

PFE IMA5

Jukebox multi-pièces



Alexandre Jouy

Table des matières

PFE IMA5	1
Jukebox multi-pièces	1
Introduction.....	3
1. Méthode de travail	4
2. Architecture des modules	5
2.1. Présentation des modules.....	5
2.2. Type de module 1	5
2.3. Type de module 2	5
2.4. Une solution DIY pour des coûts maîtrisés.....	6
3. Déroulement du projet.....	7
3.1. Recherches préliminaires	7
3.2. Sprint 1	8
3.3. Sprint 2	9
3.4. Sprint 3	10
3.5. UPnP et DLNA pour Android.....	13
3.6. UPnP et DLNA sur PC	14
Les solutions existantes.....	14
Fonctionnement des protocoles	15
Partage des médias	16
L'interface.....	17
Résultats	18
Objectif futur	18
Sources et documents	18
Conclusion	19
Annexes	20
1. Schéma de l'amplificateur à LM386	20
2. Schéma de l'amplificateur à LM3886	20

Introduction

Le but de ce projet est de réaliser un jukebox multi-pièces moderne. Concrètement, cela consiste à réaliser un dispositif pouvant jouer de la musique streamée depuis un smartphone, une tablette ou un ordinateur, et ce dans plusieurs pièces à la fois. Dans ce rapport je présenterai tout d'abord le travail effectué jusqu'aux vacances de Noël, puis je vous parlerai des différentes étapes qui ont abouti à la création de l'application actuelle.

1. Méthode de travail

Dans un premier temps nous nous sommes réunis avec nos tuteurs, Rodolphe Astori et Alexandre Boé pour faire un point de départ sur le projet, ainsi que la manière dont nous allions nous y prendre.

On nous a alors proposé de travailler selon la méthode Scrum dont le principe général est de découper la durée totale du projet en plusieurs « sprints » beaucoup plus courts d'une quinzaine de jours. Un objectif est fixé au début de chaque sprint et nous devons essayer de s'y tenir, en n'avancant que dans la direction donnée. A la fin de chaque sprint une réunion est organisée de manière à faire le point sur ce qui a été fait, définir les objectifs du prochain sprint et éventuellement corriger la trajectoire.

Les deux premières semaines de notre projet ont été consacrées aux recherches préliminaires et nous avons commencé le fonctionnement par sprint la semaine du 15 octobre.

2. Architecture des modules

2.1. Présentation des modules

Pour clarifier les choses, nous appellerons « module » le haut-parleur et ses équipements embarqués, la partie applicative pour streamer la musique étant mise de côté.

Dans le cadre de ce projet, nous sommes partis sur deux types de module distincts puisque nous souhaitons proposer au public une solution modulaire et non fermée. Dans les deux solutions, nous avons choisi d'ajouter à la Raspberry Pi une carte son USB afin d'améliorer considérablement la qualité sonore que nous envoyons vers le haut-parleur.

2.2. Type de module 1

Le premier module embarque une Raspberry Pi avec MusicBox pour la réception de la musique ainsi qu'une carte son USB.

Cette solution permet à l'utilisateur de brancher n'importe quel haut-parleur actif à la carte son avec un connecteur jack 3.6 mm. Il pourra ainsi garder son propre haut-parleur s'il le désire.

2.3. Type de module 2

Le second module embarque lui aussi une Raspberry Pi avec MusicBox ainsi qu'une carte son USB. Cependant nous proposons en plus dans celui-ci un amplificateur audio auquel nous connecterons un haut-parleur passif.

L'avantage de cette solution est de maîtriser la qualité audio du haut-parleur pour notre utilisateur et de ne fournir qu'une seule « boîte » avec solution toute intégrée.



Schéma de principe de la chaîne de transmission

2.4. Une solution DIY pour des coûts maîtrisés

Un des objectifs importants de ce projet est le coût puisque nous verrons par la suite après établissement d'un état de l'art (Sprint 2) qu'un haut-parleur multi-room dans le commerce se trouve dans une fourchette de prix allant de 100 à 500 euros. Il est donc évident que l'achat d'une enceinte par pièce revient très rapidement assez cher.

Notre but sera donc d'obtenir un module ayant un coût maximum de 80 euros pour une solution prête à l'emploi. La modularité du projet permettra à l'utilisateur de mettre le prix qu'il souhaite pour la qualité sonore de son choix (un haut-parleur actif pour le module 1, un couple amplificateur/haut-parleur pour le module 2).

Le danger majeur porte sur la qualité audio qui diminue très rapidement quand le coût baisse. La somme de 80 euros est un bon compromis pour notre projet.

Nous avons aussi vu très récemment que la Raspberry Pi Zero à 6 euros d'une puissance suffisante pour notre projet était disponible, contre une trentaine d'euros pour la solution actuelle. Elle a également l'avantage d'être beaucoup plus compacte que le modèle B+.

Cette solution permettra dans le futur de pouvoir faire diminuer le coût total du projet pour une personne voulant le réaliser par la suite.

3. Déroulement du projet

3.1. Recherches préliminaires

Les deux premières semaines ont été consacrées aux recherches préliminaires ainsi qu'à réfléchir aux solutions à mettre en place pour le mener à bien. Nous nous sommes convenus qu'il fallait développer :

- Une enceinte embarquant un amplificateur audio et un système pour recevoir le wifi
- Une application sur smartphone permettant de choisir et streamer le flux musical
- Une application sur PC permettant la même chose

Pour l'enceinte, nous avons choisi d'utiliser une Raspberry Pi disposant d'une distribution Linux embarquée et permettant ainsi de lui ajouter simplement une clé Wifi ainsi qu'une carte son USB. Cette solution nous est naturellement venue en tête car elle est simple à mettre en œuvre et relativement peu coûteuse.

Du côté de l'amplification nous avons retenu en premier temps deux solutions envisageables. La première était un amplificateur à base de LM386. Cette solution est peu coûteuse et extrêmement simple à réaliser mais ne permet d'obtenir qu'une puissance de sortie faible et un son de qualité médiocre (annexe 1). La deuxième était un amplificateur à base de LM3886 qui permet d'avoir une bonne puissance de sortie et un son de bonne qualité au détriment d'être un peu plus coûteuse (annexe 2).

D'un point de vue logiciel nous nous sommes dans un premier temps intéressés à l'application sur PC. Nous pensions développer l'application en C++ en utilisant Qt, un framework graphique fonctionnant sur toutes les plateformes (Windows, Linux et Mac OS).

Nous avons également discuté avec nos tuteurs de manière à définir les points essentiels constituant le cahier des charges de notre projet. Le projet doit être :

- Modulaire
- Entièrement réalisable soi-même
- Relativement peu cher
- Open source

L'aspect modulaire est défini par le fait de pouvoir inter changer facilement les différentes parties comme l'amplificateur, les enceintes, la carte son, etc... L'objectif à long terme étant un produit dont le coût est maîtrisé et qui correspond à chacun.

3.2. Sprint 1

A la fin des recherches préliminaires nous avons alors défini les objectifs à atteindre en fin de sprint 1. Il s'agissait d'arriver à streamer une musique depuis un appareil quelconque, vers une Raspberry Pi Connectée à une enceinte.

Nous avons alors pris la décision de partir sur une application destinée à fonctionner sur un PC. Nous avons choisi de développer cette application en C++, en utilisant le framework Qt, car il permet la réalisation de programmes graphiques sur toutes les plates formes.



On a alors commencé à développer une application en C++ tout en cherchant des méthodes pour arriver à streamer simplement sur la Raspberry Pi. Une des voies envisagée était d'utiliser une librairie VLC intégrable à Qt disponible sur le site vlc-qt.tano.si. L'avantage de passer par VLC pour streamer un flux musical est que c'est un logiciel open source, disponible à la fois sur Windows, Linux, Mac OS et qui s'intègre parfaitement à Qt.

On a également réussi à streamer un flux musical depuis VLC, ainsi que depuis un Mac, en installant SharePort, vers la Raspberry Pi.

A la fin du sprint, on a alors effectué un retour sur ce qu'on avait fait avec nos tuteurs ainsi que Jean-Eude Laurenge. On n'a malheureusement pu faire la démonstration que du stream depuis un Mac.

On a alors rediscuté de l'avancement du projet et des objectifs futurs. En effet, la démonstration n'était pas convaincante, et assez restreinte par rapport au résultat attendu.

On a déterminé ensemble qu'il était alors nécessaire pour le sprint 2 de :

- Faire une veille technologique complète sur les solutions existantes
- Se concentrer sur l'application mobile dans un premier temps.

En effet, l'intérêt principal de pouvoir streamer de la musique dans plusieurs pièces à la fois est de pouvoir contrôler la musique tout en se déplaçant. L'intérêt de contrôler le flux musical à l'aide d'un PC est donc assez limité.

On a alors défini comme objectifs pour le sprint 2 :

- Faire un état de l'art des technologies existantes
- Arriver à créer une chaîne de transmission complète du flux musical (du mobile jusqu'aux enceintes)

3.3. Sprint 2

Dans un premier temps il s'agissait de réaliser une veille technologique décrivant les solutions existantes autant au niveau logiciel et matériel DIY que celles commercialisées par des marques connues. Il fallait également arriver à streamer de la musique vers au moins une enceinte en utilisant les solutions trouvées.

Les solutions commercialisées (sous le nom d'enceintes multiroom) sont les suivantes :

- Sonos, haut de gamme, à partir de 229€
- Samsung, milieu de gamme, à partir de 99€
- Bose, haut de gamme, à partir de 199€
- LG, milieu de gamme, à partir de 179€
- Yamaha, haut de gamme, à partir de 299€
- Google Chromecast, entrée de gamme, à partir de 39€



L'intérêt de la Chromecast par rapport à toutes les enceintes existantes, c'est qu'elle permet de transformer n'importe quelle enceinte en système intelligent

Nous avons également fait des recherches sur les logiciels existants (autres que les logiciels constructeurs des enceintes précédemment présentées) permettant de streamer de la musique. Ces recherches se sont portées sur les principaux systèmes d'exploitation, et plus particulièrement pour les mobiles.

En effet, nous nous sommes convenus avec nos tuteurs qu'il était judicieux de privilégier en priorité l'application mobile plutôt que commencer par développer une application pour ordinateurs. On a alors répertorié les solutions déjà existantes qui paraissaient être les plus performantes et les moins compliquées à mettre en place :

- Pour mobile :

- Android : BubbleUPnP
- Iphone : AirPlay (intégré)



- Pour PC :

- Windows : Windows Media Player (intégré)
- Mac OS : AirPlay (intégré)



Pour la Raspberry Pi nous avons choisi d'installer un OS Linux modifié nommé MusicBox. Ce système d'exploitation est prévu pour gérer les flux musicaux. En effet, il intègre un serveur DLNA, la possibilité d'y associer un compte Spotify ou encore de streamer un flux Youtube ou une musique située sur une clé USB branchée à la Raspberry. Il dispose également d'une interface web qui permet de contrôler facilement tous les flux et son code est entièrement open source. C'est donc la solution idéale pour streamer facilement de la musique depuis un mobile en utilisant le DLNA.

Lors de la réunion de fin de sprint nous avons effectué une démonstration d'une chaîne complète. Nous avons réussi à diffuser deux musiques différentes depuis un même portable vers deux enceintes séparées. L'objectif fixé en début de sprint est donc atteint et nous avons pu définir de nouveaux objectifs pour le sprint suivant :

- Réaliser notre propre application et arriver à streamer une musique vers une enceinte
- Réaliser le PCB du circuit d'amplification à base de LM3886

L'intérêt de réaliser une application Android pour streamer de la musique est qu'elle soit open source, contrairement à BubbleUPnp.

3.4. Sprint 3

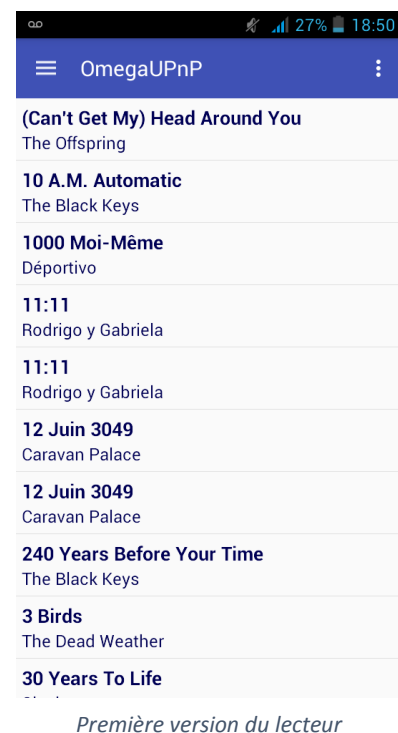
Pour le sprint 3 nous nous sommes concentrés sur la réalisation d'une application Android. En effet, comme iOS dispose d'une solution intégrée déjà existante nous avons jugé qu'il n'était pas nécessaire de développer une application dédiée.

Nous avons choisi de développer notre application en utilisant Android Studio. Cette application est donc réalisée en Java, et nous avons fait en sorte qu'elle soit compatible avec tous les téléphones disposant de la version 4.1 (Jelly Bean) ou plus. Cela permet qu'un maximum d'utilisateurs puisse profiter de l'application tout en la rendant compatible avec les versions récentes.

Dans un premier temps nous avons créé une activity Android standard disposant d'un panneau latéral qui nous permettra par la suite de rajouter les fonctions de recherche de serveur DLNA.

Dans un premier temps nous avons créé une classe Song qui contient les variables liées à chaque chanson et les méthodes nécessaires pour pouvoir les manipuler.

Il a ensuite été nécessaire de créer une autre classe contenant les méthodes permettant de manipuler les chansons. Cette classe, nommée MusicService, permet de lier la liste des chansons que l'on a récupérées dans l'Activity principale au lecteur de média. En effet, le



Première version du lecteur

lecteur de média est un service sous Android et est donc indispensable à lier si l'on veut lire un média quelconque.

Une fois ces étapes faites, nous avons affiché cette liste dans l'activité principale et l'avons trié par ordre alphabétique. On initialise alors le service permettant de lire les médias et on lit l'action de toucher une chanson dans la liste au démarrage de ce service. On a alors une première version du lecteur musical, qui pour l'instant est limitée à la lecture en local.

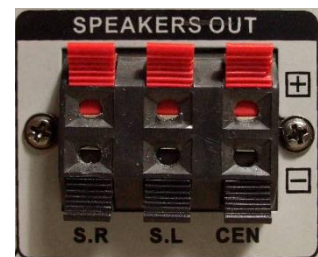
On a ensuite implanté une barre permettant de contrôler la lecture (lecture, pause, chanson suivante...). Les contrôles étant déjà présents dans le service « Media Player » il était seulement nécessaire de rajouter le widget contenant notre barre à l'activité principale puis de lier les boutons aux contrôles.

Nous avons également réalisé le layout des deux PCB du module 2 : un pour l'amplification à base de LM3886 et un pour le circuit d'alimentation. Afin de dissiper sa chaleur importante lors de l'utilisation de l'amplificateur, nous avons commandé un dissipateur thermique sur lequel nous visserons le LM3886.

Nous nous sommes fortement inspirés de l'amplificateur en annexe 2 où les informations sur le site correspondant sont très détaillées et nous permettent de comprendre l'importance de beaucoup de composants dans un amplificateur audio.

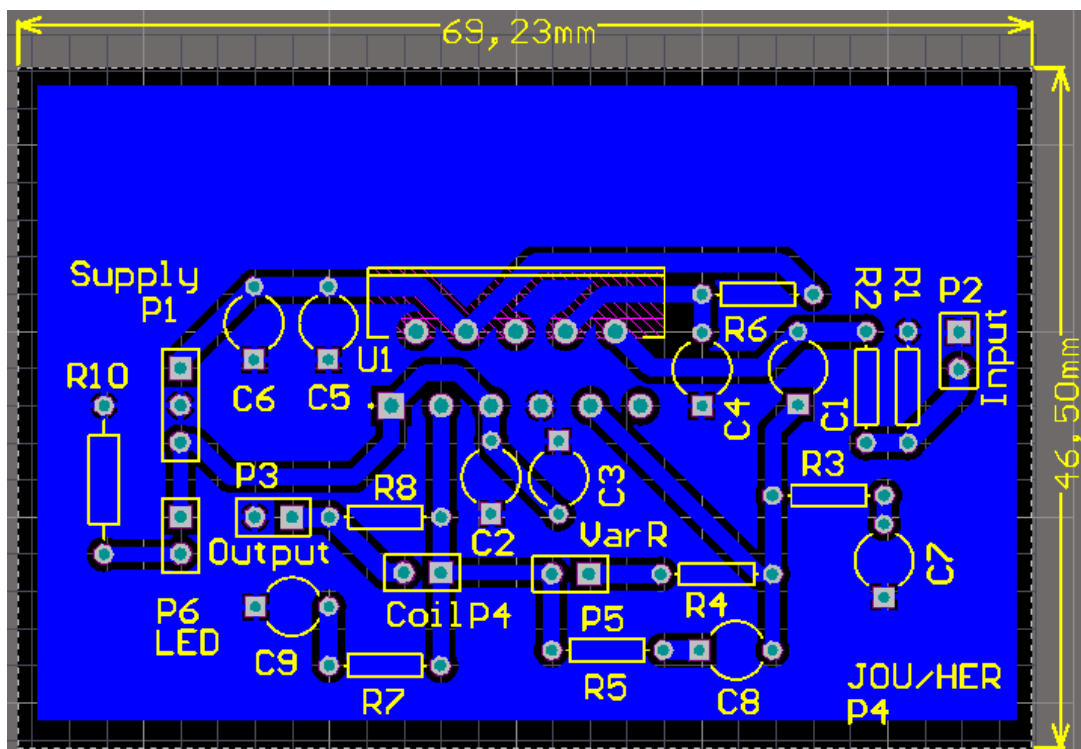
Par rapport au circuit d'origine, nous avons ajouté une LED de fonctionnement de l'amplificateur, un switch ON/OFF ainsi qu'un potentiomètre rotatif pour pouvoir agir sur le volume sonore.

Pour l'entrée audio, l'utilisateur trouvera en façade de l'amplificateur une entrée stéréo RCA ainsi qu'une entrée jack 3.6 mm. Pour la sortie, nous avons prévu un bornier standard pour connexion de haut-parleur ainsi qu'une sortie jack femelle 3.6 mm.



Bornier 2.1

La version quasiment finale du PCB est donc la suivante :



Les objectifs de ce sprint n'ont donc pas tout à fait été atteints :

- Le lecteur Android ne permet pour l'instant que de lire localement les musiques
- Le PCB n'a pas encore été tiré

Il est donc nécessaire pour le sprint suivant de corriger au plus vite ces deux points de manière à ce que le projet puisse aboutir. Les objectifs pour le sprint suivant sont donc de faire en sorte d'arriver à streamer un flux musical vers une enceinte avec notre application ainsi que de tirer les PCB, les souder et commencer les tests.

Après les vacances de décembre, le travail n'a plus été séparé en sprints à proprement parler. Cependant des réunions régulières ont été organisées de manière à pouvoir rendre compte de l'avancement de chacun et à continuer à travailler dans la bonne direction.

Etant donné la forte séparation dans le groupe en terme de contenu, je présenterai dans ce rapport uniquement le travail que j'ai effectué. Néanmoins, j'ai également suivi le travail fait par mon binôme et serait capable d'en parler dans une certaine mesure.

3.5. UPnP et DLNA pour Android

L'objectif principal au retour des vacances était d'arriver à streamer une musique stockée localement sur un smartphone vers la Raspberry Pi flashée avec MusicBox. Pour cela j'ai choisi de continuer à travailler avec les protocoles UPnP et DLNA car au premier abord ils présentaient plusieurs avantages comparé à d'autres méthodes :

- Protocole résultant d'un accord de plus de 250 sociétés réparties à travers le monde ce qui garantit en théorie une uniformité et une compatibilité dans les appareils qu'on peut trouver dans le commerce
- Facilité d'accès aux spécifications
- Utilisation de plus en plus importante (objets connectés, télévisions, enceintes...)

Après quelques recherches sur internet, j'ai alors trouvé une librairie Java nommée Cling (<http://4thline.org/projects/cling/>) qui permet en théorie d'implémenter facilement tous les éléments nécessaires au stream en DLNA.

L'utilisation du Java est nécessaire pour développer sous Android et l'utilisation d'une librairie déjà existante permet de simplifier et d'accélérer le travail nécessaire pour arriver à une application fonctionnelle.

Dans un premier temps j'ai téléchargé les fichiers .jar contenant tous les éléments nécessaires au développement et j'ai essayé d'implémenter un code simple trouvé sur le site.

Après avoir configuré les éléments nécessaires dans le manifest pour autoriser l'application à accéder au réseau wifi ainsi qu'aux fichiers stockés sur le téléphone, il a fallu que j'ajoute les dépendances nécessaires à Gradle (le gestionnaire de projet utilisé par Android Studio). Je suis alors tombé sur plusieurs problèmes :

- Le peu de documentation sur le site officiel de Cling explique l'implémentation de la librairie sous Eclipse avec le gestionnaire de projet Maven
- Une des dépendances principales est Jetty, or cette dépendance n'est pas utilisable telle quelle sous Android.
- Il existe extrêmement peu de documentation concernant le DLNA sous Android

J'ai alors cherché un moyen de pouvoir faire fonctionner cette librairie sous Android Studio car l'extension Eclipse pour le développement Android n'est plus supportée officiellement par Google depuis le 26 juin 2015. (<http://goo.gl/hlijyf7>)

Il existe un projet open source nommé i-jetty qui permet de l'implanter sur mobile, mais cependant cela ne réglait pas entièrement mon problème car même en essayant d'implémenter les exemples relativement simples proposés sur le site de Cling, je rencontrais des erreurs de compilation.

J'ai alors cherché d'autres moyens d'implémenter le DLNA sous Android. Je suis tombé sur quelques projets open-source assez peu, voire pas du tout documentés comme CyberLink4Android par exemple.

J'ai alors fait appel à Benjamin Barbry, l'intervenant extérieur que nous avons eu pour les travaux pratiques d'architecture logicielle, et plus particulièrement pour l'initiation à Android. Je lui ai exposé mon problème, pour savoir s'il avait une solution afin de pouvoir au moins implémenter les services de bases nécessaires à la diffusion en DLNA. Une semaine plus tard il m'a alors répondu qu'il n'avait pas réussi à implanter la librairie non plus et de plutôt chercher dans une autre direction, par exemple d'essayer avec CyberLink4Android, que j'avais déjà essayé d'implanter.

J'ai alors parlé avec mes responsables de projet et nous avons déterminé qu'il était préférable de continuer le projet avec le développement d'une application sur PC plutôt que sur Android.

3.6. UPnP et DLNA sur PC

Les solutions existantes

Sur PC il existe de nombreuses solutions open source permettant le streaming vers des lecteurs DLNA, comme par exemple :

- Serviio (<http://serviio.org/>) un serveur DLNA multiplateformes permettant de streamer des médias vers des lecteurs
- Stream What You Hear (<http://www.streamwhatyouhear.com/>) un logiciel sous Windows permettant de diffuser tous les son du PC vers un lecteur DLNA
- Kodi (<https://kodi.tv/>) un autre serveur DLNA multiplateformes, dans la pratique je n'ai pas réussi à la faire fonctionner avec MusicBox sur la Raspberry Pi

L'intérêt de développer sa propre application réside alors dans le fait de pouvoir avoir un contrôle total sur le code et ainsi de pouvoir mettre et retirer facilement les fonctions jugées utiles. Elle est également open source ce qui permet à n'importe qui de pouvoir la récupérer et l'utiliser.

J'ai décidé de coder l'application en C# car c'est un langage que je connais bien et qui utilise le framework .NET de Microsoft. Le choix s'est porté sur ce langage principalement car c'est le langage permettant la création d'une interface graphique que je maîtrise le mieux.



Par la suite il sera possible d'obtenir une certaine portabilité de l'application sur d'autres systèmes d'exploitation grâce au projet Mono qui est une implémentation open source du framework .NET pour Linux, Mac OS mais aussi Windows.

Fonctionnement des protocoles

Dans un premier temps j'ai dû implémenter les protocoles nécessaires à la communication entre les différents éléments constituant mon « réseau » UPnP/DLNA. J'ai mis un temps considérable avant de réussir à lire pour la première fois une musique, et ce malgré les échantillons de code dont je disposais, car le guide décrivant le fonctionnement du DLNA et de l'UPnP est exhaustif (plus de 1700 pages). J'ai donc fait des recherches et utilisé Wireshark pour arriver à comprendre le fonctionnement global.



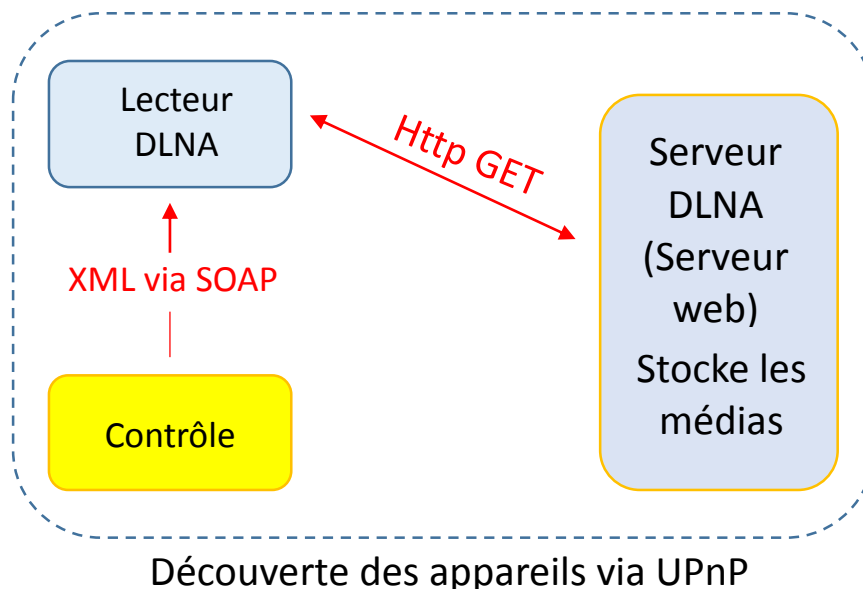
L'UPnP est en réalité basé sur le protocole SSDP (Simple Service Discovery Protocol). Il envoie un message multicast à l'adresse 239.255.255.250 en UDP sur le port 1900. Quand il reçoit une réponse, il lit alors une description en stockée dans un fichier XML à l'adresse qui lui a répondu. Cela permet ainsi à l'émetteur du message de connaître la liste des services disponibles pour chaque appareil.

Ce protocole décrit aussi les méthodes de contrôle des appareils multimédias. En l'occurrence le contrôle d'un lecteur DLNA se fait avec le protocole SOAP via http. Un message en XML est alors envoyé et permet de commander le lecteur (lecture, pause, etc...).

Pour la lecture en DLNA le client vient chercher le fichier directement sur le serveur. J'ai donc dans un premier temps essayé de trouver un serveur DLNA fonctionnel et simple pouvant héberger ma musique, si possible open source. Comme mes recherches n'étaient pas fructueuses je me suis alors penché sur les échanges entre le client et le serveur. Je me suis aperçu que le fichier était alors demandé avec un simple GET en http.

Dans un premier temps je me suis servi de WAMP pour voir si j'arrivais bien à récupérer un fichier sur mon ordinateur depuis MusicBox, sans succès. En regardant le trafic sur le réseau j'ai remarqué une erreur 403 (accès refusé) en réponse à la demande. J'ai alors modifié les droits d'accès sur mon serveur web et j'ai enfin réussi à lire un fichier sur la Raspberry Pi.

Principe de fonctionnement :



Partage des médias

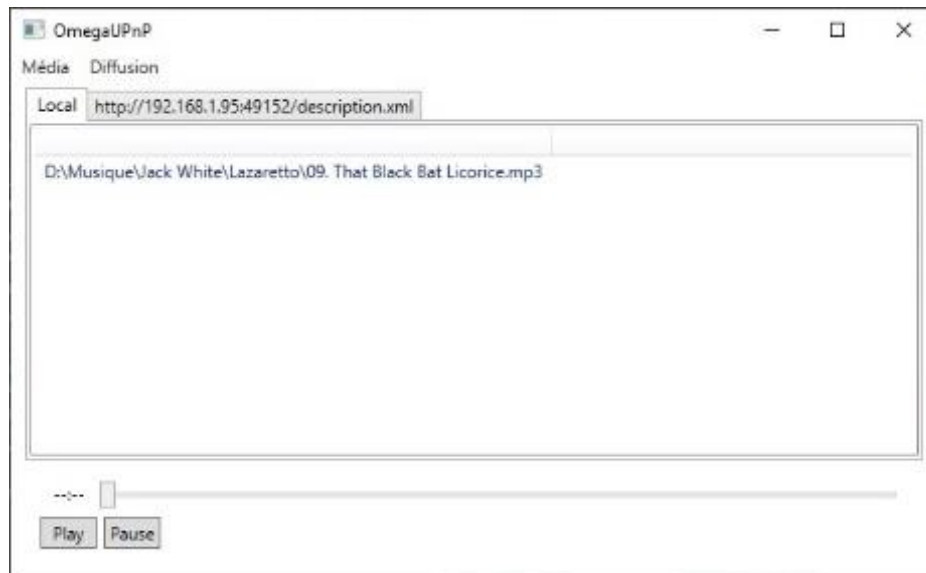
Pour gagner du temps j'ai décidé d'employer un serveur web portable prêt à l'emploi plutôt que de le coder moi-même. J'ai trouvé deux solutions envisageables :

- TinyWeb (<https://www.ritlabs.com/en/products/tinyweb/>) un serveur open source de 53Ko gérant le http et https
- Mongoose (<https://www.cesanta.com/products/mongoose/>) un serveur cross plateforme open source de 168Ko codé en C/C++ gérant le http et https

J'ai fini par choisir TinyWeb car il permettait de choisir en ligne de commande le port sur lequel le serveur doit être démarré et il est plus transparent pour l'utilisateur car il n'affiche pas d'icône à son lancement.

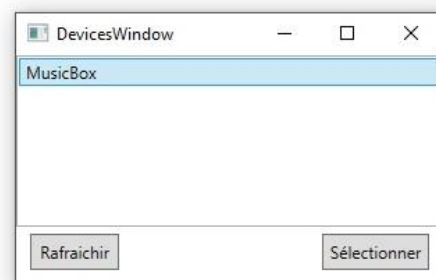
L'interface

J'ai alors commencé la réalisation d'une interface graphique. L'objectif était de créer une interface simple, légère et permettant à l'utilisateur de streamer facilement à un ou plusieurs appareil simultanément.



Pour éviter de bloquer l'interface graphique lors de la recherche de périphériques UPnP j'ai utilisé les threads. Ainsi, au démarrage de l'application, après le lancement du serveur web, un thread se lance et exécute la fonction bloquante qui envoie un message en multicast et attend la réponse.

Ce thread de recherche est aussi exécuté lors de l'appui sur le bouton « Rafraichir » de la fenêtre servant à sélectionner le lecteur sur lequel on désire streamer notre musique. Afin d'éviter de surcharger l'ordinateur et le réseau un `join()` est fait de manière à ne pouvoir lancer qu'un thread de découverte à la fois.



Résultats

L'application que j'ai développée est fonctionnelle et permet de streamer à une ou plusieurs enceintes connectées. Il reste néanmoins du code à écrire comme par exemple la possibilité d'ajouter directement un dossier contenant de la musique à la liste de lecture, ou encore une barre de lecture fonctionnelle.

Objectif futur

L'objectif suivant était de pouvoir coder une interface permettant d'utiliser une paire d'enceinte connectée (deux modules différents) de manière à pouvoir obtenir un effet stéréo. Pour cela il est nécessaire de séparer les flux musicaux en deux canaux et de pouvoir sélectionner facilement l'enceinte de droite et celle de gauche.

Cette dernière nécessité peut être problématique si les enceintes ont le même nom par défaut (MusicBox par exemple). Il faudrait alors mettre en place un système permettant de les différencier, comme par exemple un son lors de la sélection, et ainsi pouvoir les choisir facilement.

Ce système de stéréo peut également entraîner un autre problème non négligeable : la non synchronisation des deux canaux. En effet, la mémoire tampon est réglée sur le lecteur (ici la Raspberry) et donc est différente suivant quel lecteur l'utilisateur décide d'utiliser. Comme le temps nécessaire au remplissage de la mémoire dépend de la qualité de la connexion et que les enceintes connectées en Wifi peuvent avoir des différences parfois importantes de qualité de signal (position, antennes différentes, etc...) une différence audible peut alors rapidement se créer. Lors de l'implantation de cette fonctionnalité, il sera alors nécessaire de mettre en place un système de synchronisation entre les enceintes (comme par exemple attendre le remplissage du tampon des deux enceintes avant de lancer la lecture) permettant ainsi d'éviter au maximum un décalage dans le son.

Sources et documents

Le code ainsi que l'exécutable de l'application est disponible sur ma page Github (<https://github.com/Fraise/OmegaUPnP>). J'ai également rédigé un tutoriel destiné à un utilisateur non averti expliquant toutes les étapes nécessaires à la réalisation du module connecté.

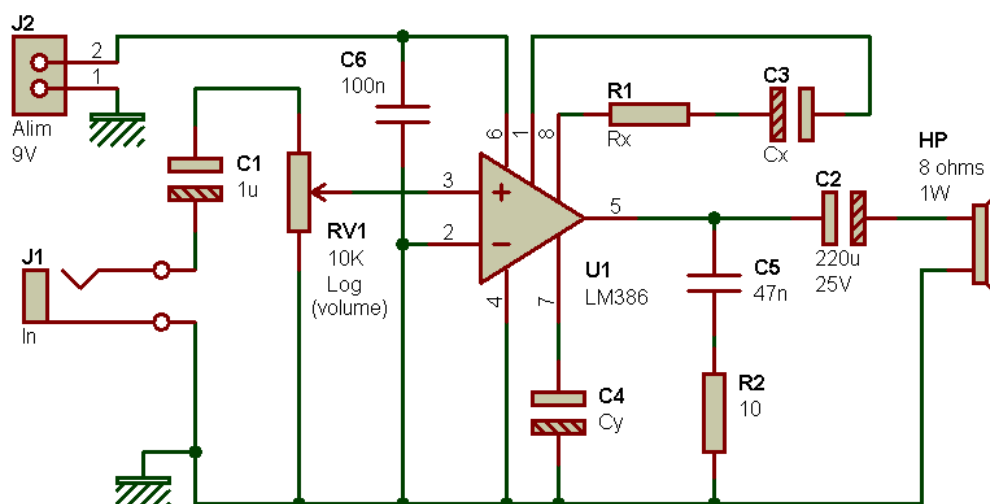
Conclusion

Au cours de ce projet j'ai rencontré plusieurs difficultés m'ayant fait prendre un retard considérable par rapport au planning initial. En effet, l'implémentation des services UPnP/DLNA dans l'application Android a demandé un temps conséquent, et malgré de nombreuses recherches et essais je n'ai pas réussi à arriver à quelque chose de fonctionnel. Le protocole en lui-même fut également assez difficile à saisir, non pas en terme de technique pure, mais dû au fait que la documentation était soit exhaustive, soit inexistante.

Au final, j'ai quand même réussi à avoir une application fonctionnelle, bien que limitée dans ses fonctionnalités, qui pourra être reprise et améliorée par qui le désire. La rédaction des instructions permettra à un utilisateur quelconque de pouvoir mettre en place lui-même les éléments lui permettant de streamer en DLNA sur son réseau local. Ainsi, dans l'ensemble le projet est fonctionnel, mais reste assez basique et nécessite d'être amélioré et poussé plus loin.

Annexes

1. Schéma de l'amplificateur à LM386



Pour un gain de 20 (26 dB) : ne rien raccorder aux broches 1 et 8 (les laisser en l'air)
 Pour un gain de 20 à 200 (26 dB à 46 dB) : résistance R1 + condensateur C3 entre broches 1 et 8.
 Exemples :
 - Pour un gain de 50 (34 dB) : R1 = 1K2 et C3 = 10uF
 - Pour un gain de 200 (46 dB) : Condensateur seul entre broches 1 et 8 - C3 = 10uF (R1 = 0)

2. Schéma de l'amplificateur à LM3886

