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 webservice qui recevra les commandes utilisateur.
- Un serveur central qui va orchestrer l'accès à tous les modules
- Un module de reconnaissance de la parole
- Un module de parsing de commandes
- Un module de traitement de commandes
- Un méta-serveur qui connaîtra la liste globale de chansons
- Un ou plusieurs serveurs de fichiers musicaux

De toutes les parties, celle qui m'intéresse le plus est le traitement du langage naturel. C'est-à-dire, les modules de reconnaissance de la parole et du parsing de commandes. Je vais essayer de permettre au client d'exécuter plusieurs types de commandes vocales différentes et de pouvoir donner les commandes en prononçant des choses différentes pour une même action.

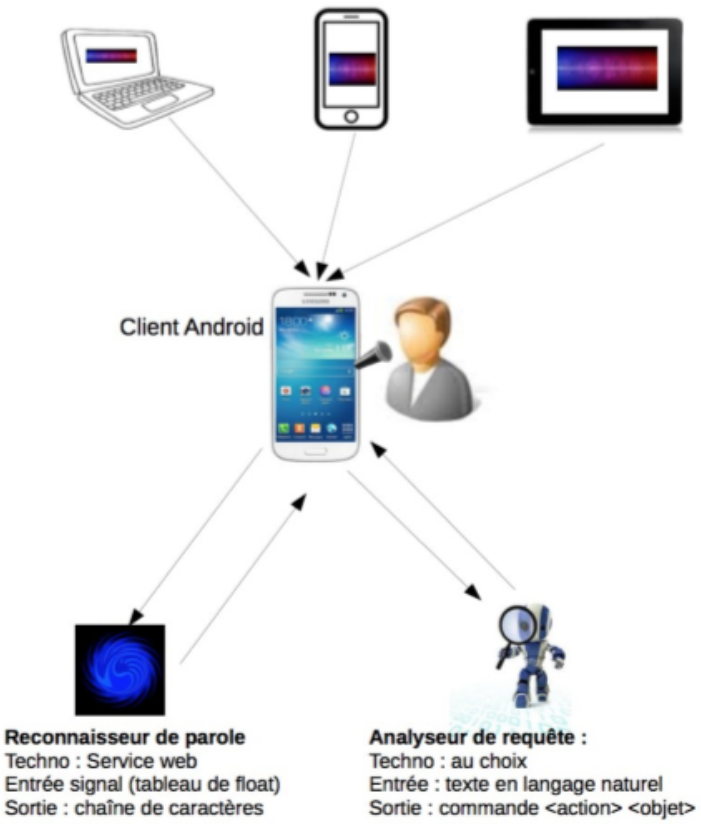

FIGURE 1 – Vision que le client a du système.

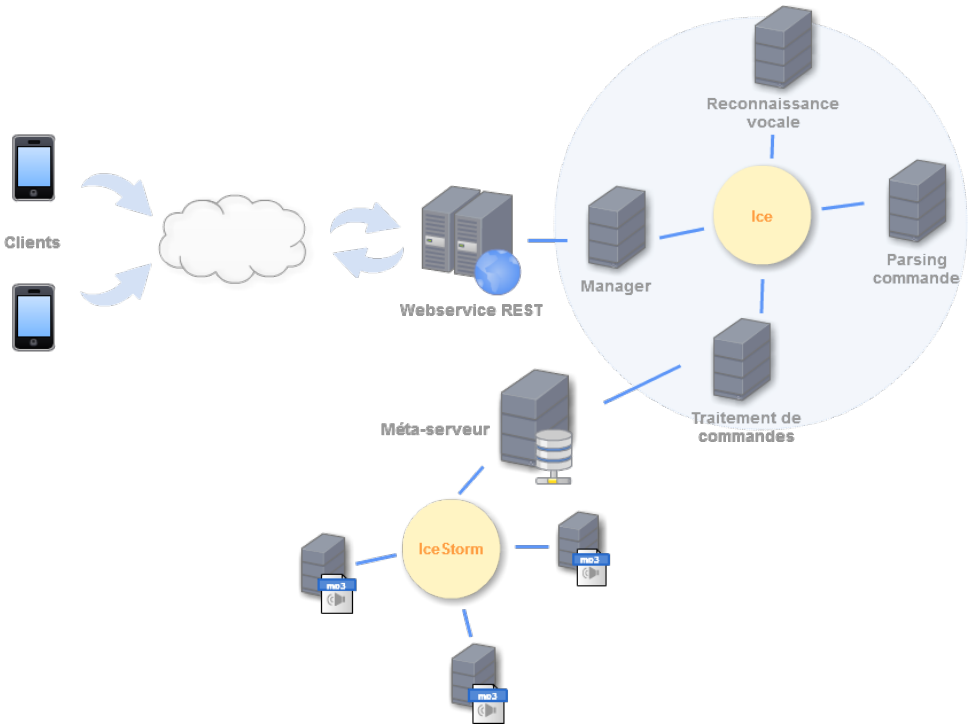

FIGURE 2 – Architecture du système.

2 APPLICATION ANDROID

L'application Android sera développée en Java, utilisant la bibliothèque graphique JavaFX.

Ce client permettra à l'utilisateur de connaître les fichiers musicaux existants dans l'ensemble de serveurs. Il 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.

Dans un premier temps, les commandes vocales disponibles seront :

- "Jouer X"
- "Mettre en pause"
- "Arrêter"
- "Chercher X"
- "Augmenter volume"
- "Baisser volume"
- "Silence"
- "Ajouter aux favoris"
- "Supprimer des favoris"
- "Éteindre le player"
- "Éteindre le player dans Y minutes"

Cette liste pourra changer par rapport à la difficulté de reconnaître les commandes et du temps disponible pour la réalisation de l'ensemble du projet.

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 central pour être traitée.

3 RÉCEPTION DE COMMANDES UTILISATEUR

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

Le webservice sera le point d'accès du côté utilisateur au serveur central, détaillé ci-dessus.

L'interface du webservice proposera :

- commandeVocale(float[] parole) : Commande
  Recevra la commande vocale, fera appel aux modules du système et retournera une structure de données qui indiquera la commande reconnue et si elle a été bien traitée. Le client pourra réagir par rapport au résultat de cette méthode.
- commandeManuelle(Commande) : bool
  Les commandes manuelles permettent d'utiliser le système sans être obligé à parler, par exemple dans des endroits trop bruyants. La commande retournera true si tout s'est bien passé, false dans le cas contraire.
  Le client pourra réagir par rapport au résultat de la commande. Par exemple, s'il reçoit false pour une demande de chanson, il saura qu'il n'y aura pas de streaming musical à consommer.

Le format de la structure de données Commande sera :

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

4 SERVEUR MANAGER

Le serveur manager sera à la charge d'orchestrer le traitement d'une commande, envoyé à lui par le webservice : appel à un premier module pour reconnaître la phrase, puis au parsing, etc.

Je vais utiliser le framework Ice, créé par ZeroC. Afin de simplifier le déploiement, plusieurs modules liés à la commande seront installés dans le même serveur physique : reconnaissance, parsing et traitement de commandes.

Le langage de programmation à utiliser sera Pyhton. En principe je vais essayer d'implémenter la plupart de services en ce langage, notamment le module de parsing, parce que le semestre précédent j'ai trouvé que l'intégration avec Ice et les bibliothèques utilisées était plus simple que, par exemple, avec Java.

5 RECONNAISSANCE VOCALE

Ce module recevra une commande en forme de voix (tableau de float) et retournera un string avec la phrase détectée.

L'interface Ice fournira la méthode :

- reconnaitreVoix(sequence<float> parole) : string La méthode reçoit la phrase en forme de son et retourne un chaîne de caractères.

Attention

J'ai encore un doute sur ce point. Il existent des bibliothèques installées de manière native sur Android qui font la reconnaissance vocale off-line. Les utiliser pourrait être la manière la plus simple d'implémenter cette fonctionnalité, par contre, si jamais on veut faire évoluer la technologie de reconnaissance, utiliser ce qui est installé dans le dispositif client n'est pas la meilleure option. Je vais partir de la base expliquée ci-dessus, il y aura un module à-part dans le serveur qui fera la tâche. Si au final ceci complexifie trop l'implémentation, je pourrais toujours utiliser les bibliothèques natives.

6 PARSING DE COMMANDES

Le module permettra de parser la phrase prononcée 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 du 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;
}
```

7 TRAITEMENT DE COMMANDES (MÉTA-SERVEUR)

Ce module sera chargé de recevoir les commandes utilisateur déjà interprétées et d'effectuer les traitements nécessaires. Si c'est une commande pour jouer ou arrêter une chanson, il fera suivre la demande au Méta-serveur. Sinon, enverra la réponse correspondante au client.

8 MÉTA-SERVEUR

Ce module sera chargé de maintenir une base de données actualisée des chansons existantes dans les serveurs musicaux.

Pour communiquer avec eux je vais utiliser le framework IceStorm. Ce module va s'inscrire à un service de messagerie IceStorm pour recevoir les mises-à-jour des fichiers musicaux dans les serveurs distants. De leur côté, ces derniers seront inscrits à IceStorm en tant qu'émetteurs.

Il 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 et ensuite basculer vers une BDD SQLite.

Gestion de la charge des serveurs :

Une chanson pourra exister dans plusieurs serveurs, elle sera identifié par son nom et sera associée à la liste de serveurs qui la contiennent. Si un client demande une chanson existante dans plusieurs serveurs, on enverra cette demande au premier serveur dans la liste, puis on déplacera ce serveur à la dernière place afin d'équilibrer la charge.

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.

- Côté serveur de fichiers :
  - enregistrerServeur(string ipServeur) : int
    Permettra à un serveur de s'ajouter à la liste de serveurs de fichiers disponibles.
  - supprimerServeur(int idServeur)
    Permettra à un serveur de musique de se désinscrire de la liste. Toutes les chansons qui sont uniquement contenues dans le serveur partant seront supprimées.

9 SERVEURS MUSICAUX

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

Afin de simplifier la gestion de fichiers, je vais implémenter un processus en arrière plan qui va inspecter un répertoire spécifique. Les fichiers MP3 existant dans ce répertoire seront ceux que le serveur mettra à disposition des clients. À chaque fois qu'un fichier sera ajouté ou supprimé, le thread en arrière plan va notifier le serveur IceStorm. Cette notification arrivera ensuite au serveur central puisqu'il sera aussi abonné au service de messagerie.

Il y aura en plus une application Python en console qui proposera à l'administrateur quelques options pour s'inscrire/désinscrire du serveur central et configurer le répertoire de MP3s. On aura aussi accès au log sur les notifications envoyées afin de détecter les possibles erreurs.

10 LIENS UTILES

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