Grammaire en couleur interactive pour l'apprentissage du français sur Android

Projet NTIC

Etudiants : JIMENEZ Rafael et AHMED Kamran
Responsables du projet : NERIMA Luka et NEBHI Kamel

Table des matières

1 Présentation

1.1 Introduction

Dans le cadre du module de NTIC lors du 2ème semestre nous avons dû choisir un projet. Celui-ci a pour but de transporter la web-application FipsColor réalisée par Kamel Nebhi sous Android.

« L'analyseur multilingue `FiPS` permet de transformer une phrase en une structure syntaxique riche et accompagnée d'informaiton lexicales, grammaticales et thématiques. Notre application fonctionne sur l'adaptation des structures en constituants de `FiPS` à une nomenclature grammaticale simplifiée.

Cette web-application est directement destinée au monde de l'enseignement. Le but de FipsColor est de reprendre les principe pédagogiques de la grammaire en couleur (Plan d'étude et Maitrise du français - Corome Suisse Romande) selon lesquels toute catégorie lexicale et fonciton de constituant seront représentés par une couleur : tout cela afin de faciliter l'acquisition de connaissances chez l'enfant. »

FIGURE 1.1: Application web FipsColor

1.2 Outils informatiques utilisés

- Plateforme : Ubuntu 11.04 Natty Narhal / Windows 7
- Environnement de développement intégré : Eclipse-Helios
- Plugin pour Eclipse : Android Development Tools 11.0.0
- SDK : Android SDK r11
- Android Plateform Version : Android 2.1 (API Level 7)

2 Architecture logicielle

Ce chapitre est axé sur les aspects techniques internes de la plate-forme. La *figure 2.1* illustre les principaux composants du système d'exploitation Android. Les points importants des différentes couches sont détaillés ci-dessous en parcourant les couches par ordre décroissant d'abstraction, soit de la couche "noyaux" à la couche "application".

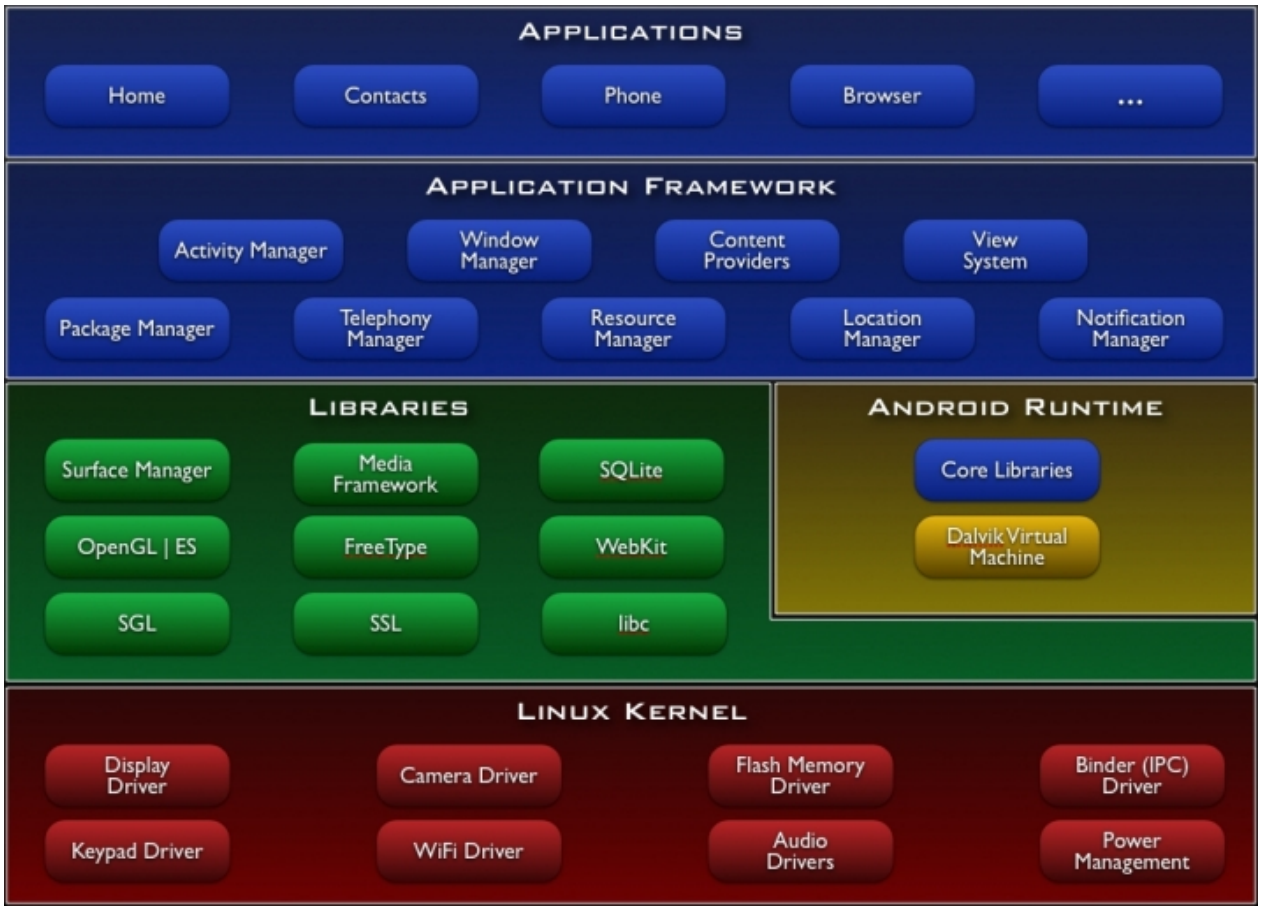

FIGURE 2.1: Architecture d'Android

2.1 Linux, le noyau (*Kernel*)

Android est un framework [1] de développement qui utilise un noyau **Linux** et **java** comme langage de programmation. Le noyau Linux utilisé est le 2.6. Comme dans tous les *OS*, le noyau a pour but de

1. Ensemble de bibliothèques et de conventions permettant le développement rapide d'application

gérer le matériel à l'aide des mémoire, des couches réseaux basses, des drivers ou encore des processus. Le noyau agit également comme une couche d'abstraction entre le matériel et le reste de la pile logicielle.

2.2 Les libraires

Android inclut un ensemble de bibliothèques C/C++ utilisées par les divers composants. Certaines bibliothèques de base sont citée ci-dessous.

Il ne manque de rien au niveau **média**, Android peut lire les formats d'images *PNG* et *JPG*, il est possible d'encoder et de décoder des flux audio et vidéo aux formats *AAC*, *H.264*, *MP3*, *MPEG-4* ou *Ogg Vorbis*

En ce qui concerne le **rendu graphique**, il y a également diverses bibliothèques graphiques permettant l'affichage des polices de caractère ou encore le rendu 2D et 3D. Android a mis en oeuvre une implémentation *OpenGL ES*[2] *1.0*

Ce système d'exploitation, permet l'utilisation de **bases de données *SQLite***. Comme son nom l'indique elle est très légère. Sa particularité est qu'elle est directement intégrée aux programmes. La totalité de la base de données est stokée dans un fichier indépendant.

2.3 Framework

C'est à ce niveau de l'architecture qu'il faut travailler pour faire ses propres applications. Les programmeurs ont accès aux mêmes *API*[3] que celles utilisées par les applications de base d'*Android*. En fournissant une plateforme de développement ouverte, Android offre aux développeurs la possibilité de construire des applications innovantes et extrêmement riche.

2.4 Applications

Android est livré avec un ensemble d'application de base, dont un client de messagerie, une application pour lire et écrire des *SMS*, calendrier, cartes, navigateur, les contacts et autre. Toutes les applications sont écrites en utilisant le langage de programmation **Java**.

2. Embaddes Systems
3. Application Programming Interface / ensemble de fonctions, procédures ou classes mises à disposition par une bibliothèque logicielle

3 Projet

Ce chapitre montre comment le projet à été réalisé.

3.1 Cas d'utilisation

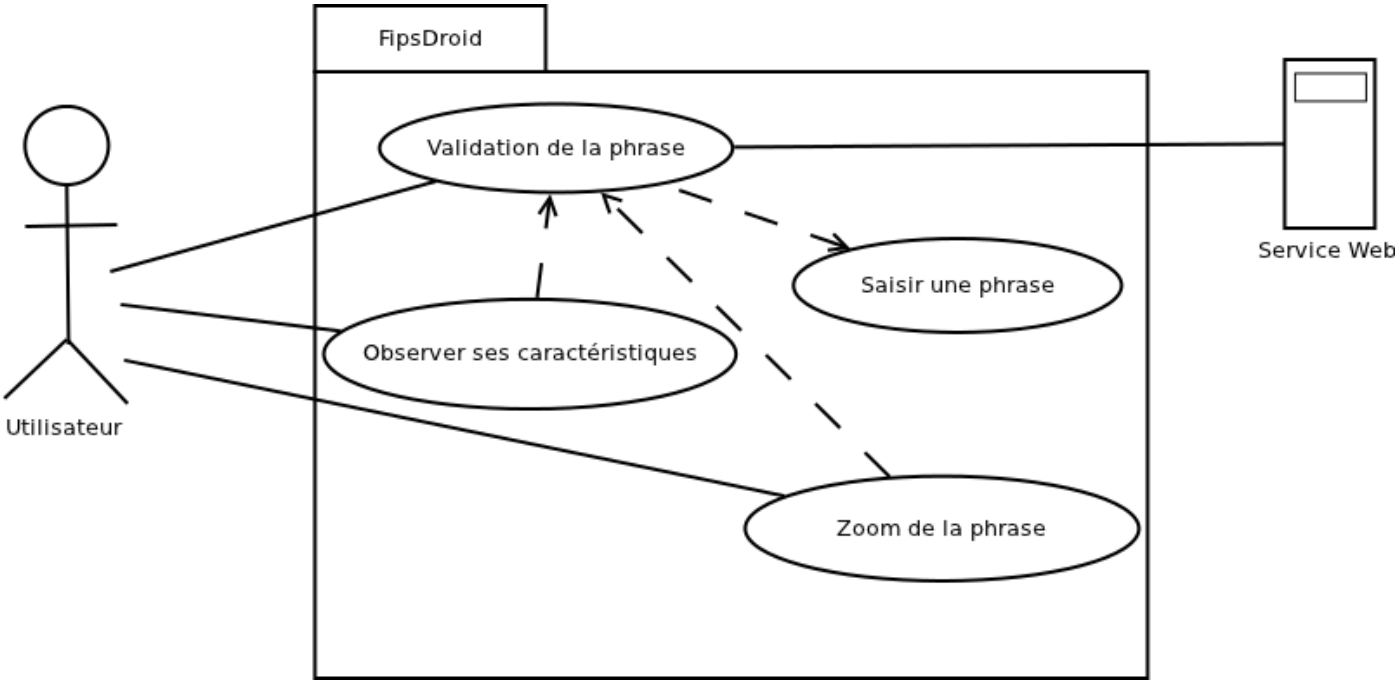

FIGURE 3.1: Diagramme des cas d'utilisations

3.2 L'interface

Dans cette partie du projet nous avons passé un certain temp à réfléchir à une interface ergonomique avec un affichage clair. Voici l'une des nombreuses ébauches :

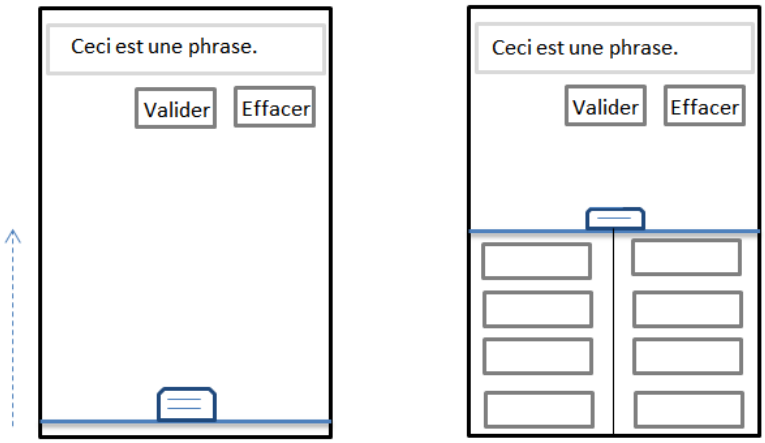

FIGURE 3.2: Une des ébauches

Cette ébauche consistait à avoir un menu rétractable avec deux 'scrollview' pour afficher tous les boutons. Le problème avec cet interface était qu'une fois que le menu était affiché il prenait énormément de place et donc l'utilisateur ne pouvait pas écrire de longues phrases de plus le zoom aurait été très limité.

Pour parer tous ces problèmes nous sommes passés à cet interface : Nous avons remplacé le menu

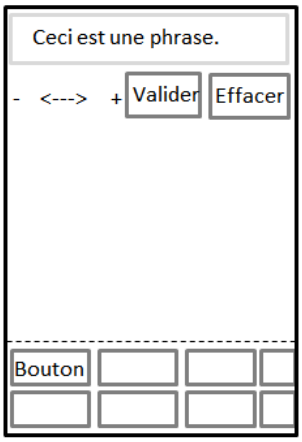

FIGURE 3.3: L'interface choisie

rétractable par deux scrollview horizontal qui permettent d'augmenter l'espace du texte et surtout permettent d'avoir accès à tous les boutons à n'importe quel moment.

3.3 Les Boutons

Pour sélectionner ou déselectionner les différentes catégorie ou fonction grammatical, nous utilisons des *ToggleButton* qui peuvent être comparé à un interrupteur.

3.3.1 Personnalisation

Il existe déjà des *ToggleButton* par défaut. Cependant, comme nous avons besoin de couleurs différentes pour chaque bouton, nous avons dû personnaliser leurs aspects. Pour ce faire, il faut d'abord définir les deux états (checked / not checked) et ceci est définit pour chacun des boutons dans le dossier drawable.

Le début du nom du fichier correspond à la catégorie et la fin des noms des fichiers correspond à :

- **checked** : représente l'état sélectionné
- **not_checked** : représente l'état déselectionné
- **back** : les deux états précédents réunis qui représente le bouton dans tous ces états

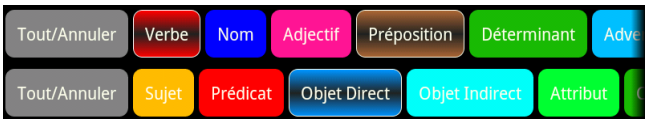

FIGURE 3.4: Boutons personnalisés

3.4 Communication avec le service web

Pour obtenir les informations grammaticale du texte saisi il faut dialoguer avec un service web. Tout d'abord nous pensions qu'il fallait utiliser le protocole soap pour transmettre un message au service. Alors, qu'en vrai une simple requête HTTP était suffisante.

3.4.1 Requête HTTP

La requête HTTP est de la forme suivante :

`http://129.194.19.89/Parser?ap=XMLTei&ln=fr&in=phrase%20saisie`

- L'attribut **ap** correspond au format de sortie.
- L'attribut **ln** correspond à la langue du texte saisi.
- L'attribut **in** correspond tout simplement au texte saisi.

La requête s'effectue grâce à la classe ***ContainerData.java*** qui a une seule méthode qui une fois parsé le fichier retourne une liste contenant tous les termes avec leur spécification.

3.5 Le "parsage" du fichier xml

Une fois la requête transmise, le service web nous retourne un fichier xml dans ce genre :

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
<teiHeader> </teiHeader>
<text>
<body>
<div type="analyse">
<s xml:lang="french">
  <phr type="DP" function="SUBJ">
    <w type="PRONOM-PERSONNEL_1_SIN-ING" lemma="je">Je</w>
  </phr>
  <phr type="" function="Predicate">
    <w type="VERBE-IND-PRE_1_SIN" lemma="regarder">regarde</w>
  </phr>
  <phr type="DP" function="OBJ">
    <w type="DETERMINANT-DEFINI_SIN_MAS" lemma="le">le</w>
    <w type="NOM-COMMUN_SIN_MAS" lemma="chat">chat</w>
  <phr type="DP" function="SUBJ">
    <w type="PRONOM-RELATIF_SIN_MAS" lemma="qui">qui</w>
  </phr>
  <phr type="DP" function="SUBJ">
  </phr>
  <phr type="" function="Predicate">
    <w type="VERBE-IND-SUB-PRE_3_SIN" lemma="manger">mange</w>
  </phr>
  <phr type="DP" function="OBJ">
    <w type="DETERMINANT-INDEFINI_SIN_FEM" lemma="un">une</w>
    <w type="NOM-COMMUN_SIN_FEM" lemma="pomme">pomme</w>
  </phr>
  </phr>

</s>
</div>
</body>
</text>
</TEI>
```

Nous nous intéressons uniquement aux balises :

- <**w**> (word) : correspond à un mot grammatical qui a comme attribut :
  - **type** : qui caractérise l'élément en utilisant la partie du discours appropriée. Correspond à la couleur du mot.
  - **lemma** : qui fournit le lemme du mot.
- <**phr**> : représente un syntagme grammatical qui a comme attribut :
  - **function** : caractérise la fonction grammatical du segment. Correspond à la partie soulignée

3.6 Le Texte

Une fois notre saisie validée il est temp d'afficher le texte. Il existe déjà une classe (*TextView*) qui permet d'afficher du texte et même de le colorer de la couleur de notre choix. Cependant, pour notre

application, nous devons également souligner du texte. C'est pour cette raison que nous avons dû créer notre propre classe qui s'appelle ***UnderlineTextView.java*** et qui hérite de *TextView*.

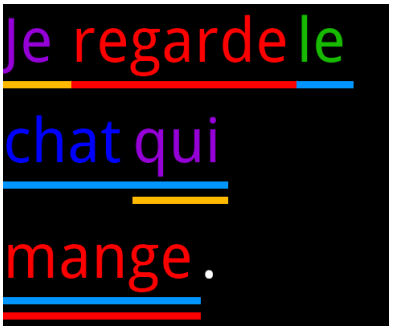

FIGURE 3.5: Texte affiché

3.6.1 Zone de texte

Tout d'abord il est important de savoir que les couleurs de chaque catégorie lexical ou grammatical sont stocké dans le fichier **res/values/colors.xml**. Il suffit de changé les valeurs dans ce xml ce qui va engendrer un changement complet dans l'application, de la couleur des bouton à la couleur du texte en passant par la couleur des soulignements.

3.6.2 Catégorie lexical

La catégorie grammatical (Nom, Verbe, Adjectif, Préposition, Déterminant, Adverbe, Conjonction, Pronom) est représentée grâce à la couleur du terme. Le texte est donc coloré à l'aide des *toggles lexicaux* qui utilisent le listener *lexicalListner*. En appuyant sur l'un des *toggle*, il va colorer ou enlever la couleur de la catégorie sélectionnée.

3.6.3 Soulignement (catégorie grammatical)

UnerlineTextview contient un objet *feed* qui représente les caractéristiques d'un terme. A l'aide de ces informations nous allons savoir si il faut ou non souligner un terme et de quel couleur.

Pour le soulignement nous avons dû redéfinir la méthode ***onDraw(Canva canva)***. C'est dans cette méthode que le texte est dessiné. Le soulignement est dessiné à l'aide d'un rectangle. Le texte est affiché dans une zone qui s'appelle canvas. Pour que le soulignement soit visible nous avons dû agrandir cette zone de dessin en redéfinissant la méthode ***OnMesure(int widthMeasureSpec, int heightMesuresSpec)***. Pour ce qui est de la hauteur nous avons ajouté l'écart nécessaire qui représente la taille des soulignement et nous avons également ajouté un écart à la largeur ce qui nous permet de voir un espace entre les termes.

3.6.4 Positionnement des termes

Le positionnement des termes se fait dans la class ***FipsDroidActivity*** à l'aide de la méthode calibre(). Dans notre interface graphique nous avons une zone réservée à l'affichage du texte. Cette zone est représentée par un layout vertical ayant comme id : linearLayoutPrincipal. Chaque terme est représenté par un objet UnderlineTextView. Pour que les termes soient côte à côte nous les ajoutons donc dans un layout horizontal créé à l'aide de la méthode newLinearLayout(). Dès qu'un des éléments dépasse l'écran nous l'ajoutons dans un nouveau layout horizontal qui lui même est ajouté au layout principal.

3.6.5 Le Zoom

A l'aide de la zone de zoom (- +) nous pouvons agrandir ou rétrécir le texte juste en déplacent notre doigt sur cette zone. Une fois le doigt pressé si nous nous déplaçons sur la droite, le texte va s'agrandir. Le déplacement est calculé à l'aide des coordonnées de l'écran. Après chaque changement on appelle la fonction ***calibre()*** qui va nous ajuster correctement le texte à l'écran.

4 Conclusion

4.1 Travail effectué

L'application a une seule et unique vue. Dans celle-ci l'utilisateur est amené à saisir une phrase. A ce moment il a la possibilité entre effacer la phrase ou de la valider. En la validant, un service web va être interrogé à l'aide d'une requête HTTP. La phrase saisie est donc affichée à l'écran et l'utilisateur peu donc commencer à regarder les différentes caractéristiques grammaticales en appuyant sur les toggles dans les deux menus déroulant du bas. Il y a également une option zoom qui permet d'agrandir ou de rétrécir le texte. Cette application a été conçue pour être la plus simple d'utilisation possible et avec un affichage clair. L'application peut être orienté verticalement ou horizontalement. Des test ont montré que l'application était compatible également avec des tablettes tactiles.

FIGURE 4.1: Vue principale

4.2 Evolutions pouvant être apportées

Quelques améliorations sont possibles. Notamment, il est envisageable de créer une assistance pour aider les utilisateurs à comprendre les caractéristiques grammaticales. Une seconde amélioration serait de pouvoir choisir une autre langue que le français, car le service web le gère déjà. Et la dernière amélioration serait de remplacer les boites de dialogues qui peuvent cacher le texte par un affichage plus simple.