Rapport sur le projet : TWiC sur Android

Claudio Pereira

27 mai 2011

1 Le but du projet

Le but du projet est d'implémenter TWiC sur le système d'exploitation mobile Android. L'application sera une application dite *standalone*. Cette application utilise le SDK officiel du système Android. L'utilisateur saisira une phrase et sélectionnera le mot a traduire. Il pourra ainsi demander une traduction au serveur du LATL ainsi qu'une traduction au moteur de traduction de Google. Il pourra aussi avancer mot à mot dans une phrase.

2 Diagramme des cas d'utilisations

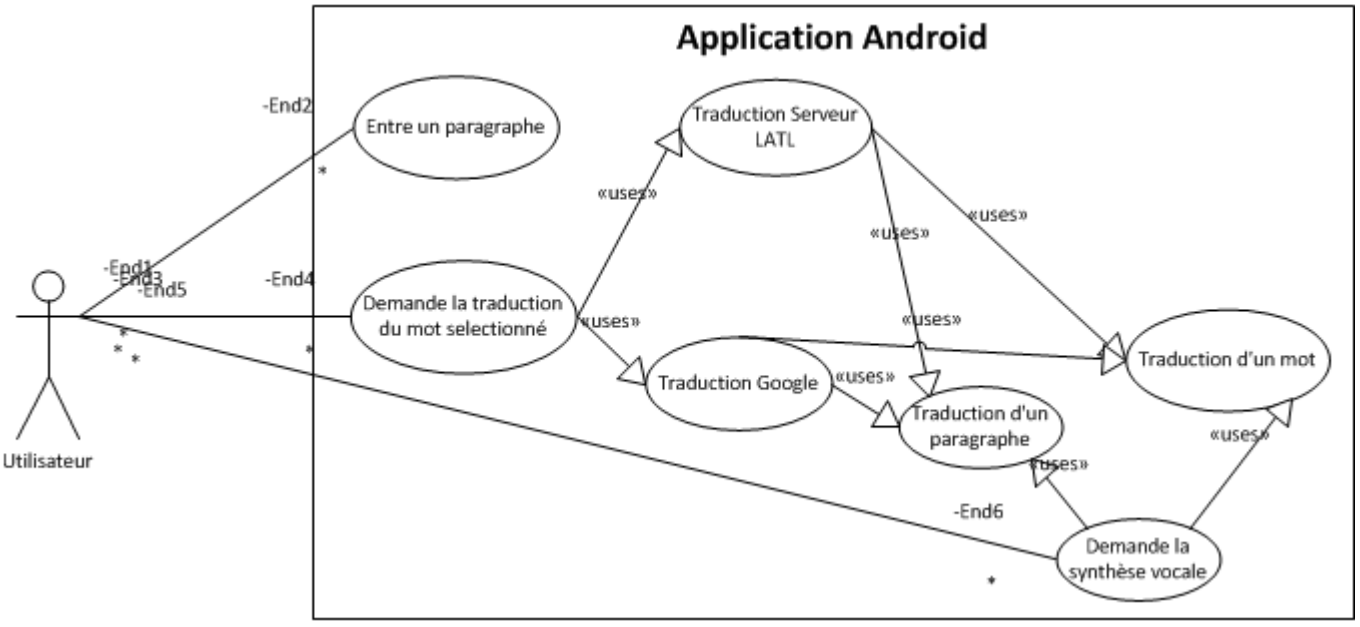

FIGURE 1 – Le diagramme UML des cas d'utilisations

3 Déroulement du travail

Comme décrit précédemment, le projet se passe sur la plateforme Android. Il faut donc d'abord se préparer à d'abord se procurer les outils de développement.

3.1 Installer le SDK Android

Pour travailler sur la plateforme, il n'y a aucune contrainte au niveau du système d'exploitation contrairement au SDK pour l'iOS d'Apple. Le SDK est donc disponible pour les plateformes Windows, Mac OS X et Linux. Pour télécharger le SDK, il faut se rendre à l'adresse `http://developer.android.com/sdk/index.html`. En téléchargeant le SDK, vous installerez les principaux composants pour le développement. Il faut savoir que la plateforme Android souffre d'une grande fragmentation. Il existe actuellement 9 versions du système Android. Pour chacune des plateformes, il y a un SDK associé. La dernière version des outils du SDK est la révision 11. Il faut savoir que en installant le SDK, vous installer aucune version des outils du SDK. Il faut lancer le programme **SDK Manager.exe**. Ce programme se trouve à la racine du répertoire. Ce programme est le programme qui permet d'installer les outils. Il suffit de sélectionner **Available packages**. Il faut ensuite sélectionner **Android Repository**. Dans ce menu se trouvent les différents SDK pour les différentes plateformes. Le langage utilisé par le SDK Android est le Java.

Il faut savoir qu'une étude à montré que plus de 50% des utilisateurs de smartphone Android sont au moins dans la version 2.1 du système. Les versions inférieures du système tendent à disparaitre. Il est peut être intéressant d'installer les SDK à partir de la version 2.1 . Il faut encore noté que la version 3 du système ne concerne pas les smartphone mais les tablettes. Google a annoncé sont intention d'unifier à nouveau les SDK pour smartphone et tablettes à partir de la version 3.3 du système.

Pour facilité le développement d'application Android, Google a mit en place un plugin pour l'environnement de développement Eclipse. Sur le lien `http://developer.android.com/sdk/eclipse-adt.html` se trouve toutes les informations pour l'installation du plugin.

Maintenant que le SDK est installé, il est temps de commencer le projet.

3.2 Le début du projet

L'environnement de développement utilisé est Eclipse ainsi que le plugin ADT. En installant le plugin, une nouvelle option apparait pour la création d'un projet Android. Une nouvelle fenêtre apparait et l'image 3.2 montre à quoi elle ressemble. Pour la création d'un projet il faut spécifier un nom de projet. Il faut également définir sur quelle plateforme l'application va tourner. Il est possible de spécifier plusieurs plateformes.

Pour finir il faut spécifier

Le nom de l'application : Le nom de l'application est le nom sous lequel l'application sera présente sur le système Android.

Le nom du paquetage : Le nom du paquetage est doit être composer de deux identifieurs comme par exemple ch.unige.

La version minimale du SDK : La version minimale est la version minimale du système qui peut exécuter l'application. Si plusieurs systèmes ont été sélectionnés la version la plus basse sera la version minimale.

Dans le cadre du projet les valeurs utilisées sont

Le nom de l'application : Twic

Le nom du paquetage : ch.unige.ntic.twic

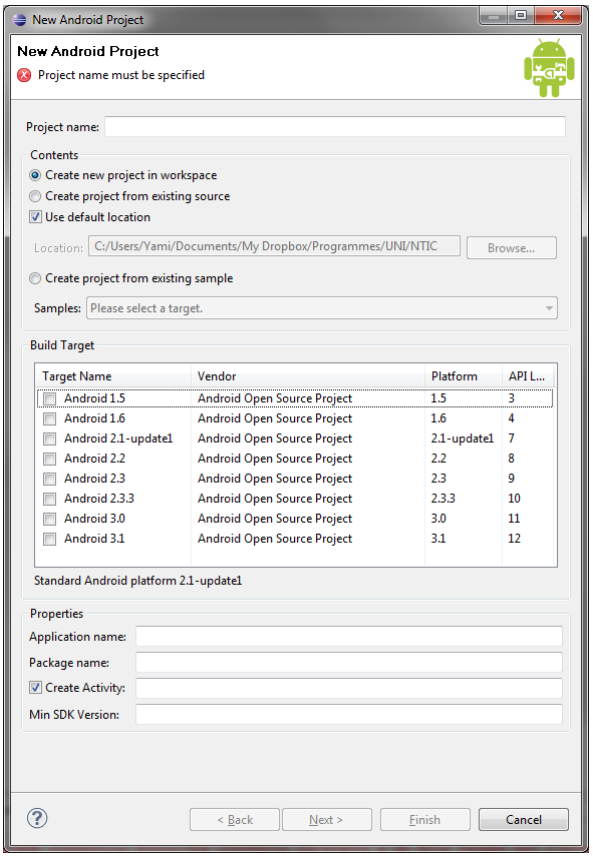

FIGURE 2 – Fenêtre de création d'un nouveau projet Android

La version minimale du SDK : La version minimale est la version 7 qui correspond au système Android 2.1.

Une fois ceci terminé, le projet est enfin créé. On remarque alors que le projet contient déjà un bon nombre des choses. Il contient les dossiers suivant

- Le dossier **src** qui contient les sources de l'application.
- Le dossier **gen** qui contient la classe **R** qui référence les ressources du projet.
- Le dossier **assets**
- Le dossier **res** qui contient les ressources du programme.

Le dossier src

Le dossier **src** est le dossier contenant l'ensemble des classes du projets. Il est celui qui est le plus utilisé.

Le dossier gen

Le dossier **gen** est un dossier générer automatiquement par le plugin Eclipse lors de modifications dans le dossier **res**. Il contient la classe **R**. Cette classe est une classe ne contenant que les adresses des éléments présent dans le dossier **res**. Cette classe permet d'utiliser les éléments présent dans le dossier **res** dans le code Java. Cette classe ne tolère aucune modification manuelle. Il est possible de générer cette classe si l'on décide de ne pas utiliser l'environnement de développement Eclipse.

Le dossier assets

Les éléments de type "assets" sont des ressources qui ne seront pas référencés par la classe R. Les éléments présents dans ce dossier ne subiront aucune transformation lors de leur copie sur le terminal Android. Pour accéder au éléments de ce dossier il faut utiliser un **AssetManager** qui en lui soumettant le nom du fichier dans la méthode *open* retournera un **InputStream** sur le fichier. Il est possible d'appeler le **AssetManager** en le demandant au **Context** avec *Context.getAssets()*.

Le dossier res

Le dossier **res** est le dossier contenant les ressources qui seront utilisées par l'application. Les différentes ressources sont disponibles dans des dossiers selon leur types. Ces ressources seront aussi référencées dans la classe **R**. Il sera possible alors de les appeler grâce a leur identifiant dans le code Java et ainsi les utiliser. Ces ressources sont typés. Il est donc important de bien ranger chaque ressource au bon endroit. Ces ressources sont ensuite transformées en objets binaire lors de la création du fichier .apk qui contient l'application. Lors de la génération du projet sous Eclipse, le plugin crée trois dossiers pour trois types de ressources dans le dossier **res**.

drawable : Contient toutes les images jpg et png du projet. Le plugin crée trois dossiers drawable. Ces dossiers servent à séparer les images selon leur résolution. Le système Android choisira lui même l'image à utiliser selon la résolution du terminal.

layout : Contient les fichiers de disposition de l'interface utilisateur de l'application. Le format des fichiers utilisé est le XML.

values : Contient les fichiers à contenu textuel contenu dans des fichiers XML. Le contenu peut être stocké dans un seul fichier ou bien dans plusieurs. Les conventions Android indique qu'il est préférable de catégoriser les fichiers XML selon leur contenu. Par exemple les chaines de caractères du programme sont stockées dans le ficher **string.xml**, les tableaux dans **arrays.xml**. Il est également a noté qu'il est possible de permettre la traduction de son application en déclarant un dossier **values-fr** par exemple pour le français. En effet si la langue locale du téléphone est le français, alors ce n'est pas le dossier **values** mais le dossier **values-fr** qui fournira les ressources textuelles au programme.

De base Eclipse ne crée que ces trois dossiers mais il est possible d'en créer encore trois autres.

anim : Contient toutes les animations du projet au format XML.

menu : Contient tous les menus , menus d'options qui apparaissent sur pression du bouton **MENU** et les menus contextuels qui apparaissent après une pression longue sur l'écran, au format XML.

xml : Contient tous les fichiers XML qui ne sont pas utilisé par le système Android. Ces fichiers sont traiter dans le code au travers de l'objet **XmlResourcesParser** obtenu par la méthode **Resources.getXml(int id)**. L'objet **Resources** est récupéré à partir de l'objet **Context**.

Avant d'attaquer le projet lui-même, il faut encore parler d'un fichier important le **AndroidManifest.xml**

L'AndroidManifest.xml

Le fichier **AndroidManifest.xml** est un petit fichier qui contient des informations sur l'application. Ce fichier contient des informations pour la configuration de l'application. Il est possible de modifier a partir de l'interface du plugin d'Eclipse. Il est cependant souvent plus simple d'éditer directement le ficher XML directement. Il est possible de faire une application sans jamais modifier le fichier manifest mais cependant la plus part du temps il faut quand même légèrement modifier.

Il est maintenant temps de commencer le projet. Les prochaines sections détailleront le fonctionnement de l'application et les choix pour l'interface homme-machine.

4 L'application

L'application TWiC est une application dans laquelle la navigation entre les différentes partie se fait à l'aide d'onglets. La déclaration de l'interface peut se faire de deux façons différentes dans le developpement d'une application Android. La première méthode est déconseillée dans le cadre de gros projets. Cette méthode consiste à créer tous les éléments dans le code Java. Il est possible de procéder de la sorte pour un petit projet. La deuxième méthode consiste à créer un fichier XML qui contiendra tous les éléments de l'interface. Cette méthode est celle utilisée dans ce projet. Les fichiers XML d'interface sont dans le dossier **res/layout**. Lors de la création du projet, un fichier **main.xml** est créé. Il est tout à fait possible de le modifier, dans ce projet il a été modifié.

Le fichier **main.xml** a été modifié pour convenir au choix de l'interface choisie. Il faut savoir que lors de la création de l'interface graphique à l'aide des fichiers XML, un éditeur graphique est présent pour aider à la construction de l'interface. Il faut malheureusement savoir que pour la création d'une application contenant des onglets, il n'est pas possible d'utiliser l'éditeur graphique, il faut donc obligatoirement modifier le fichier XML à la main.

Il faut savoir que ce soit pour la création d'une application iPhone qui possède des onglets pour la navigation ou bien une application Android qui possède également des onglets pour la navigation n'est pas facile à réaliser. Mais une fois que le template à utilisé a été trouvé, les choses deviennent nettement plus facile.

Tout d'abord il faut modifier le fichier **main.xml** et le modifier comme l'exemple montré. Contrairement à la plus part des fichiers XML pour la création d'interface, ce fichier doit commencer par une balise **TabHost** qui est donc l'hôte des onglets. Il est possible ensuite de placer le reste des éléments dans une disposition très classique à l'aide du **LinearLayout**. Le **LinearLayout** est une des disposition de base la plus utilisée. Il permet d'aligner simplement les éléments qu'elle contient que se soit à l'horizontale ou à la verticale. Il faut ensuite ajouter la barre d'onglets à l'aide de l'élément **TabWidget**. Il faut ensuite déclarer une zone dans laquelle se trouvera les éléments propres à chaque onglet. Il faut utiliser un élément **FrameLayout** qui permet d'assigner toute une zone pour afficher un élément. Dans le cadre de se projet, les éléments qui seront contenus dans le **FrameLayout** seront les éléments de chaque onglets. Chaque onglet chargera un fichier XML qui contiendra les éléments à afficher.

Fichier xml 4.1

```
   <?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
```

```
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```

Le code XML pour mettre en place une application contenant des onglets est maintenant terminé. Il faut maintenant aller jeter un coup d'œil au niveau de la classe Java qui a été crée en même temps que le projet.

Le classe principale du projet est la classe **Twic**. En ouvrant cette classe , on constate qu'elle étend une **Activity**. Une **Activity** est un composant principal d'une application. C'est également le composant qui porte les éléments à la vue de l'utilisateur. Une application n'est pas limitée à ne posséder qu'une seule activité. Il est possible d'en avoir autant que l'on veut. Les **Activities** sont capable de communiquer entre elles a l'aide d'objet d'intention les **Intents**. Une activité peut en réveiller une autre activité en la nommant directement. C'est le mode explicite. Il est également possible de réveiller une activité non pas en la nommant mais en listant la liste des caractéristiques de l'activité. À ce moment la, le système Android cherche une activité possédant les caractéristiques demandées. Il exécute alors l'activité qui correspond à la demande. Si plusieurs activités peuvent répondre à la demande envoyée alors le système demande a l'utilisateur de choisir entre les activités possibles. Il faut savoir que si on décide de faire une application qui est composée d'une liste dans laquelle on sélectionne un élément et on avance a travers les vues successivement, il ne faut pas que la classe principale de l'application étende la classe **Activity** mais **ListActivity**. Dans le cas de **Twic**, la navigation se fait à l'aide d'onglets. Il faut donc que la classe **Twic** étende la classe **TabActivity**.

Une méthode est présente, c'est la méthode **onCreate(Bundle savedInstanceState)**. Cette méthode est automatiquement lancée à l'initialisation de l'application. Une application Android suit un cycle de vie bien précis.

Il faut savoir que seul l'implémentation de la méthode **onCreate()** est obligatoire. Il est possible d'implémenter autant de méthode du cycle de vie que l'on souhaite.

Le cycle de vie d'une application Android est composée des nombreuses méthodes.

onCreate() : Cette méthode est exécutée quand l'utilisateur clique sur l'icône de l'application pour la première fois. Elle est utilisée pour l'initialisation des vues ainsi que des fichiers ou données temporaires.

onRestart() : Cette méthode est exécutée lorsque l'activité a été arrêtée et qu'elle est redémarrée. Ceci signifie qu'elle revient au premier plan.

onStart() : Cette méthode est exécutée après chaque **onCreate()** ou **onRestart()**. Elle charge les données sauvegardée durant le dernier arrêt.

onResume() : Cette méthode est exécutée après chaque **onStart()** et lorsque l'activité passe au premier plan de l'activité. Cette méthode permet l'initialisation d'une connexion à une base de donnée (sqlite) et met à jour des données qui aurait pu être modifié avant son appel.

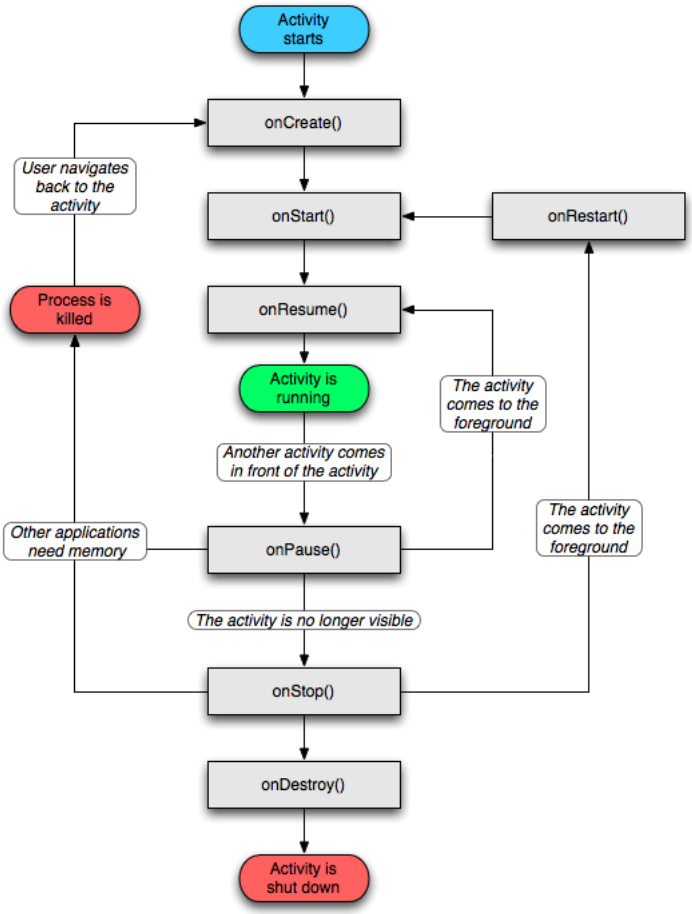

FIGURE 3 – Cycle de vie d'une application Android

onPause() : Cette méthode est exécutée avant la méthode **onStop()**, à chaque fois que l'utilisateur change d'activité, lorsque l'utilisateur demande un arrêt de l'activité ou bien lorsque le système Android a besoin de libérer de la mémoire. Cette méthode sauvegarde des données qui seront perdues après l'arrêt si elles ne sont pas sauvegardées et termine la connexion à la base de donnée. Il faut que cette méthode soit rapide à s'exécuter car la prochaine activité ne sera pas lancée avant l'arrêt de la méthode.

onStop() : Cette méthode est exécutée avant chaque mise en veille du système et avant la méthode **onDestroy()**. Elle permet la libération des ressources.

onDestroy() : Cette méthode est exécutée après lors de l'arrêt de l'activité. Si cette méthode a été appelée, il faudra exécuter la méthode **onCreate()** lors du prochain lancement de l'application. Cette méthode permet la libération des ressources et des fichiers temporaires.

Les méthodes **onCreate()** et **onDestroy()** sont les seules à être implémentées dans ce projet.

La méthode **onCreate()** de la classe **Twic** ne contient que 2 lignes de code. La première est l'appel à la méthode **onCreate** de la classe mère et la seconde méthode sert à définir l'apparence de l'application. En effet la méthode **setContentView(R.layout.main)** permet de définir l'interface graphique d'une activité à l'aide d'un fichier XML. Pour spécifier le fichier XML à utiliser, il faut utiliser la classe **R**. Il

faut se souvenir que la classe **R** est la classe qui contient les éléments présents dans le dossier **res**. C'est à présent que la séparation des fichiers dans des dossiers correspondant se montre utile car en essayant d'accéder à un fichier XML qui sert à l'interface il est inutile de chercher le fichier correspondant dans le dossier **values**. Un grand nombre de méthodes du SDK Android acceptent un entier. Il faut savoir que **R.layout.main** retourne la valeur entière de l'emplacement de l'élément. Il n'y a pas donc besoin de se préoccuper avec des type.

L'activité **Twic** actuelle effectue une seul chose. L'application se lance et affiche l'interface. Il est maintenant temps de tester l'application sur le simulateur.

4.1 Création d'un appareil virtuel et test dans le simulateur

Pour utiliser le simulateur il faut tout d'abord créer un appareil virtuel. Il faut lancer le programme **SDK Manager.exe**. C'est le même programme qui permet l'installation des différentes versions du SDK. Il est possible de lancer **SDK Manager.exe** depuis Eclipse lorsque le plugin est installé en cliquant sur l'icône d'un robot Android et d'une boite grise avec une flèche blanche pointant vers le bas. Il faut sélectionner à droite **Virtual devices**. Il faut ensuite cliquer sur **new**. Une nouvelle fenêtre s'ouvre et des champs sont a remplir. Seuls les champs **Name** et **Target** sont importants actuellement. Dans le champ **Name**, il faut indiquer un nom de l'appareil virtuel. Il peut être intéressant de nommer l'appareil virtuel selon le type d'appareil que l'on souhaite émuler. En effet, il est tout à fait possible de simuler un terminal Android existant en précisant les mêmes paramètres que le terminal réel. Le champ **Target** indique la version du système à émuler. L'application **Twic** est supposée fonctionner à partir de la version 2.1 du système Android. Maintenant il suffit de valider tous les choix et voila un terminal virtuel viens d'être créé.

Dans la fenêtre du programme **SDK Manager.exe** apparait le terminal virtuel. Il suffit de le sélectionner et de presser le bouton **Start**. Attention le lancement de l'émulateur peut prendre de longues minutes avant de s'être complètement initialisé. Une fois le terminal virtuel opérationnel, il est temps d'exécuter l'application. Sous Eclipse il suffit de créer une nouvelle configuration de lancement et de choisir le projet **Twic**. Après quelques instants l'application est chargée sur le terminal virtuel et est lancée. Le résultat actuel est que une fenêtre noir apparait à l'écran. Il n'y a aucune trace de la barre d'onglet. Il y a juste une petite barre grise avec le nom **Twic** qui indique que l'application est bien lancée. Tout ceci est normal. Une barre d'onglet a bien été déclarée mais elle n'est pas affichée. La réponse est simple. Aucun élément n'a été ajouté dans le **TabHost**. Le **TabHost** est le contrôleur de la vue. C'est a lui qu'il faut ajouter des vues pour qu'il puisse afficher des onglets. Il existe deux façons de créer les éléments d'une barre d'onglet. La première méthode consiste à créer des fichiers XML et de les inclure dans le **TabHost**. La deuxième méthode est similaire. Il suffit de créer une activité par onglet et de les ajouter dans le **TabHost**. Il y aura en quelque sorte plusieurs programmes dans le même programme. C'est la deuxième solution qui est implémenté dans ce projet. Il n'y a pas de meilleure méthode. Ces deux méthodes ont leurs avantages et inconvénients.

5 Twic et ses onglets

Lorsque l'on décide de faire une application dont la navigation se fait à l'aide d'onglets il est important de savoir ce que chaque onglets doit représenter. L'application *Twic* est composée de cinq onglets.

1. Le premier onglet affiche que du texte. Il a pour but d'informer sur le comportement de **Twic**.
2. Le deuxième onglet est l'onglet principal de **Twic**. C'est dans cet onglet que le contexte est saisi et que la demande de traduction est effectuée.

3. Le troisième onglet affiche le mot à traduire ainsi que sa traduction fournie par le serveur du LATL.
4. Le quatrième onglet est une copie conforme du troisième onglet à la différence que la traduction est fournie cette fois-ci par Google Translate.
5. Le cinquième onglet affiche les informations sur la conception de l'application. C'est l'onglet "à propos".

Il a été décidé d'utiliser une activité par onglet. Il faut donc créer cinq classes qui héritent de la classe **Activity**.

Les cinq classes sont

VueDescription : Cette activité affiche le premier onglet qui décrit l'application.

VueTwic : Cette activité affiche le deuxième onglet qui sert pour la saisi du contexte et du mot à traduire.

VueTraduite : Cette activité affiche le troisième onglet affiche la traduction fournie par le serveur du LATL.

VueGoogle : Cette activité affiche le quatrième onglet affiche la traduction fournie par Google Translate.

VueAPropos : Cette activité affiche le cinquième onglet qui affiche les informations au sujet de de la conception de l'application.

Ces cinq classes possèdent toute une méthode **onCreate()**. Cette classe a le même comportement que la méthode **onCreate()** de la classe **Twic**. Le seul changement va être effectuer au niveau du fichier XML à charger pour l'interface. Si **setContentView(R.layout.main)** n'est pas changé, chaque onglet essayera d'afficher la même chose. Il faut donc créer cinq fichiers XML pour chacune des activités.

Le nom de chaque fichier XML est en minuscule car Android n'aime pas les noms de fichiers en majuscule pour les fichiers XML.

designdescription.xml : Ce fichier contient les informations pour l'interface de l'activité VueDescription.

designtwic.xml : Ce fichier contient les informations pour l'interface de l'activité VueTwic.

designtraduite.xml : Ce fichier contient les informations pour l'interface de l'activité VueTraduite.

designgoogle.xml : Ce fichier contient les informations pour l'interface de l'activité VueGoogle.

designapropos.xml : Ce fichier contient les informations pour l'interface de l'activité VueAPropos.

Tous ces fichiers XML sont à placer dans le dossier **res/layout**. Ces fichiers XML ont été créer à l'aide de l'éditeur graphique du plugin Eclipse.

5.1 L'éditeur graphique

Le fonctionnement de l'éditeur graphique pour les interfaces est très simple. Il suffit de sélectionner un widget dans la palette sur la gauche et de le glisser-déposer sur le rectangle noir. Le résultat est alors afficher immédiatement. Sur le haut de l'éditeur il y a des boutons qui peuvent aider a changer une propriété du widget sélectionné. En sélectionnant un widget dans la vue, il est possible de lui modifier ses propriétés en faisant un clic droit. Par défaut le rendu de l'interface est prévu pour la version 3.0 d'Android. La version 3.0 d'Android est consacré au tablettes tactiles. Les tablettes sont plus grandes que les smartphones donc le rendu ne sera pas exactement le même.

5.2 designdescription.xml

Ce fichier XML fait parti des fichiers XML les plus simple. En résumé l'activité **VueDescription** n'a qu'une seule tache, afficher du texte. Avant de placer les widgets tels que les boutons, il est important de choisir un layout. Le layout le plus simple a utilisé est le **LinearLayout** vu plus tôt. Une petite astuce à utiliser lorsqu'on ne connait pas la résolution de l'écran sur lequel l'application va être utilisé, il faut penser que la hauteur de la vue peut être trop petite pour afficher tous les widgets. Pour éviter ce petit souci, il vaut mieux toujours travailler avec l'objet **ScrollView** comme base. L'objet **ScrollView** se trouve dans le menu **Composite** de la palette de l'éditeur. Il est possible ensuite d'insérer un **LinearLayout**. L'objet **LinearLayout** se trouve dans le menu **Layouts** de la palette de l'éditeur. L'orientation du **LinearLayout** est de base verticale. Donc le but de cette vue est d'afficher que du texte. Il faut donc utiliser des objets **TextView** pour afficher du texte. Il suffit de les glisser dans le **LinearLayout** pour que les **TextView** s'ajoute les uns sous les autres. Dans cette vue cinq **TextView** sont nécessaire.

1. Le premier sert de titre.
2. Le deuxième sert de séparation entre le titre et le premier paragraphe.
3. le troisième sert pour le premier paragraphe.
4. Le quatrième sert de séparation entre le premier paragraphe et le second.
5. le troisième sert pour le second paragraphe.

Pour l'instant cette vue n'affiche que des **TextView** comme texte. Mais il est temps de remédier à ça. Comme expliquer précédemment pour le texte, il ne faut pas le coder en dur. Lorsqu'il faudra traduire l'application il faudra modifier le code et cela va finir par être compliquer à gérer. Il existe un moyen simple de stocker les données textuelles. Il faut les stocker dans un fichier appeler **strings.xml**. Ce fichier est de base générer avec le projet. Il contient normalement déjà une entrée. Son fonctionnement est simple. Il faut entrer un identifiant et le texte associé c'est tout. Ensuite en faisant clic droit sur le premier **TextView** et en sélectionnant **Edit Text..**, la liste des chaines de caractères présentes dans le fichier **strings.xml**. Depuis cette même fenêtre, il est également possible d'ajouter une nouvelle chaine de caractère à la liste. Une fois la chaine choisie, le texte de la **TextView** change automatiquement avec le nouveau contenu. Une nouvelle astuce, pour faire des séparateurs facilement il suffit de créer une chaine de caractères qui aura comme nom vide et comme valeur rien du tout. En définissant le widget avec cette nouvelle valeur aucun texte sera affiché et l'illusion d'un vide apparait. Ceci ne fonctionne pas sur les tablettes,l'espace est trop petit par rapport à la taille de la tablette.

5.3 designtwic.xml

Cette vue est certainement la plus compliquée au niveau des éléments à afficher. Cette vue doit avoir une zone de texte pour la saisie du contexte. Il faut également deux listes déroulantes pour le choix de la langue source du contexte et la langue dans laquelle le mot doit être traduit.Et pour finir il faut trois boutons.

- Le premier permet d'envoyer la demande de traduction.
- Le deuxième permet de traduire le mot suivant dans le contexte
- Le troisième permet de réinitialiser la zone de texte et d'effacer le contenu de **TextView** dynamiques.

Il faut savoir que des **TextView** sont nécessaires. Il servent a indiquer la fonction d'un autre widget. Cette vue est plutôt compacte. Il n'y a pas vraiment de **TextView** qui servent de séparateur. En effet en sélectionnant la zone de texte le clavier virtuel va apparaitre. Le clavier virtuel va donc occuper une grande place dans l'écran. Il a donc été choisi d'essayer au possible de rapprocher les éléments pour qu'ils soient presque toujours visible. Il faut que tous les éléments qui ne sont pas des **TextView** ait

FIGURE 4 – Apparence de l'onglet Description

un identifiant unique. Les zones de textes sont des objets **EditText**. Les listes déroulantes sont des objets **Spinner**. Les boutons sont des objets **Button**.

FIGURE 5 – Apparance de l'onglet Twic

5.4 designtraduite.xml

Cette vue a pour but d'afficher du texte. Une partie de ce texte est statique, une autre partie est dynamique. Pour ce qui est de la partie statique, il faut procéder de la même manière que pour le fichier **designdescritpion.xml**. Pour ce qui est de la partie dynamique , il faut procéder de la manière suivante. Il faut sélectionner le widget qui affiche du texte de façon dynamique. Ensuite faire un clic droit et sélectionner l'option **Edit ID ...** à ce moment la il faut donner un nom distinctif pour

pouvoir retrouver le widget plus tard. Il faut savoir que a chaque fois qu'on élément est ajouté dans l'éditeur, celui-ci ajoute un identifiant a chaque widget de façon automatique. Il se trouve que cette vue possède trois boutons également dans la partie inférieur de la vue. Il est possible de les aligner en utilisant un **LinearLayout** et en le définissant comme horizontal au lieu de vertical. Il est possible d'affecter le texte de la manière que un **TextView**. Il faut que ces trois possèdent un identifiant unique. Ils seront utiliser dans la classe **VueTraduite**. Ces trois boutons serviront pour activer la fonction Text-To-Speech d'Android. Le troisième bouton servira à lancer la traduction du mot suivant dans la phrase.

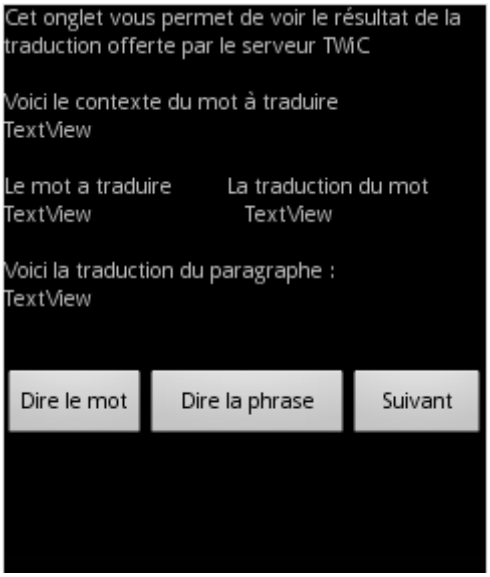

FIGURE 6 – Apparence de l'onglet Traduction

5.5 designgoogle.xml

Ce fichier est une copie du fichier **designtraduite.xml**. Il faut simplement changer le identifiant des objets qui changent de valeur de façon dynamique.

5.6 designapropos.xml

Cette vue est encore plus simple que le fichier **designdescription.xml**. Ici il y a juste quatre **TextView**. Chaque **TextView** affiche une chaine de caractère précise. Toutes ces chaines sont disponibles dans le fichier **strings.xml**. Il y a juste la dernière chaine qui est en réalité un lien vers une page web. Il faut savoir que de base un **TextView** n'interprète pas le langage HTML. Mais suite a une manipulation dans la classe contenant l'activité, il est possible d'interpréter l'HTML et de rendre le lien cliquable. Tout ceci sera expliquer plus loin. Il ne faut donc pas oublier de donner un identifiant unique au widget qui affichera le lien.

5.7 Les classes Java

La classe la plus simple à modifier est **VueDescription**. Cet onglet n'affiche que du texte. La seule chose à modifier dans cette classe est la ligne **setContentView(R.layout.main)**. Avant il n'y avait qu'un seul fichier XML pour les layouts. Mais maintenant tous les fichiers XML de layout sont créés. Il faut donc modifier dans chaque classe la ligne qui permet d'afficher le contenu d'un fichier XML.

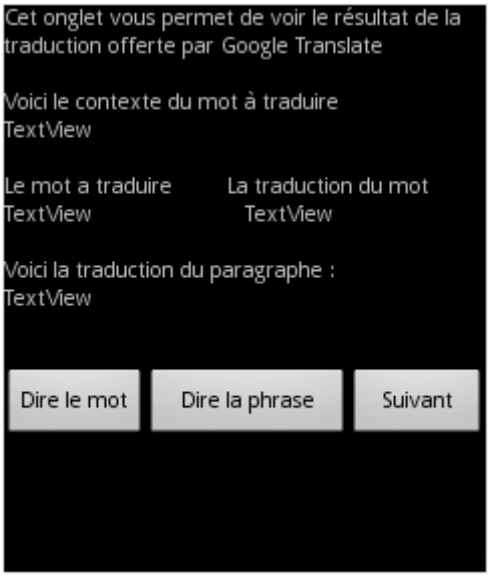

FIGURE 7 – Apparence de l'onglet Trad Google

FIGURE 8 – Apparence de l'onglet A Propos

Donc dans le cas de la classe **VueDescription** il faut modifier **setContentView(R.layout.main)** par **setContentView(R.layout.designapropos)**.

La classe qui demande ensuite le moins de modifications est la classe **VueAPropos**. Cette classe affiche également que du texte. Néanmoins, cette classe affiche un lien vers une page web. Il faut donc traiter le **TextView** qui est sensé afficher le lien. Pour pouvoir travailler sur le **TextView** en question, il est important de lui avoir bien donner un identifiant unique. Une fois que son identifiant est fixé, il est possible d'appeler la méthode **findViewById(int)**. Cette méthode permet toutes les vues déclarées dans les fichiers XML. Il faut simplement lui passer en paramètre l'adresse de la vue. Il faut donc utiliser la classe **R** pour récupérer l'adresse voulue. La classe **R** possède un champ **id**. Dans ce champ

id sont listé tous les identifiants de tous les objets présents dans les fichierx XML. Il faut ensuite caster selon le type de widget utilisé. Par convention lorsqu'on récupère un widget, on le déclare comme étant **final**. Donc dans le cas d'un **TextView**, il faut donc écrire **final TextView lien = (TextView)findViewById(R.id.lien)**. Pour interpréter une chaine de caractère contenant du code HTML, il faut appeler la classe **Html** et appeler la méthode **fromHtml(String)**. Il faut ensuite passer à cette méthode le contenu du **TextView**. Pour récupérer le contenu d'un **TextView**, il faut appeler la méthode **getText()**. Il suffit ensuite de faire un appel à la méthode **toString()** pour avoir une chaine de caractères. Pour insérer du texte dans un **TextView**, il faut appeler la méthode **setText(String)**. Donc au final il suffit d'écrire **lien.setText(Html.fromHtml(lien.getText().toString()))** pour que le code HTML soit interpréter. Malheureusement le lien n'est pas cliquable. Il faut écrire **lien.setMovementMethod(LinkMovementMethod.getInstance())** pour que le lien soit cliquable. Lorsqu'on clique sur le lien, la page est affichée dans une nouvelle vue. Voila l'onglet **A Propos** est terminé.

Les classes **VueGoogle** et **VueTraduite** sont presque identiques. Malheureusement elles dépendent fortement de la classe **VueTwic**. Il faut donc d'abord s'occuper de la classe **VueTwic**.

La classe **VueTwic** est la classe centrale de l'application. C'est elle qui s'occuper de tout le traitement. Pour commencer il faut récupérer les objets **EditText**, **Spinner** et **Button**. Le traitement initial de l'objet **EditText** est simple, il faut simplement initialiser le texte à une chaine vide. Pour insérer ou récupérer du texte dans un **EditText** il faut utiliser les mêmes méthodes que pour un objet **TextView**. Une fois que tous les élements sont tous déclarés, il faut commencer par les boutons. Les boutons sont ceux qui provoquent le fonctionnement du programme. Pour chaque bouton il faut lui associcer un écouteur. Commençons par le bouton le plus simple. Le bouton **Reinitaliser** est un bouton qui permet de vider la zone de texte de son texte et de nettoyer les zones de texte des classes **VueTraduite** et **VueGoogle**. Mais un problème se pose. Il n'est pas possible pour une activité de modifier une autre activité. Par contre il est possible d'envoyer une intention (**Intent**) d'une activité à une autre. Mais un autre problème se pose. Ce mécanisme permet de lancer une autre activité, mais dans ce cas-ci l'activité est déjà lancée. Il faut donc procéder d'une autre façon. Une solution simple et efficace est la création d'une classe **Singleton**. Ce **Singleton** n'autorise qu'une seule instance à la fois. Donc les trois activités partageront le même **Singleton**. Il n'y a pas de soucis de concurrence car une seule activité modifie les valeurs du **Singleton**. Les deux autres activités se contentent de lire les informations du **Singleton**.

Le singleton est donc une nouvelle classe à créer, le constructeur initialise tous les champs de texte avec une chaine vide. Il faut donc stocker dans ce singleton, le mot à traduire, le contexte, la traduction du mot par le serveur du LATL, la traduction du contexte par le serveur du LATL, la traduction par Google Translate et pour finir la traduction du contexte par Google Translate. Il faut ensuite créer les accesseurs et mutateurs. Il ne faut pas oublier de créer la classe qui permet de récupérer l'instance actuelle.

Lorsqu'une valeur du singleton change normalement le changement s'opère sur toutes les vues qui utilisent cette valeur. Par contre il y a un délai avant la mise à jour de la vue. Il faut donc trouver un moyen de forcer la mise à jour immédiate. Il existe une méthode simple qui consiste à envoyer une intention en broadcast. Il faut ensuite créer un filtre d'intention qui réagira sur le fait de recevoir un certain message dans les classes qui doivent capter le message. Il faut donc créer un objet **BroadcastReceiver** et implémenter sa méthode **onReceive()**. Dans cette méthode il suffit juste de vérifier sir le contenu du message délivrer par l'intention est égal au message qui provoque une réaction. Lorsque le message correspond il suffit d'accéder aux **TextView** qui doivent être modifiées et d'affecter la

valeur actuelle du singleton comme texte. Ainsi le changement est immédiat. Il ne faut pas oublier d'enregistrer l'objet **BroadcastReceiver** et de créer un objet **IntentFilter** sur le message à recevoir dans la méthode **registerReceiver()**.

Le bouton le plus simple vient d'être implémenté. Il est temps maintenant de construire le cœur de l'application. Il faut donc s'attaquer au bouton principal de l'application, le bouton **Envoyer**. La fonction du bouton **Envoyer** est simple. Il faut d'abord récupérer le contenu de l'objet **EditText**. Il faut ensuite récupérer le mot qui à été sélectionné. Il faut utiliser deux méthodes qui appartiennent à l'objet **EditText**. Il faut utiliser les méthodes et . Ces méthodes permettent de récupérer la position du curseur. Si un mot a été sélectionné à l'aide de la double pression, les valeurs de retour des deux méthodes est différente. Si le mot n'a pas été sélectionné, la valeur de retour des deux méthodes est identique. Il faut donc détecter le mot. Pour détecter le mot, il faut simplement rechercher les espaces. Il y a aussi deux exceptions, si le début du curseur se trouve à la position 0 alors il suffit de splitter la ligne sur les espaces et récupérer le premier morceau. La deuxième exception est si la position du curseur est à la fin de la ligne, il suffit donc de splitter la ligne et récupérer le dernier morceau.

Obtenir la traduction de la part de Google Translate est assez facile. Il existe un projet qui offre une API pour pouvoir traduire facilement du texte. Dans ce projet cette API a été utilisée. Elle est disponible sur `http://code.google.com/p/google-api-translate-java/`. Elle apporte également une classe contenant les langues que la traduction Google supporte. Cette classe **Language** est utilisée. Elle est utilisée car pour traduire avec cette API, il suffit de faire **translate(lang_from,lang_to,text)**. Il faut savoir que cette classe permet d'afficher la langue comme "French" mais permet également de récupérer la forme courte de la langue du genre "fr". Ceci est très intéressant car en peuplant les quatre langues supportée dans **Twic** dans les listes déroulantes et ensuite de juste récupérer la forme courte pour construire la requête pour le serveur du LATL.

Pour peupler une liste déroulante, il faut créer un objet **ArrayAdapter**. Il suffit ensuite d'ajouter une langue dans l'adaptateur avec la méthode **add()**. Il suffit ensuite d'utiliser la méthode **setAdapter()** de la liste déroulante pour que cette liste affiche le contenu de l'adaptateur.

Pour pouvoir demander une traduction au serveur du LATL. Il faut d'abord construire la requête. Lorsque la requête est construite, il faut créer un objet **URL** avec la requête construite. Il faut ensuite ouvrir une connexion avec le serveur distant. Il faut créer un objet **HttpURLConnection** à l'aide de la méthode **openConnection()** de l'objet **URL**. Il faut ensuite lire les informations présentes dans l'objet **HttpURLConnection** à l'aide des flux d'entrée. Une fois que tout à été récupérer il suffit de rechercher la première occurrence du tag <**translation**> et la première occurrence de </**translation**>. Il suffit de soustraire les deux indices et on peut récupérer la traduction. Cette méthode vaut autant pour la traduction d'un mot que d'une phrase. La seule chose qui change est le tag à rechercher. Il faut pour finir ne pas oublier de fermer la connexion. Ceci achève le traitement du bouton **Envoyer**. Il faut appliquer presque le même traitement pour le bouton **Suivant**. Il faut modifier simplement le morceau de la phrase a traduire.

Il faut maintenant attaquer les deux dernières classes. Ces classes sont identiques. Les seules différences sont les identifiants des **TextView**. Il faut donc récupérer les valeurs du singleton et les insérer dans les **TextView**. Ces deux classes possèdent trois boutons. Les deux premiers servent a activer la fonction Text-To-Speech. La fonction Text-to-Speech est disponible seulement depuis la version 1.6 du système. Il n'y a que cinq langues disponibles,l'anglais, le français, l'allemand, l'espagnol, et l'italien. Pour utiliser la fonction Text-to-Speech, il faut d'abord le déclarer. Il faut ensuite lui spécifier la langue dans laquelle il doit parler. Lorsque la fonction Text-to-Speech n'est plus a utiliser il ne faut pas oublier de lui demander de s'arrêter. Le système Android se chargera ensuite de tuer le processus.

Le premier bouton se charge de demander la synthèse vocale pour le mot uniquement. Le deuxième bouton se charge de demander la synthèse vocale pour le contexte en entier. Le troisième bouton envoie une demande en broadcast pour effectuer la traduction du mot suivant. La classe **VueTwic** reçoit le message et effectue le même traitement que si le bouton **Suivant** de sa propre vu avait été pressé.

Maintenant que toutes les classes sont finies, il faut modifier la classe principale . En effet si on compile actuellement l'application, il n'y aurait aucune modification par rapport au début du projet. C'est normal aucune activité n'a été ajoutée dans le **TabHost**. Pour commencer il faut d'abord récupérer le **TabHost** de l'activité. Ensuite il faut créer une intention qui contient comme classe une des activités. Ensuite il faut créer un objet **TabSpec** qui sert a décrire les informations sur un onglet comme le nom de l'onglet ainsi que l'image à afficher. Il faut ensuite ajouter l'objet **TabSpec** au **TabHost**. Pour une meilleur prise en main de l'application, lorsqu'une demande de traduction est terminée alors l'activité **Twic** reçoit un message en broadcast demandant de changer vers l'onglet contenant la traduction faite pas le serveur du LATL.

Après tout ceci, il reste à modifier le fichier **AndroidManifest.xml**. En effet dans l'état actuel, l'application ne fonctionne pas. Elle ne fonctionne pas car elle ne peut pas se connecter à Internet. Il faut donner la permission dans le manifeste pour pouvoir se connecter. Il existe plusieurs éléments qui doivent avoir l'autorisation dans le manifeste pour pouvoir fonctionner. De cette façon Google sur son Android Market en regardant la liste des permissions peut informer les utilisateurs des potentiels risques d'une application. Il faut également lister les activités que contient l'application ainsi que la liste de leurs actions possibles.

Au final le fichier **AndroidManifest.xml** ressemble à ceci.

Fichier xml 5.1

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ch.unige.ntic.twic" android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <application android:label="@string/app_name" android:icon="@drawable/
        twicbutton"
        android:debuggable="true">
        <activity android:name=".Twic" android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
                <action android:name="change_view" />
            </intent-filter>
        </activity>
        <activity android:name=".VueDescription"></activity>
        <activity android:name=".VueGoogle">
            <action android:name="refresh" />
            <action android:name="remove" />
        </activity>
        <activity android:name=".VueTraduite">
```

```
            <action android:name="refresh" />
            <action android:name="remove" />
        </activity>
        <activity android:name=".VueTwic">
            <action android:name="suivant" />
        </activity>
        <activity android:name=".VueAPropos"></activity>
    </application>
    <uses-permission android:name="android.permission.INTERNET" />

</manifest>
```

6 Tests et Résultats

Les tests ont été réalisé sur le simulateur ainsi que sur le terminal Android Samsung Nexus S avec le système 2.3. Des soucis ont eu lieu au niveau du passage du simulateur au terminal physique. Heureusement en utilisant le debugger et l'ensemble des méthodes de l'objet **Log** de nombreux problèmes ont disparu. L'objet **Log** permet d'envoyer des messages qui peuvent être lu si le debuggage est activé. En utilisant la méthode **e(String tag,String message)** il est possible d'indiquer une erreur. Le tag correspond a un nom pour identifier de qui provient le message. Il est judicieux de choisir comme tag le nom de l'activité. Un problème qui fut rapidement résolu est l'utilisation du Text-To-Speech. En demandant après une exécution de la synthèse d'éteindre la fonction, le système Android éliminait directement l'objet charger de la synthèse vocale provoquant une erreur lors d'une demande de synthèse avec une nouvelle langue. En utilisant le réseau WiFi, l'application est très rapide mais en utilisant la 3g selon la quantité d'informations à transmettre l'application peut sembler être gelée. Une amélioration possible est d'afficher une fenêtre de dialogue pour montrer que l'application n'est pas gelée. Cette fenêtre doit par contre avoir quelques secondes devant elle avant de s'ouvrir. Si on essaye d'ouvrir une telle fenêtre en utilisant une connexion WiFi, elle n'aura même pas le temps de s'afficher que le résultat est disponible. Au niveau de l'interface de la vue principale, il est semble-t-il déconseillé de modifier le menu contextuel apparaissant lorsqu'une pression longue est effectuée. Il est possible de créer un menu contextuel qui s'affiche lorsque une pression longue est détectée sur la zone de texte mais il est pas possible de le faire sur du texte directement. En essayant sur du texte, un menu contextuel apparait effectivement mais ce menu est pour le copier/coller. Il est possible de le modifier mais il n'y pas beaucoup d'informations à ce sujet.

7 Conclusion

L'application fonctionne correctement même sur un appareil physique. Certaines erreurs peuvent encore survenir mais il est bon de signaler que certaines ne peut qu'être difficilement évité car des fois le système Android décidera de tuer un composant de l'application la rendant inutilisable. L'application devra alors être relancée.