
# Rapport initial

## Application distribuée

RODRÍGUEZ, RICARDO MARTÍN

Année scolaire 2016-2017
Master Informatique M1
ILSEN
Université d'Avignon et des Pays de Vaucluse

# TABLE DES MATIÈRES

# 1 INTRODUCTION

Il faut développer une application distribuée qui permettra à des clients Android de jouer des fichiers de musique. Les chansons se trouveront dans un ensemble de serveurs distants.

**Le système comportera plusieurs modules :**

- Application cliente Android
- Un service "façade" qui recevra les commandes utilisateur.
- Un service de reconnaissance de la parole
- Un service de parsing de commandes
- Un service de traitement de commandes
- Un ou plusieurs serveurs de fichiers musicaux
- Un serveur middleware central qui permettra l'accès à tous les services

# 2 APPLICATION ANDROID

L'application Android sera sûrement développée en Java, utilisant la bibliothèque graphique JavaFX. Néanmoins, je pense analyser la possibilité de la réaliser en Python puisque le client du semestre précédent était fait en Python.

L'application permettra à l'utilisateur de connaître les serveurs de fichiers musicaux existants. Il sera de même possible de lister l'ensemble de chansons dans tous les serveurs. L'utilisateur pourra choisir une chanson et la jouer/arrêter avec des commandes manuelles ou des commandes vocales. La chanson sera reçue en forme de streaming. Pour ainsi faire, je compte utiliser la librairie VLC.

**À priori, les seules commandes vocales disponibles seront :**

- "Jouer X"
- "Arrêter"
- "Chercher X"

La commande vocale donnée par l'utilisateur ne sera pas structurée. Par exemple, on pourra dire "jouer X chanson", "jouer X", "jouer la chanson qui s'appelle X".

La commande sera capturée par le microphone du portable et envoyée directement au serveur middleware pour être traitée.

# 3 SERVEUR MIDDLEWARE

Je vais utiliser le framework Middleware Ice, créé par ZeroC. Afin de simplifier le déploiement, le middleware sera installé dans le même serveur physique que tous les services du système : reconnaissance, parsing et traitement de commandes.

Les deux langages de programmation à utiliser seront Java et Pyhton. En principe je vais éssayer d'implémenter la plupart de services en Python, notamment le module de parsing, parce que le semestre précédent je trouvait que l'intégration avec Ice et les bibliothèques utilisais était plus simple qu'en Java.

# 4 RÉCEPTION DE COMMANDES UTILISATEUR

Ce premier module sera le point d'entrée à l'application du côté client. Il s'agira d'un webservice SOAP qui fera l'appel aux modules nécessaires du Middleware.

**L'interface du webservice proposera :**

- commandeVocale(sequence<float> parole) : bool Recevra la commande vocale et retournera true si elle a été bien traitée. Le client pourra réagir par rapport au résultat de cette méthode.
- commandeManuelle(string commande, string chanson) : bool Les seules commandes possibles seront "jouer" et "arreter". Si on reçoit jouer, on va utiliser le paramètre chanson pour chercher la chanson à jouer. La commande retournera true si tout s'est bien passé, false dans le cas contraire.

# 5 PARSING DE COMMANDES

Le module permettra de parser la phrase prononcé par le client et d'identifier la commande et la chanson concernées. Il faut prendre en compte qu'il n'y aura pas de contraintes dans les phrases. Pour ainsi faire, je vais étudier les bibliothèques disponibles en Python. Dans le cours de ma formation en Uruguay j'ai appris à utiliser quelques outils de traitement de Langage Naturel contenus dans la bibliothèque NLTK, qui sont très puissants et plutôt simples à utiliser. http ://www.nltk.org/

**L'interface Ice fournira la méthode :**

- parsingPhrase(string phrase) : Commande La méthode reçoit la phrase en forme de string et retourne une structure de données contenant la commande à exécuter.

La structure de données Commande contiendra plusieurs champs afin de connaître l'action à exécuter.

```
struct Commande {
    bool error;         // Sera false si on n'a pas réussi
                        // à identifier la commande
    string commande;
    string chanson;
}
```

# 6 TRAITEMENT DE COMMANDES

Ce service sera chargé de recevoir les commandes utilisateur déjà interpretées et d'effectuer les traitements nécessaires. Il sera aussi chargé de maintenir une base de données actualisée des chansons existantes dans les serveurs.

Ce service devra stocker la liste de serveurs existants et la liste globale de chansons. Pour ainsi faire, je vais d'abord utiliser des structures en mémoire pour ensuite inclure une BDD SQLite.

Chaque chanson sera associée à tous les serveurs qui la contiennent.

**Le service proposera une interface avec les méthodes suivantes :**

- Côté client :
  - traiterCommande(string ipClient, Commande commande)
    Le client peut envoyer seulement deux types de commandes : jouer une chanson ou arrêter la chanson en cours.
    On reçoit en paramètre l'IP du client pour savoir à qui envoyer le flux de streaming musical.
    Lors de la réception d'une commande pour jouer une chanson, on va renvoyer la demande au premier serveur dans la liste de serveurs disponibles. Ensuite, ce serveur sera mis à la fin de la liste afin d'équilibrer la charge.

- Côté serveur de fichiers :
  - enregistrerServeur(string ipServeur) : int
    Permettra à un serveur de s'ajouter à la liste de serveurs de fichiers disponibles.
    On va retourner l'id associé à ce nouveau serveur.
  - supprimerServeur(int idServeur)
    Permettra à un serveur musicaux de se désinscrire de la liste. Toutes les chansons qui sont uniquement contenues dans le serveur seront supprimées.
  - notifierAjoutChanson(int idServeur, string chanson) : int
    Cette méthode sera appelée par les serveurs musicaux afin de notifier l'ajout d'une nouvelle chanson.
    Le service va regarder si une chanson avec le nom donnée existe déjà. Si elle existe, on retournera l'id actuel. Si la chanson n'existe pas, on va l'ajouter et retourner l'id associé.
    J'estime que celle-ci est la manière la plus simple pour avoir une BDD de chansons centrale actualisée.
  - notifierSuppressionChanson(int idServeur, int idChanson)
    Cette méthode sera appelée par les serveurs musicaux afin de notifier la suppression d'une chanson.

# 7 SERVEURS MUSICAUX

Le système permettra à plusieurs serveurs de fichiers musicaux de proposer leurs chansons.

Afin de simplifier la gestion des fichiers, je vais éssayer d'implémenter un processus en arrière plan qui va lire un répertoire spécifique. Les fichiers MP3 existant dans ce répertoire seront ceux disponibles pour le système. Le thread sera chargé de détecter l'ajout/suppression de fichiers dans le répertoire afin de faire appel aux méthodes de notification du Service de Traitement de commandes.

Le thread principal sera une application Python en console qui porposera à l'administrateur des options pour s'inscrire/désinscrire au serveur central. On aura aussi accès à la liste de chansons afin de détecter des possibles erreurs.

# 8 LIENS UTILES

**Bibliothèque NLTK** `http://www.nltk.org/`