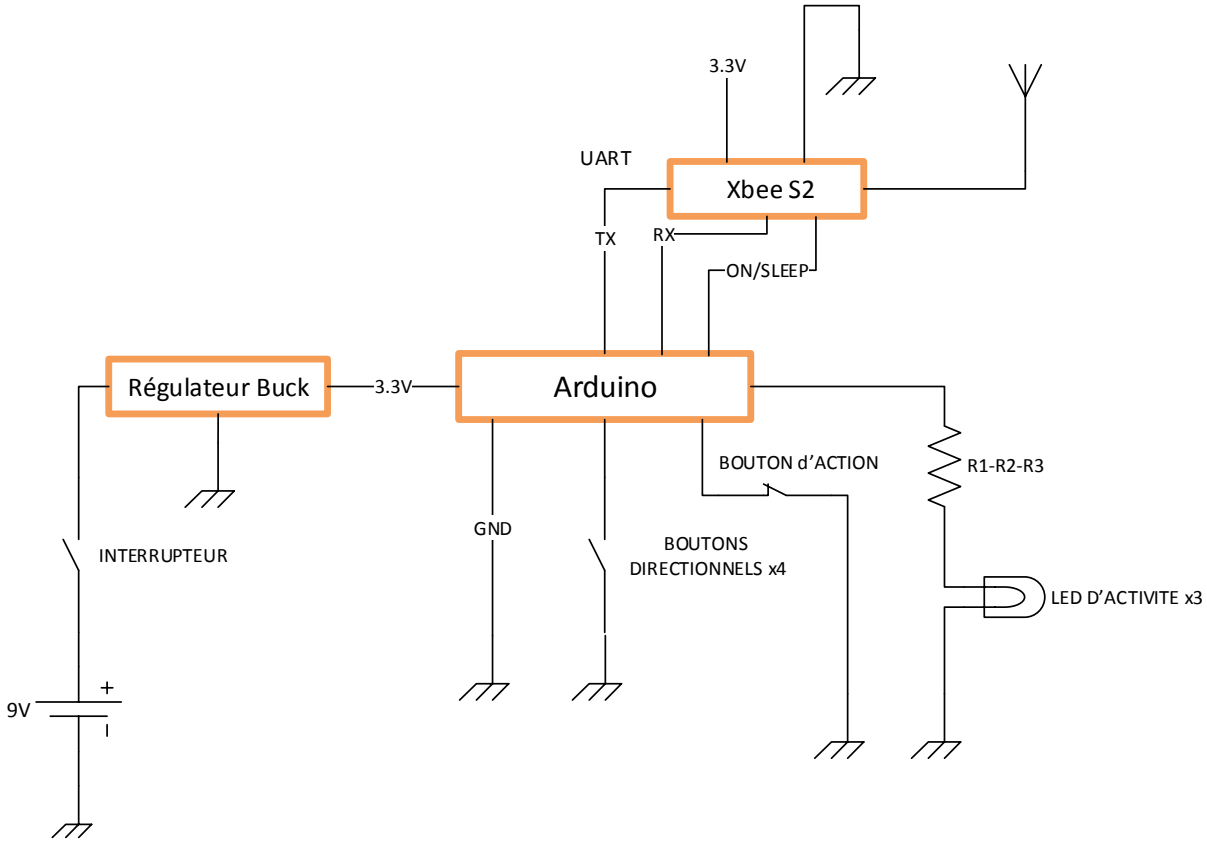


Figure 12 : schéma électrique de la télécommande

La pile 9V, grâce à un adaptateur permettant de transformer les bornes de celle-ci en fils, délivre une tension au régulateur Buck qui transforme celle-ci en 3.3V permettant d'alimenter l'Arduino et la Xbee S2. Ces deux composants communiquent entre eux grâce à la liaison UART série Rx/Tx, qui permet l'envoie et la réception de données à travers la carte radio.

La sortie ON/SLEEP de la Xbee est utilisée en entrée de l'Arduino pour connaitre l'état de la carte radio. Si celle-ci est prête à transmettre, la liaison est à l'état logique 1, sinon elle est à 0. Ceci sert à contrôler la bonne transmission des données, la carte se mettant en veille 3 secondes après la dernière réception de données.

L'Arduino a pour entrées les pins reliés aux différents boutons. Les boutons sont des interrupteurs naturellement ouverts, à l'exception ici du bouton d'action, qui dans notre prototype est bouton naturellement fermé. Les entrées sont maintenues à l'état logique 1 grâce à des résistances internes dites de « pull-up ». A l'appui du bouton (ou à la relâche pour le bouton d'action), les entrées sont reliées à la masse donc à l'état logique 0. Nous avons ainsi une information à traiter quant à l'état des boutons.

Les sorties de notre carte sont les liaisons pilotant des LEDs d'activité du boitier. Elles sont au nombre de 3 et chacune est en série avec une résistance afin de limiter le courant la traversant. Elles peuvent être contrôlées indépendamment : ainsi l'une peut servir à indiquer l'état de marche du boitier, les autres les moments où le boitier est actif pour un vote ou une séquence de jeu. A noter que

les LEDs sont branchées sur des sorties en Pulse Width Modulation (PWM) de l'Arduino, ce qui permet de faire varier leur intensité et donc de créer des effets.

Voici comment donc s'articule le schéma électrique de notre télécommande, regardons maintenant l'agencement du code de l'Arduino.

Application embarquée

Structure du code

Comme décrit précédemment, nous pouvons programmer l'Arduino (plus précisément le microcontrôleur ATmega 328), dans la limite de la mémoire disponible sur la carte. Ce code sert la logique de contrôle et de commande de la télécommande. Sans rentrer dans le détail du code, nous exposerons ici la logique de cette implémentation à travers des organigrammes. A noter que nous ne nous intéresserons ici qu'au mode de fonctionnement normal, et que ce code fonctionne en synergie avec l'implémentation logicielle que nous verrons plus tard.

Le fonctionnement général peut être visualisé grâce à cet organigramme :

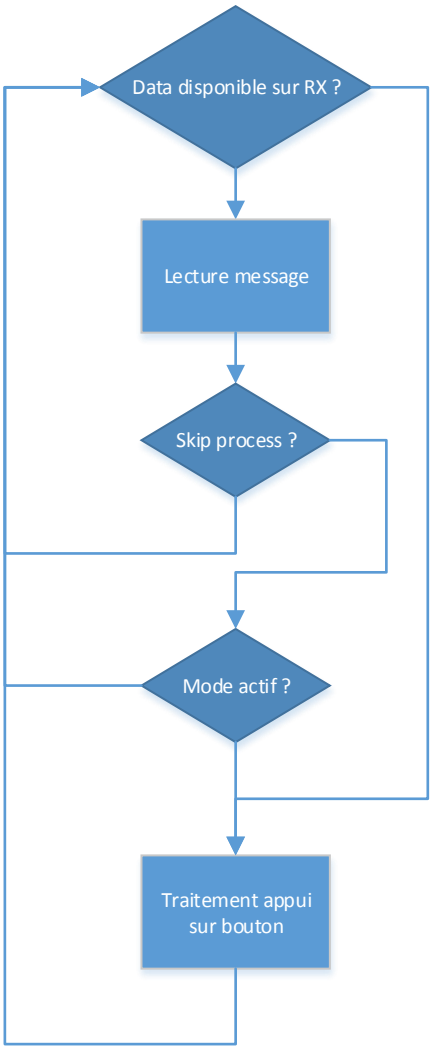


Figure 13 : Mode normal de fonctionnement

Le fonctionnement consiste en une boucle qui se répète tant que le microcontrôleur est sous tension. Après une phase d'initialisation, le programme rentre dans cette boucle et n'en ressort jamais.

Le programme commence à vérifier si des données en provenance du coordinateur sont disponibles (ces données sont stockées temporairement dans la carte Xbee). Il lit le message et en

exécute le contenu. Puis en fonction du type de message, il décide de sauter ou non le traitement de l'appui des boutons et autre et de recommencer la boucle (le symbole losange indique un choix).

Le traitement des messages se fait de la manière suivante :

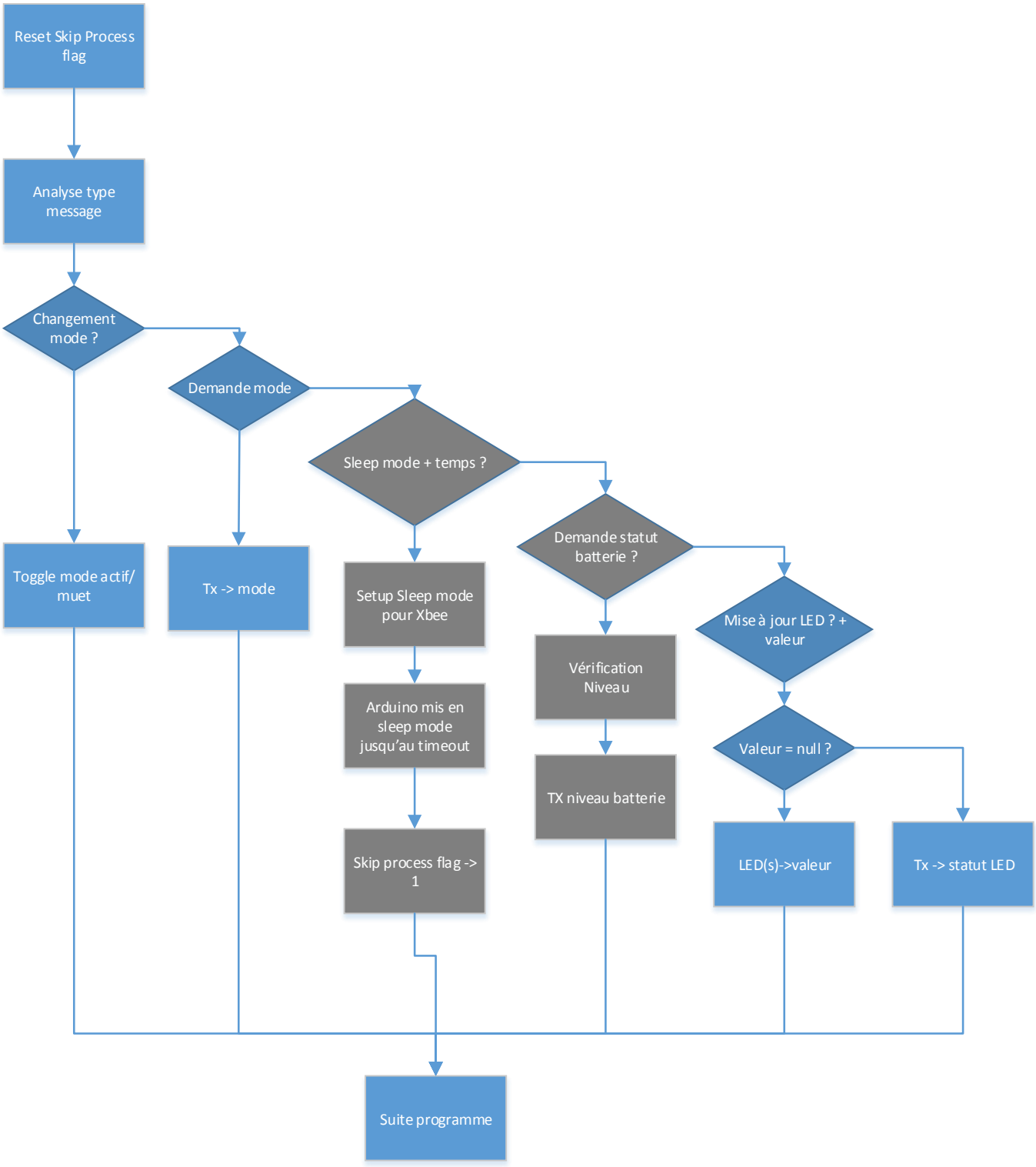


Figure 14 : Traitement des messages

Une suite de « If statements » permet de voir de quel type de message il s'agit. Nous pouvons donc changer de mode de la télécommande : actif ou muet. Nous pouvons demander dans quel mode nous sommes, l'information sur ce mode est transmise à la Xbee pour envoi. Nous pouvons mettre à jour les LED avec une valeur particulière, c'est-à-dire choisir son intensité. S'il n'y a pas de valeur transmise, le programme transmet l'état des LED. A savoir que ce type de message est implémenté pour chaque LED séparément ou pour toutes les LED en même temps.

Les types de message en grisés sont implémentés partiellement, c'est-à-dire que les fonctions associées ne sont pas opérationnelles mais la structure des messages l'est. Nous en reparlerons plus tard.

Le programme passe ensuite au traitement de l'appui des boutons, seulement si le mode défini pour la télécommande est actif.

La gestion de l'appui de bouton se fait comme ceci :

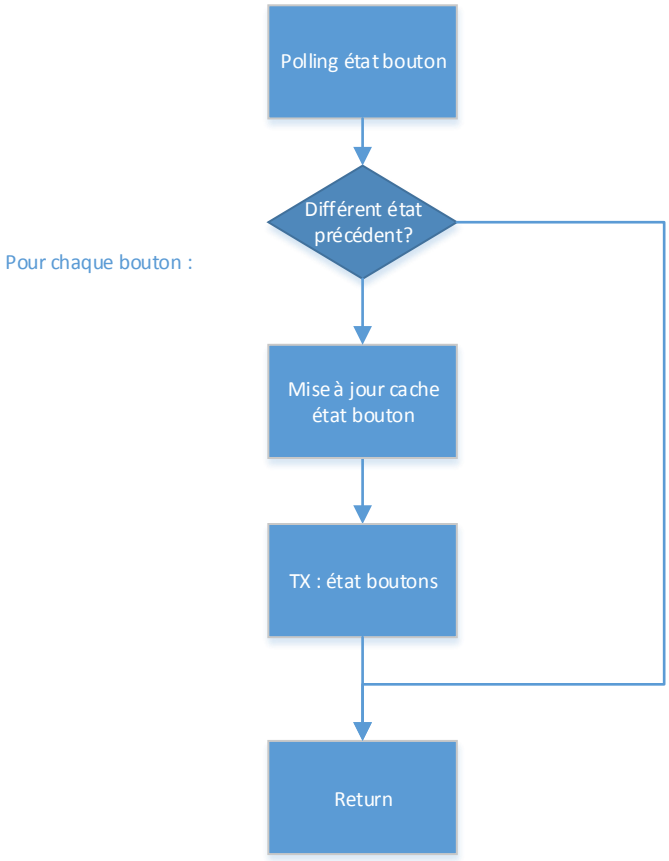


Figure 15 : Le traitement de l'appui de bouton

Tout simplement, un tableau stocke l'état des boutons du passage précédent ; on compare ce tableau à l'état actuel des boutons, et s'il y a différence, on met à jour ce tableau et on transmet l'état à la Xbee pour envoi.

Le fonctionnement de l'envoi est simple. A chaque instruction Tx dans le programme, on stocke le message dans un « buffer », puis on l'envoie à la fin de la boucle principale si la carte Xbee est réveillée. On peut ainsi stocker plusieurs messages prêts à l'envoi et ne pas perdre d'information.

Nomenclature des messages

Ici, nous détaillerons la nomenclature des messages envoyés et reçus par la télécommande lors du fonctionnement normal. Ils sont composés de plusieurs octets de donnés : nous les détaillerons ici en valeurs hexadécimales (un octet = 8 bits ; 255 valeurs de 0x00 à 0xFF en hexadécimal). Le premier octet défini la nature du message, et l'octet 0xFF est utilisé comme octet annonçant la fin du message, il est donc à la fin de chaque message et ne sera pas détaillé ici. Là encore, les messages non implémentés sont grisés.

Messages reçus par la Xbee (commande)

Type message	***1er octet***	***2ième octet***	***3ième octet***
Sleep command	*0xC8*	*<Time>*	*<Time>*
ACTIVE_MODE_CMD	*0x96*	*-*	*-*
MUTE_MODE_CMD	*0x64*	*-*	*-*
ASK_MODE_CMD	*0x50*	*-*	*-*
BATT_STATUS_CMD	*0x10*	*-*	*-*
LED_CMD	*0x20*	*Null ou 0x00 à 0xFE (intensité lumineuse)*	
LED1_CMD	*0x21*	*Null ou 0x00 à 0xFE (intensité lumineuse)*	
LED2_CMD	*0x22*	*Null ou 0x00 à 0xFE (intensité lumineuse)*	
LED3_CMD	*0x23*	*Null ou 0x00 à 0xFE (intensité lumineuse)*	

Messages émis par la Xbee

Type message	***1er octet***	***2ième octet***	***3ième octet***	***4ième octet***
BUTTON_LEFT_TX	*0xB0*	*0x00 ou 0x01*		
BUTTON_UP_TX	*0xB1*	*0x00 ou 0x01*		
BUTTON_RIGHT_TX	*0xB2*	*0x00 ou 0x01*		
BUTTON_DOWN_TX	*0xB3*	*0x00 ou 0x01*		
BUTTON_ACTION_TX	*0xB4*	*0x00 ou 0x01*		
BATT_STATUS_TX	*0x10*	*0x00 à 0xFE*		
LED_CMD	*0x20*	*0x00 à 0xFE (LED1)*	*0x00 à 0xFE (LED2)*	*0x00 à 0xFE (LED3)*
LED1_CMD	*0x21*	*0x00 à 0xFE*		
LED2_CMD	*0x22*	*0x00 à 0xFE*		
LED3_CMD	*0x23*	*0x00 à 0xFE*		
ASK_MODE_CMD	*0x50*	*0x00 (mute) ou 0x01 (active)*		

L'intégralité du code de l'Arduino est disponible en annexe A3.

Réalisation

Le prototype de télécommande a été réalisé sur platine de prototypage pour la partie électrique. Nous avons cependant construit un boitier assez grand pour contenir celle-ci et créer un appareil complet et fonctionnel. Il ressemble à ceci :

Figure 16 : le prototype de télécommande

Il est temps de passer à la description de la composante logicielle de notre projet intitulée VoxPopuli.

Software : VoxPopuli, l'interface avec la Régie

Les défis proposés par ce projet ne sont pas purement matériels. La gestion du réseau de télécommande nécessite une bonne connaissance du fonctionnement d'un réseau XBee, du protocole de communication utilisé par le calculateur à l'intérieur de chaque télécommande. Bref la gestion bas-niveau du réseau de télécommande ne peut pas être déléguée aux concepteurs de "l'application régie" et doit être une composante importante de la solution que nous proposons.

Le rôle des applications VoxPopuli et de prendre en charge le réseau de télécommande depuis l'ordinateur régie et de proposer des outils de communication fiable et ergonomique pour contrôler ce réseau.

Enjeux du logiciel d'interfaçage

Fiabilité

Le premier objectif du logiciel d'interfaçage est sa fiabilité. Bien sûr cela suppose qu'il doit planter le moins possible, mais les crashs sont toujours difficiles à prévoir (surtout avec un temps de "test" aussi court). En cas de crash il faut donc que le logiciel puisse restaurer l'état dans lequel il se trouvait avant le crash, ainsi le réseau sera en "roue libre" pendant quelques secondes mais retrouvera rapidement son comportement asservie.

Multiplicité de l'interfaçage

L'enjeu certainement le plus important du logiciel est de pouvoir communiquer avec une ou plusieurs applications desquelles nous ignorons tout, au moment de concevoir l'interface. Le réseau doit pouvoir être accessible depuis les "applications régies" quel que soit le langage dans lequel elles ont été codées. De plus, l'ordinateur de régie doit pouvoir fonctionner sous linux et sous Windows (le logiciel doit pouvoir être exécuté sur ces deux plateformes).

Ergonomie

Pour faciliter sa prise en main, le logiciel d'interface doit permettre de contrôler un réseau à la topologie simple : l'application régie n'a en fait besoin de voir que des télécommandes ou des groupes de télécommandes possédant 5 boutons et 3 LEDs. Toutes informations internes à la gestion du réseau (niveau de batterie ou de signal, état de la télécommande, connexion et déconnexion...) ne doivent pas perturber la clarté de ce réseau simplifié.

Enfin le logiciel doit apporter une solution adapté à l'environnement particulier que constitue le milieu du spectacle : il doit permettre la préparation du spectacle puis sont implantations dans des salles ou des environnements différents.

Design et philosophie de VoxPopuli

C'est dans l'optique de satisfaire aux enjeux énoncés plus haut que VoxPopuli a été conçu. Nous aborderons ici les solutions proposées pour répondre à chaque point du cahier des charges.

Fiabilité

Pour améliorer la fiabilité du logiciel d'interfaçage, les fonctions proposées par VoxPopuli ont été séparées dans deux logiciels distincts. Le cœur de l'interface : VoxPopuli Deamon, ne contient que les fonctionnalités de bases. Il est capable de gérer le réseau (connexion, déconnexion), de communiquer avec les applications de régie et de configurer leurs interactions avec les télécommandes. Cependant, il ne possède aucune interface graphique pour limiter les risques de crashs. Ainsi, c'est le deuxième logiciel VoxPopuli GUI qui est chargé de communiquer avec VoxPopuli Deamon et de fournir une interface graphique capable d'afficher, de modifier et de paramétrer le réseau.

En représentation seul VoxPopuli Deamon est réellement utile (puisque toutes les configurations ont normalement été réglées au préalable), c'est donc ce dernier qui dispose d'un mécanisme de récupération de plantage. En pratique, toutes les données manipulées par le logiciel durant son exécution sont mises en cache sur le disque (leur syntaxe sera exécutée plus loin dans le rapport). Lorsque le logiciel quitte normalement il vide ce cache, mais lorsqu'il plante ces fichiers restent sur le disque et permettent de restaurer l'état du Deamon lorsqu'on le relance ensuite. Ces fichiers permettent aussi une sauvegarde de la configuration du réseau, en vue de la réutiliser d'un spectacle à l'autre sans avoir besoin de tout reconfigurer.

Multiplicité de l'interfaçage

Pour répondre au principal défi du cahier des charges, VoxPopuli apporte plusieurs solutions pour interfacer une application avec le réseau de télécommande.

Evènements clavier

La plus simple est certainement d'utiliser les entrées clavier. Il est en effet possible de demander à VoxPopuli Deamon de générer un "faux" évènement clavier (appuie ou relâchement d'une touche) lors de l'appuie ou de relâchement d'un bouton d'une des télécommandes du réseau. Cette méthode permet un interfaçage simple avec les applications supportant les raccourcis clavier (on peut ainsi contrôler le défilement d'une présentation power point, jouer aux jeux ne nécessitant pas de souris, contrôler un lecteur vidéo...). Cependant, bien que cet interfaçage soit simple à mettre en place, il est à déprécier. En effet, la distribution de l'évènement clavier est déléguée au système d'exploitation qui le transmettra à l'application ayant le focus (en général la fenêtre actuellement sélectionnée). Il est donc impossible de communiquer avec plusieurs applications en même temps et celles-ci n'ont aucun moyen d'action sur le réseau (impossible pour elle d'allumer un LED par exemple).

Evènements MIDI et OSC

Pour établir une communication plus complète entre le réseau et les applications de régie, il faut donc utiliser un vrai protocole de communication entre applications. Ils existent de nombreux protocoles différents pour réaliser cette tâche. Nous avons donc sélectionnée ceux les plus utilisés dans le milieu du spectacle. VoxPopuli peut ainsi générer et recevoir des évènements MIDI (Musical Instrument Digital Interface) et sa version améliorée OSC (Open Sound Control). Tout comme les évènements clavier, on peut configurer le logiciel pour qu'il fasse correspondre un évènement MIDI ou OSC à un changement d'état d'un bouton d'une télécommande, mais il est aussi possible de contrôler la puissance de LED via ces protocoles. Et contrairement à l'interfaçage clavier, il est possible de paramétrer plusieurs ports MIDI ou OSC, pour chacune des applications régies.

CLI via telnet

Enfin une dernière solution de communication a été mise en place. En effet, malgré l'efficacité du MIDI et de l'OSC pour contrôler le réseau pendant la représentation, ces protocoles se montrent parfaitement inadaptés pour configurer VoxPopuli Deamon. Pour combler cette lacune, VoxPopuli Deamon dispose d'une Interface en Ligne de Commande (CLI) disponible via telnet. Cette interface en ligne de commande permet d'accéder à la totalité des fonctionnalités du logiciel d'interfaçage que ce soit pour configurer et surveiller le réseau de télécommande ou paramétrer les interactions avec les différentes applications régies. Le fonctionnement de cette CLI sera détaillé plus loin dans ce rapport. C'est l'interface à privilégier pour le paramétrage du réseau (c'est celle utilisée par VoxPopuli GUI pour communiquer avec VoxPopuli Deamon), et elle peut aussi être utilisée pour relayer les évènements provenant ou à destination des télécommandes (même si elle possède une latence légèrement supérieure par rapport au MIDI).

Fichiers de cache

Une dernière solution existe pour configurer VoxPopuli Deamon : modifier directement les fichiers mis en cache sur le disque qui décrivent la totalité des variables manipuler par le logiciel. Cette méthode est cependant déconseillée à moins d'être sûr de la syntaxe de ces fichiers.

En bref

Le schéma suivant présente tous les interfaçages possibles du réseau :

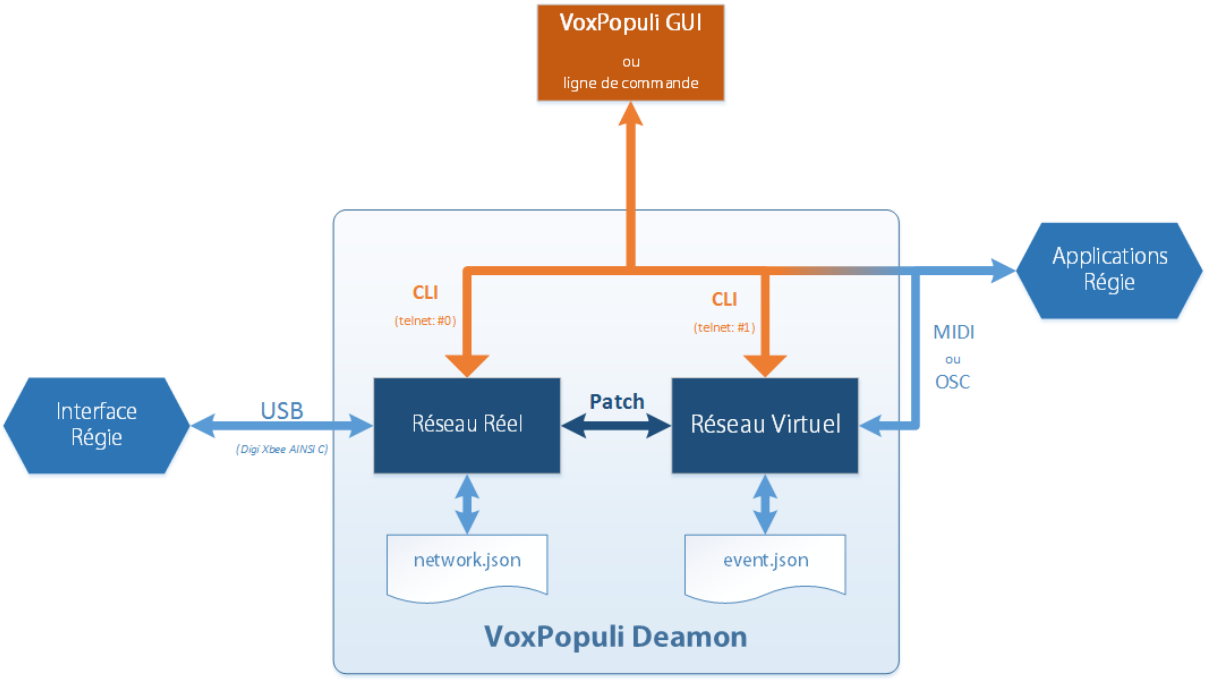


Figure 17 : Interfaçages possibles du réseau

Ergonomie et simplicité d'utilisation

En vue d'être utilisé dans le contexte d'un spectacle, VoxPopuli Deamon propose une architecture qui permet à la fois de configurer et de tester le réseau de télécommande sans avoir besoin d'installer toutes les télécommandes, mais aussi de pouvoir adapter une installation de télécommande propre à une représentation à la configuration réalisée.

Pour ce faire, le réseau "physique" (ou réel) de télécommande est découplé du réseau "virtuel" qui communique avec les Applications Régies. Lors de la conception du spectacle, on définit des télécommandes virtuelles et on configure les évenements qu'elles enverront ou pourront recevoir durant la présentation. Ces interactions peuvent alors être simulées sans avoir recours à une vraie télécommande. Au moment de l'installation des télécommandes dans la salle de spectacle chacune d'entre elle devra être connectée à une télécommande virtuelle et toutes les interactions de la deuxième seront alors répercutées sur la première. Ces connexions sont réalisées au moyen d'un patch interne à VoxPopuli Deamon.

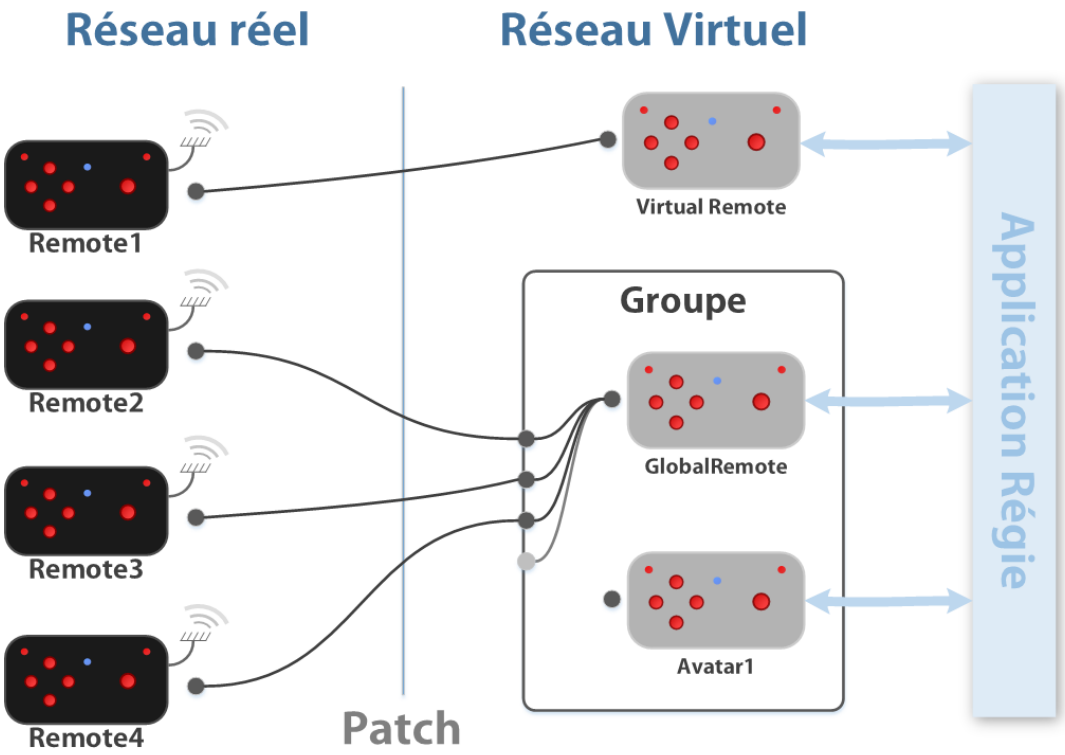


Figure 18 : Réseaux réel et virtuel peuvent être patchés

À ce mécanisme de patch s'ajoute la possibilité de définir Groupe dans le réseau virtuel. Un groupe virtuel permet de gérer un ensemble de télécommande sans connaître leur nombre à l'avance. Vu du réseau réel, un groupe peut être connecté à plusieurs télécommandes réelles, vu de l'application régie il permet de modifier l'état de toutes les télécommandes simultanément via la télécommande virtuelle GlobalRemote, ou bien d'une en particulier via des télécommandes "avatars". En effet, les télécommandes virtuelles avatars peuvent être dynamiquement (et même aléatoirement) connectées à une des télécommandes réelles du groupe, mais offre une interface constante aux applications de régies.

Quelles opèrent en solitaire ou au service d'un groupe les télécommandes virtuelles ont ainsi toujours les mêmes contrôles évènementiels :

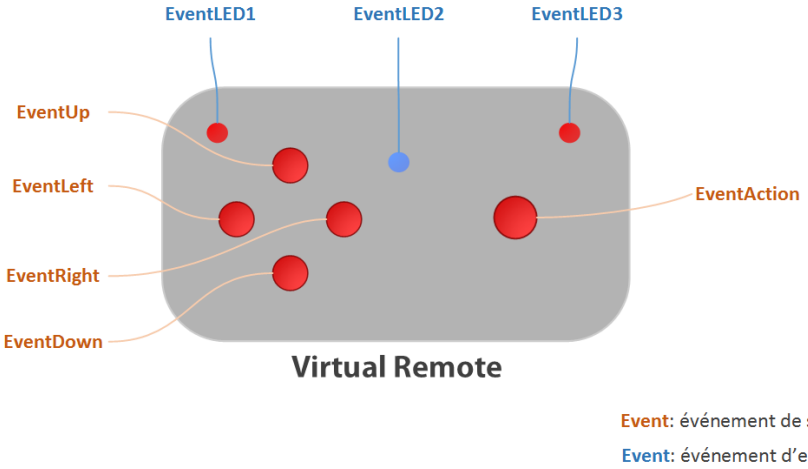


Figure 19: Contrôles événementiels

Dans le cas d'une "histoire simple", les télécommandes de chaque écran ont tout intérêt à être placées dans un groupe. Pour les séquences de vote, un contrôle de toutes les télécommandes en simultané peut-être suffisant, puis pour les séquences de jeu il suffit de demander à une télécommande avatar de se connecter à une seule télécommande réelle du groupe. L'interfaçage avec

le jeu est alors simple puisque quel que soit la télécommande réelle choisie, les évènements envoyés par l'avatar seront les mêmes.

Syntaxe de communication

La configuration du réseau consiste donc en une communication avec VoxPopuli Deamon qui passe soit par la modification des fichiers de caches du logiciel, ou par la CLI via telnet. Nous présenterons ici la syntaxe générale choisie pour ces deux communications.

Le format JSon

Le format choisit pour la sauvegarde du modèle dans les fichiers de cache est le JSon. Ce langage descriptif, extension du javascript, permet de décrire une arborescence d'objet (comme l'XML) tout en restant facilement lisible par un humain.

En JSon, un objet est défini par un ensemble de propriétés chacune définie par un nom et une valeur. Cette valeur peut être de type :

— chaine de caractère
— nombre
— booléen
— tableau de valeur (pas forcément toutes du même type)
— objet

Les objets étant acceptés comme propriétés, le JSon permet de définir des arborescences. Au niveau de la syntaxe les propriétés sont définies par leur nom entre guillemets, de deux points puis de leur valeur :

```
"nom": valeur
```

Les tableaux sont définis entre crochets, chacune de leurs valeurs étant séparée par une virgule.

Enfin, un objet commence par une accolade ouvrante et se termine par une accolade fermante, chacune de ses propriétés est séparée d'une virgule :

```
{
    "Propriété1": 1,
    "Propriété2": "valeur",
    "Tableau": ["valeur",1,true]
}
```

Il est important de noter que la totalité des informations manipulée par VoxPopuli Deamon sont stockées dans des modèles suivants cette syntaxe. Pour marquer la séparation entre réseau physique et réseau virtuel et pour pouvoir les enregistrer et recharcher séparément, chacun possède son modèle JSon propre. Ainsi toutes les informations concernant le réseau réel sont-elles stockées dans le fichier "network.json" et celles concernant le réseau virtuel dans "event.json". Les propriétés définies par ces modèles sont détaillées plus loin dans le rapport.

Interface en Ligne de Commande via telnet

La CLI est accessible via telnet sur le port 9000 de l'ordinateur de régie (elle est accessible depuis n'importe quel terminal du réseau si le pare-feu de l'ordinateur régie l'autorise). Les

commandes interprétées par la CLI sont de deux types : les commandes de configuration de la CLI qui commencent toutes par #, et les commandes destinées aux modèles JSon.

Pour modifier les propriétés d'un modèle JSon via telnet, il faut d'abords sélectionner ce modèle grâce à la commande #0 pour network.json ou #1 pour event.json. Une fois le modèle sélectionné, toutes ses propriétés sont accessible à la lecture. Il suffit de taper leur adresse pour en connaitre leur valeur. L'adresse d'une propriété est constituée par la suite de noms des nœuds qu'il faut parcourir pour l'atteindre, les noms étant séparés par des points.

Ainsi pour le modèle json suivant :

```
{
      " Remotes ": {
         " Remote1 ": {
             // ...
             " LED1 ": 0,
             // ...
         }
      },
      " USBPort ": "/ dev / ttyUSB0 "
}
```

On peut récupérer la valeur de la LED de la télécommande 1 par la commande (le > est afficher par la console telnet quand celle-ci écoute) :

```
>Remotes.Remote1.LED1
Remotes.Remote1.LED1: 0
```

De la même manière on peut modifier la valeur d'une propriété du modèle en ajoutant à son adresse deux points puis sa nouvelle valeur. Il faut tout de même noter que toutes les propriétés du modèle ne peuvent pas être modifiées depuis la CLI (par exemple le niveau de batterie d'une télécommande...). Pour allumer la LED de la télécommande 1 au maximum, il faut donc entrer la commande :

```
>Remotes.Remote1.LED1: 254
```

Il est possible de demander à la CLI d'afficher toute modification du modèle JSon et ce, quel que soit la provenance de la modification. Pour ce faire il faut configurer la CLI en mode verbose par la commande : #v (et #-v pour en sortir). Ce mode permet de synchroniser en permanence l'état du modèle manipulé par VoxPopuli Deamon.

Enfin les objets des modèles JSon disposent de méthodes qui peuvent être appelées par la CLI de la même manière que les propriétés. Ainsi tous les objets disposent d'une méthode print() qui affiche leur contenu en format JSon. Certains objets (comme les télécommandes) peuvent être renommé par la méthode rename("nouveauNom").

Ainsi pour renommer Remote1 en Telecommande1 il faut exécuter la commande :

```
>Remotes.Remote1.rename ("Telecommande1")
Remotes.Remote1 renamed "Telecommande1"
```

La réponse renvoyée par la CLI stipulant que l'opération de renommage s'est déroulée correctement ne s'affiche qu'en mode verbose.

Voici pour le fonctionnement VoxPopuli, il est temps de s'intéresser aux coûts nécessaires à la construction du dispositif.

Coûts

Ce chapitre récapitule l'ensemble des composants nécessaire à l'élaboration du système ainsi qu'une estimation du prix de chaque composant. Parfois, le choix est donné concernant le matériel : cela sera précisé dans ce cas.

Boîtier

Nous essaierons ici de donner une estimation du prix par boîtier. Lors de notre conception, nous avions des restrictions en termes de fournisseurs : nous avons donc indiqués ici les composants que nous avons achetés chez ces fournisseurs, mais il est possible de trouver des prix plus intéressants ailleurs. De plus certains composants comme les boutons peuvent être choisis autrement (par esthétique par exemple) ce qui peut amener une réduction du prix. Ces composants seront grisés dans le tableau.

Composant	**Prix (HT)**	**Référence**	**Fournisseur**
Régulateur de tension	<1€	TEXAS INSTRUMENTS TL2575-33IKTTR Régulateur à découpage Buck (Step Down), Fixe, 4.75V-40V In, 3.3V et 1A Out, TO-263-5	Farnell
Pile	1€<5€	Pile 9V	Divers
Adaptateur pile	7.74€ / 10 (paquet de 10)	MULTICOMP 440006P CONNECTEUR BATT PP3 TYP2 150MM PQT 10	Farnell
Résistances, fils	<< 1€		Divers
Interrupteur	<1€	TE CONNECTIVITY SLS121PC04 Commutateur à glissière, Série SLS, SPDT, Traversant, 250 mA, 125 V	Farnell
Arduino Mini Pro*	5.42€ OU 16.32€	Carte Microcontrôleur Pro Mini haut 3.3V qualité 8MHz ATmega328P -AU avec Broches Pour Arduino OU Arduino Pro Mini 328 - 3.3V/8MHz	Amazon
Xbee S2	20.82 €	DIGI INTERNATIONAL XB24-BWIT-004 MODULE ZIGBEE XBEE ZNET 2.5 1MW	Farnell
Boutons directionnels	1.66€ * 4	MULTICOMP R13-502A-05-R Commutateur à bouton-poussoir, SPST, Off-(On), 125 V, 3 A, A souder	Farnell
Bouton d'action**	2.9€	RAFI 1.10.001.151/0301 Commutateur à bouton-poussoir, SPST-NC, Off-(On), 250 V, 700 mA, Vis	Farnell
LEDs	<< 1€	Ex : (rouge) LUMEX SSL-LX3044IC. LED, RED, T-1 (3MM), 125MCD, 635NM	Farnell
Boitier	9.06 €	HAMMOND 1591DBK Boîtier, Retardateur de flammes, UL94V-0, IP54, Multi-usage, 150 mm, 80 mm, 50 mm	Farnell

*l'Arduino mini pro a ceci de particulier que plusieurs fabricants peuvent fabriquer la carte : c'est pourquoi les prix diffèrent du simple au quadruple en fonction du fait que l'on achète la version « officielle » ou une version sans marque. Après essai dans notre application, il semble qu'il n'y a pas de problème majeur empêchant l'utilisation de la carte à bas prix.

** Le bouton d'action choisi ici est normalement fermé contrairement aux autres. Un autre bouton réclamera sans doute un changement dans le code de l'Arduino.

L'estimation donne un total compris entre 50.6 et 65.5 euros

Passerelle

La passerelle ne devrait être conçue qu'une seule fois pour avoir un système complétement fonctionnel. On compte également ici le câble FTDI nécessaire à la programmation des Arduinos. L'adaptateur Xbee sert à la fois pour la passerelle et la configuration initiale des Xbee des télécommandes.

Composant	Prix (HT)	Référence	Fournisseur
Adaptateur Xbee UART/USB	7.83€	FT232RL xbee USB adaptateur série v1.2 module de carte pour Arduino	MiniInThebox.com
Xbee S2	20.82 €	DIGI INTERNATIONAL XB24-BWIT-004 MODULE ZIGBEE XBEE ZNET 2.5 1MW	Farnell
Câble FTDI	18.16€	FTDI TTL-232R-3V3 CABLE USB VERS TTL LEVEL SERI CONVERTER	Farnell

A propos de la PCB et des améliorations à apporter

Circuit imprimé

Notre circuit a été réalisé sur platine de prototypage (ou protoboard) ce qui n'est pas une solution viable pour une implémentation réelle. C'est pourquoi nous avons décidé de créer une PCB (Printed Circuit Board) afin de simplifier la conception. Cela avait plusieurs objectifs :

- S'affranchir d'adaptateurs de « pin headers (connecteurs) pour la Xbee (écart entre les pin de 2mm non standards)
- Utiliser notre régulateur (technologie CMS : à souder sur une plaque)
- Simplifier la mise en œuvre, la carte étant un support pour « brancher » les différents composants de manière simple.

La conception de cette carte étant arrivée en fin de projet, nous avons pu en créer une, mais des erreurs la rendent impossible à utiliser. Elle est cependant améliorable.

Améliorations possibles

Des améliorations sont à apporter à notre projet, notamment sur des points que nous n'avons pas eu le temps de gérer.

Tout d'abord, du point de vue matériel, nous avions commencé à concevoir un moyen de visualiser le niveau de batterie restant sur la télécommande. La passerelle aurait pu envoyer un message demandant son niveau de batterie au boitier et celui-ci lui répondre avec un indicateur simple de niveau (très haut, haut, moyen, bas…). La difficulté vient de la spécificité de la pile comme source d'énergie ce qui nous a empêchés de finir à temps cette partie.

Même chose pour l'intensité du signal, qui peut être réglé en fonction de la distance. Plus la télécommande est proche, moins la transmission du signal utilise de puissance. Cette fonction n'a pas été implémentée par manque de temps.

Une gestion plus poussée de la consommation aurait été atteint avec l'implémentation de modes de veille pour l'Arduino et la Xbee. Là encore, les messages permettant d'activer cette fonction sont là, mais la fonction en elle-même n'est pas implémentée.

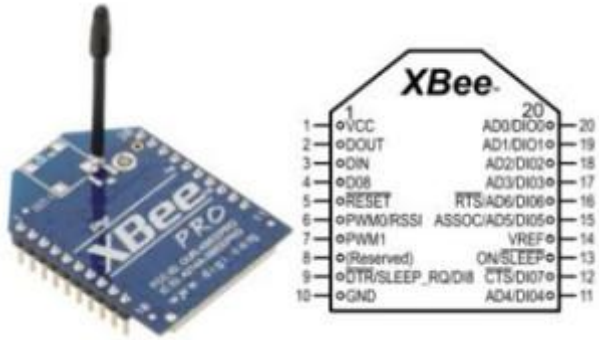
Au niveau du logiciel d'interfaçage VoxPopuli, certains objectifs non pas non plus été remplis. Le manque le plus important est l'absence d'une interface graphique pour contrôler et visualiser le réseau de télécommandes. Ce logiciel est d'autant plus nécessaire que l'implantation du réseau de télécommande dans une salle de spectacle peut se révéler fastidieuse avec un simple outil telnet. Certains points de ce logiciel de contrôle ont déjà été établis, notamment le fait qu'il soit codé en QML ce qui permet de le déployer sur des téléphones. Il manque aussi à VoxPopuli la possibilité d'envoyer et de recevoir des évènements OSC, l'implémentation de telles interactions n'est pas particulièrement difficile (la librairie OSCPack a déjà été choisie) mais le temps nous a manqué. Enfin le code aurait besoin d'être en intégralité nettoyé et commenté, tâche fastidieuse mais nécessaire pour qu'il puisse être ensuite réutilisé par d'autres.

Conclusion

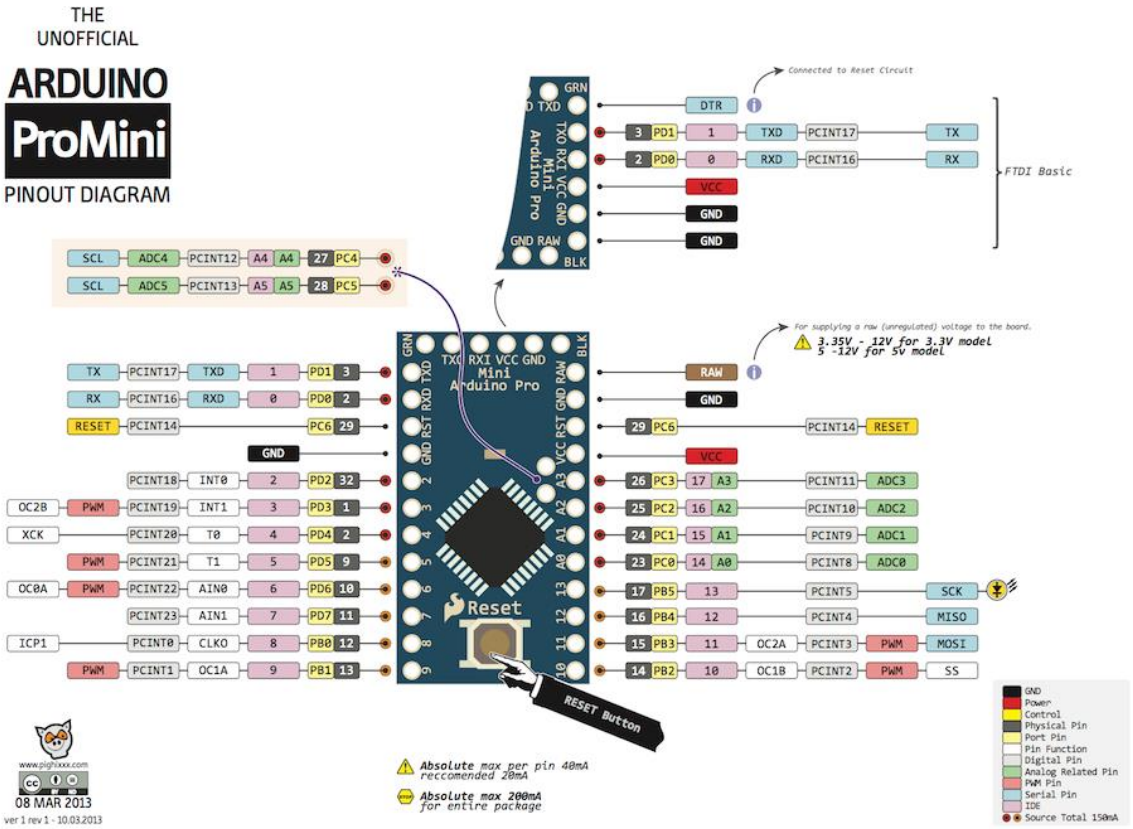
Ce projet ambitieux mais d'un intérêt certain a fait appel à des compétences dans de nombreux domaines, que ce soit en programmation, en système embarqué, en électronique et en télécommunications. Bien que 100% des objectifs n'est pas été rempli dans ce projet, nous pensons avoir fourni une solution viable pour faire de « une histoire simple » un projet unique. Nous souhaiterions donc suivre un minimum ce projet après la fin du PRT ou que celui-ci soit repris charge rapidement par une entité de l'INSA afin de finir l'implémentation, notamment en ce qui concerne le circuit imprimé.

Annexes

A.1 Xbee pinout diagram



A.2 Arduino mini pro pinout diagram



A3 Code Arduino

```
////// FRAME ALPHABET

//RX

#define FRAME_END 0xFF
#define SLEEP_CMD 0xC8
#define ACTIVE_MODE_CMD 0x96
#define MUTE_MODE_CMD 0x64
#define ASK_MODE_CMD 0x50
#define BATT_STATUS_CMD 0x10
```

```
  #define LED_CMD 0x20
  #define LED1_CMD 0x21
  #define LED2_CMD 0x22
  #define LED3_CMD 0x23


  //TX


  #define BUTTON_LEFT_TX 0xB0
  #define BUTTON_UP_TX 0xB1
  #define BUTTON_RIGHT_TX 0xB2
  #define BUTTON_DOWN_TX 0xB3
  #define BUTTON_ACTION_TX 0xB4
  byte                                                    buttonLabel[]=
{BUTTON_LEFT_TX,BUTTON_UP_TX,BUTTON_RIGHT_TX,BUTTON_DOWN_TX,BUTTON_ACTION_T
X};

  #define BATT_STATUS_TX 0x10

  ////// PIN DEFINITION

  #define PIN_ON_SLEEP 9

  #define BUTTON_LEFT 8
  #define BUTTON_UP 7
  #define BUTTON_RIGHT 6
  #define BUTTON_DOWN 5
  #define BUTTON_ACTION 4
  int                                                     buttonPin[]=
{BUTTON_LEFT,BUTTON_UP,BUTTON_RIGHT,BUTTON_DOWN,BUTTON_ACTION};

  #define LED1 10
  #define LED2 11
  #define LED3 3

  #define ADC_PIN A0

  ////// GLOBAL VARIABLES

  //FLAGS AND COUNTERS

  bool skipProcessFlag = false;
  bool Active_Mute = false;
  int IbRx=0;
  int IbTx=0;

  // BUFFERS

  int BufferRx[20];
  int BufferTx[20];
  int ButtonBuff[] = {1,1,1,1,1};

  // STATUS

  int LED1Status=0;
  int LED2Status=0;
  int LED3Status=0;


  //// SETUP

void setup() {

  //Serial operations
  Serial.begin(9600);

  ///////Pin modes
```

```
  pinMode (PIN_ON_SLEEP, INPUT);

  //Led

  pinMode (LED1, OUTPUT);
  pinMode (LED2, OUTPUT);
  pinMode (LED3, OUTPUT);

  //Buttons

  pinMode (BUTTON_LEFT, INPUT_PULLUP);
  pinMode (BUTTON_UP, INPUT_PULLUP);
  pinMode (BUTTON_RIGHT, INPUT_PULLUP);
  pinMode (BUTTON_DOWN, INPUT_PULLUP);
  pinMode (BUTTON_ACTION, INPUT_PULLUP);

}



  ///// MAIN LOOP


void loop() {

  skipProcessFlag==false;
  while(Serial.available() > 0) {

    BufferRx[IbRx]= Serial.read();
    IbRx++;
    if (BufferRx[IbRx-1]==FRAME_END){
      processMessage();
      IbRx=0;
    }

  }
  if (skipProcessFlag==false){

    if (Active_Mute==true){
      buttonProcess();
    }
    while (digitalRead(PIN_ON_SLEEP)!=1){

    }

    transmitTx();
  }
}



  /////FUNCTIONS

  // RX PROCESSING --- CAN BE BETTER HANDLED ---

void processMessage(){

  if (BufferRx[0]== ACTIVE_MODE_CMD){
    setActiveMode();
  }
  if (BufferRx[0]== MUTE_MODE_CMD){
    setMuteMode();
  }
  if (BufferRx[0]== ASK_MODE_CMD){
    byte modeStatus[]={ASK_MODE_CMD,Active_Mute, FRAME_END};
    TxToBuffer(modeStatus,3);
  }
  if (BufferRx[0]== SLEEP_CMD){
    setSleepMode(BufferRx[1]);
```

```
  }
  if (BufferRx[0]== BATT_STATUS_CMD){
    batteryStatus();
  }
  if (BufferRx[0]== LED_CMD){
    toggleLED(BufferRx[1]);
  }
  if (BufferRx[0]== LED1_CMD){
    toggleLED1(BufferRx[1]);
  }
  if (BufferRx[0]== LED2_CMD){
    toggleLED2(BufferRx[1]);
  }
  if (BufferRx[0]== LED3_CMD){
    toggleLED3(BufferRx[1]);
  }

}


  // TRANSMISSION STUFF  --- PIN ON SLEEP HANDLED IN MAIN LOOP

void transmitTx (){
    for( int i=0 ; i<IbTx ; i++){
      Serial.write(BufferTx[i]);
    }

  IbTx=0;
}

void TxToBuffer ( byte msg[], int length){
  for (int i=0; i<length ; i++){
    BufferTx[IbTx]=msg[i];
    IbTx++;
  }
}

  // ACTIVE/MUTE HANDLING

void setActiveMode(){
  Active_Mute =true;
}

void setMuteMode(){
  Active_Mute = false;
}

  // SLEEPMODE ---- WORK IN PROGRESS DON'T USE ----

void setSleepMode(int time){
  int length = 5;
  byte sleepMsg1[] ={'A','T','S','M',4};
  TxToBuffer (sleepMsg1,length);
  transmitTx;
  byte sleepMsg2[] ={'A','T','S','M',4};

}

  // BATTERYSTATUS ---- ADC on 10 bit FIX to make ---

void batteryStatus(){

  int sensorValue;
  sensorValue = analogRead(ADC_PIN);
  sensorValue=sensorValue/(1023)*(255);
  if (sensorValue==255){
    sensorValue=254;
  }
```

```
  byte battStatusTx[]={BATT_STATUS_TX,sensorValue,FRAME_END};
  TxToBuffer(battStatusTx,3);

}

  // LED MANAGEMENT

void toggleLED(int value){
  if (value==255){
    byte statusLED[]={LED_CMD,LED1Status,LED2Status,LED3Status,FRAME_END};
    TxToBuffer(statusLED,5);
  }
  else{
    int realValue=value;
    if(value==254){
      realValue=255;
    }
    analogWrite(LED1, realValue);
    analogWrite(LED2, realValue);
    analogWrite(LED3, realValue);
    LED1Status=value;
    LED2Status=value;
    LED3Status=value;

  }


}

void toggleLED1(int value){
  if (value==255){
    byte statusLED1[]={LED1_CMD,LED1Status,FRAME_END};
    TxToBuffer(statusLED1,3);
  }
  else{
   int realValue=value;
    if(value==254){
      realValue=255;
    }
  analogWrite(LED1, realValue);
  LED1Status=value;
  }


}

void toggleLED2(int value){
  if (value==255){
    byte statusLED2[]={LED2_CMD,LED2Status,FRAME_END};
    TxToBuffer(statusLED2,3);
  }
  else{
   int realValue=value;
    if(value==254){
      realValue=255;
    }
  analogWrite(LED2, realValue);
  LED2Status=value;
  }

}

void toggleLED3(int value){
  if (value==255){
    byte statusLED3[]={LED1_CMD,LED3Status,FRAME_END};
    TxToBuffer(statusLED3,3);
  }
  else{
   int realValue=value;
```

```
    if(value==254){
      realValue=255;
    }
  analogWrite(LED3, realValue);
  LED3Status=value;
  }
}



  // BUTTON HANDLING

void buttonProcess(){

  delay(1);

  int tempBuffer[5];

  for (int i=0 ; i<5;i++){

    tempBuffer[i]=digitalRead(buttonPin[i]);
    if(buttonLabel[i] == BUTTON_ACTION_TX)
      tempBuffer[i] = !tempBuffer[i];

    if (tempBuffer[i]!= ButtonBuff [i]){
      byte buttonTx[]={buttonLabel[i], ButtonBuff[i],FRAME_END};
      ButtonBuff [i] = tempBuffer[i];
      TxToBuffer(buttonTx,3);
    }
  }

}
```