Chapitre I

Les applications mobiles sous android : anatomie et organisation

Le développement pour Android est possible grâce à un **framework** et un ensemble **d'API** qui permettent de construire des applications mobiles et des jeux dans l'environnement JAVA.
Les applications Android sont écrites avec java. Les outils fournis dans le SDK d'Android compilent le code ainsi que tous les fichiers de données et de ressources dans un APK, un paquet Android, qui est un fichier d'archive avec un suffixe .apk. Un fichier APK contient tout le contenu d'une application Android et est le fichier utilisé par le système Android pour installer l'application

I.1 Anatomie d'une applications android

• Modèle à multitudes de points d'entrées

Les applications Android sont constituées d'une combinaison de composants distincts pouvant être invoqués individuellement. Dans l'essentiel ces entités sont :

Les activités, **les services**, **les récepteurs de diffusion "Broadcast receivers"** et **les fournisseurs de diffusion "Content providers"**. Ils forment ensemble ***la logique métier*** et ***la logique présentation*** de l'application.

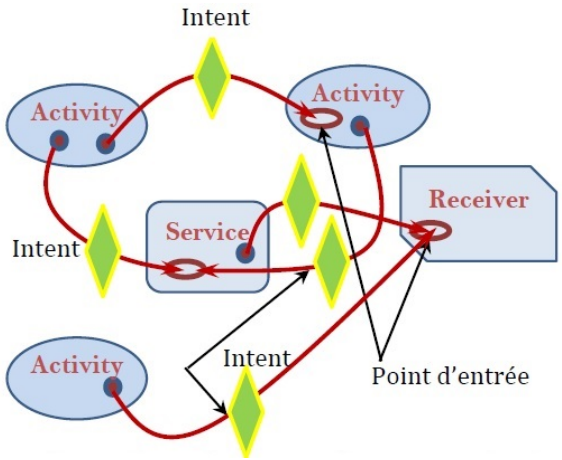

Figure I.1: "Entités composant une application android"

Les entités sont démarrées en utilisant **des intentions "Intents"** (*voir Fig.I.1*). Il est possible même de démarrer un composant dans une application différente. Ainsi, est formé **un modèle** d'applications supportant **le concept de multiplicité de points d'entrée**.

I.1.1 Les entités composant l'Android apps

Une application Android peut être formée des quatres (4) entités suivantes :

- **Les activités**
- **Les services**
- **Les récepteurs de diffusion "Broadcast receivers"**
- **Les fournisseurs d'accès "Content providers"**

- **Les activités** : La logique présentation est entièrement contenue dans les activités. L'activité est couplée à un "Layout" représentant l'interface et la vue. La logique métier de l'activité est définie dans une instance de la classe "Activity".

- **Les services** : Sont hébergées dans les services, les taches de fonds à durées longues comme les téléchargements ou le jeu de séquences audio. Un service n'est pas couplé à des vues et ne possède pas donc de "Layout". Les services fonctionnent plutôt souvent dans un contexte multithreaded. Une fois démarré, un service peut continuer à fonctionner pendant un certain temps, et même après que l'utilisateur soit passé à une autre application. On envisage trois types de services : ***foregroud***, ***background*** et les ***bind*** services. L'API WorkManager offre un moyen flexible de planifier des tâches et est capable d'exécuter ces tâches en tant que services de premier plan (foregroud) si nécessaire. Dans de nombreux cas, l'utilisation de `WorkManager` est préférable à l'utilisation directe des services de premier plan.

- **Les récepteurs de diffusion "Broadcast receivers"** : Sont utilisés pour démarrer des taches de façon asynchrone en réponse à des événements systèmes nommés aussi "les diffusions" tels que le changement de l'état de connexion du réseau, la réception d'un appel téléphonique ou son initiation, la prise de photo ou lorsque le niveau de charge de la batterie est affaibli. Aussi l'application contenant le broadcast receiver n'est pas obligée d'être maintenue active pour que le pécepteur de diffusion puisse répondre.

- **Les fournisseurs de contenu "Content providers** : permettent de gérer de façon plus flexible un ensemble partagé de données stocké sur le système de fichiers, dans une base de données SQLite, sur le Web ou sur tout autre emplacement de stockage persistant auquel l'application peut accéder. Grâce au fournisseur de contenu, d'autres applications peuvent interroger ou modifier les données si le fournisseur de contenu le permet. Par exemple, le système Android fournit un fournisseur de contenu qui gère les informations de contact de l'utilisateur. Par conséquent, toute application disposant des autorisations appropriées peut interroger le fournisseur de contenu, tel que pour lire et écrire des informations sur une personne en particulier. Pour le système, un fournisseur de contenu est un point d'entrée dans une application pour la publication d'éléments de données nommés, identifiés par un schéma URI.

I.1.2 Activation des composants

Les activités, services et récepteurs de diffusion - sont activés par un message asynchrone appelé intention " Intent ".

I.1.2.1 Lancement d'une activité

Une nouvelle instance d'activité est démarrée utilisant un "Intent" qui doit être passé en comme argument dans l'appel ***startActivity().*** L'Intent décrit l'activité à démarrer et transmet toutes les données nécessaires. Si vous souhaitez recevoir un résultat de l'activité quand il est terminé, appelez ***startActivityForResult().*** Votre activité reçoit le résultat en tant qu'objet Intent distinct dans la méthode ***onActivityResult().***

I.1.2.2 Lancement d'un service

Un service est une tâche qui s'exécute en arrière-plan et ne possède pas d'interface utilisateur. Avec Android 5.0 (API niveau 21) et ultérieure, on peut démarrer un service avec **`JobScheduler.`** Pour les versions antérieures à Android 5.0 (API niveau 21), un service est démarré en utilisant des méthodes de la classe **`Service.`** Un service est démarré pour exécuter une opération ponctuelle (comme un téléchargement de fichiers) en passant aussi comme argument un Intent dans l'appel **`startService().`** L'Intention décrit le service à démarrer et transmet toutes les données nécessaires.

I.1.2.3 Emettre (délivrance) d'une diffusion (broadcast)

Une diffusion est un message que n'importe quelle application peut recevoir. Le système délivre diverses diffusions causées par les événements système, par exemple lorsque le système démarre ou la connectivité est activée. Une diffusion (broadcast) est délivrée vers d'autres applications en envoyant un Intent à travers `sendBroadcast()` ou `sendOrderedBroadcast()`.

I.1.3 les intentions `"Intents"`

Sont envisagés sous Android deux types d'"Intents":

I.1.3.1 Les intentions explicites

Ils spécifient le composant à démarrer par nom (le nom de classe entièrement qualifié). On utilise généralement une intention explicite pour démarrer un composant dans l'application, car on connait le nom de la classe de l'activité ou du service à démarrer.

I.1.3.2 Les intentions implicites (voir §chapitre Intents)

Ne nomment pas un composant spécifique, mais déclarent une action générale à effectuer, ce qui permet à un composant d'une autre application de le traiter. Lorsque vous créez un implicite Intent, le système Android trouve le composant approprié à démarrer en comparant le contenu de l'intention aux filtres d'intention déclarés dans le fichier manifeste d'autres applications installées sur le périphérique. Si l'intention correspond à un filtre d'intention particulier, le système démarre ce composant et lui délivre l'objet Intent.
Les principaux attributs définis dans l'objet Intent sont :

- ◇ Name : Nom du composant à relancer.
- ◇ Action : une chaine de caractère qui spécifie l'action générique à prendre.
- ◇ Data : L'URI (un objet `Uri`) qui fait référence aux données à traiter.
- ◇ Category : Chaîne contenant des informations supplémentaires sur le type de composant ciblé par l'intention : `CATEGORY_BROWSABLE, CATEGORY_LAUNCHER`
- ◇ Extra : Un `Bundle` régit par le schéma Paires Clé-valeur qui porte les informations supplémentaires requises pour accomplir l'action demandée.

◦ Exemples pour `Intent` explicite et implicite

a) "Intent" explicite : Si on a créé par exemple un service dans l'application, nommé DownloadService, conçu pour télécharger un fichier à partir du Web, le service est démarré via les lignes de code suivantes:

```
Intent downloadIntent = new Intent(this,DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

b) "Intent" implicite : Lorsqu'on souhaite que l'utilisateur partage un document avec d'autres personnes, on peut créer une intention avec comme attribut pour ACTION `ACTION_SEND` et ajouter des extras spécifiant le contenu à partager. Lorsque on appele `startActivity()` avec cette intention, le système répond en proposant les activités qui sont en mesure de réaliser un "SEND".

```
Intent sending_intnt = new Intent();
sending_intnt.setAction(Intent.ACTION_SEND);
sending_intnt.putExtra(Intent.EXTRA_TEXT, textMessage);
sending_intnt.setType("text/plain");
if (sending_intnt.resolveActivity(getPackageManager())!=null)
startActivity(sending_intnt);
```

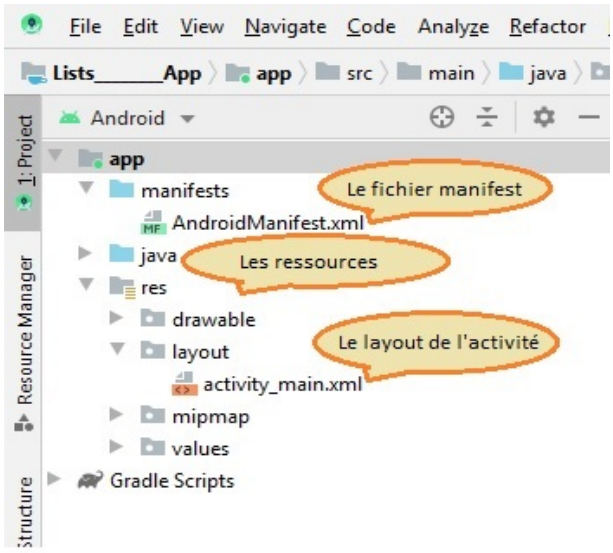

Figure I.2: "Entités composant une application android"

I.2 Le volet XML : Le fichier AndroidManifest.xml et les ressources

Chaque projet Android est muni d'un fichier XML pour ce qui est dit manifests et des fichiers supplémentaires pour le contenu statique tous en XML aussi représentant ce qui appelé les ressources. Sont repérés tous sur l'onglet "project" dans l'environnement Android Studio comme c'est montré à Fig I.2.

I.2.1 L'AndroidManifest

Chaque projet d'application doit avoir un fichier `AndroidManifest.xml` (avec précisément ce nom) à la racine du jeu de source du projet. Le fichier manifeste décrit les informations essentielles sur l'application pour les outils de génération Android, le système d'exploitation Android et Google Play.

Pour que l'application puisse utiliser des activités, ces activités et certains de leurs attributs doivent être déclarées dans le manifeste.
Dans le fichier manifeste sont déclarés entre autres les éléments suivants :

- ***Le nom de package*** : correspond généralement à l'espace de noms du code. Les outils de génération Android l'utilisent pour déterminer l'emplacement des entités de code lors de la construction du projet.

- ***Les composants de l'application*** : Les activités, les services, les récepteurs de diffusion et les fournisseurs de contenu formant l'application. Chaque composant doit définir des propriétés de base telles que le nom de sa classe et des filtres d'intention décrivant le mode de démarrage du composant.

- ***Les autorisations (permissions)*** : nécessaires à l'application pour accéder aux parties protégées du système ou à d'autres applications. On y déclare également toutes les autorisations que doivent avoir les autres applications pour accéder au contenu de l'application..

I.2.2 Exemple d'AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="1" android:versionName="1.0"
package="com.example.myapp">
<uses-sdk android:minSdkVersion="15"
android:targetSdkVersion="26" />
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/
        </intent-filter>
    </activity>

    <activity
        android:name=".DisplayMessageActivity"
        android:parentActivityName=".MainActivity" />
</application>
</manifest>
```

I.2.3 Les ressources

Les ressources sont des fichiers supplémentaires représentant et le contenu statique utilisé par le code, tels que les bitmaps, les définitions de disposition, les chaînes de caractères de l'interface utilisateur, les instructions d'animation, etc. On doit externaliser les ressources de l'application, telles que les images et les chaînes, du code, afin de pouvoir les gérer de manière indépendante.
Une fois qu'on a externalisé les ressources d'applications, on peut les accéder à l'aide des ID de ressources générées dans la classe R du projet.

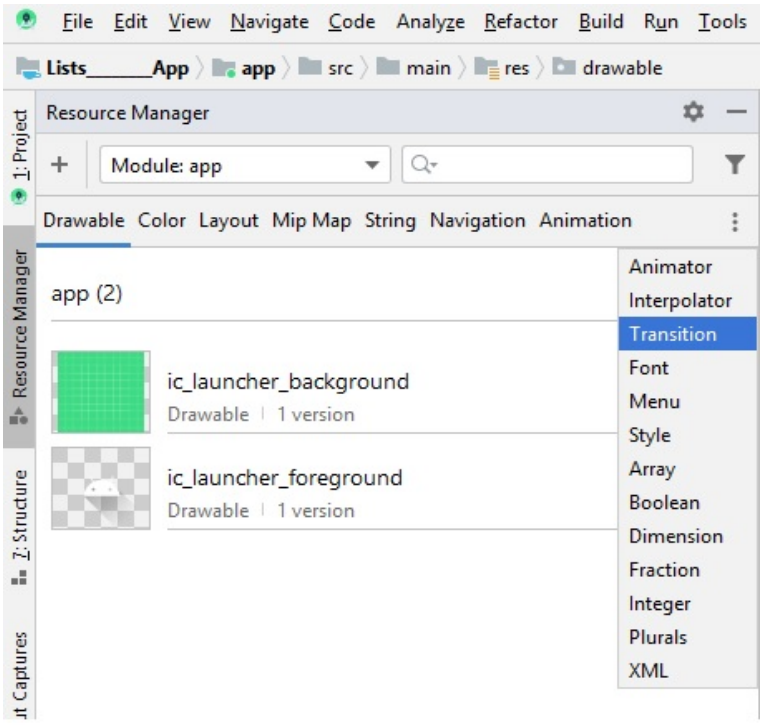

Figure I.3: "Le ressources manager"

Le ressource manager sur L'IDE Android Studio répertorie les ressources en différentes catégorie comme c'est montré à la Fig.I.3 et permet leur gestion.