Épisode 3 : Concepts fondamentaux

© Alain Menu – édition 2.1, septembre 2013

Objectifs : Connaître les composants logiciels Android et leur couplage.

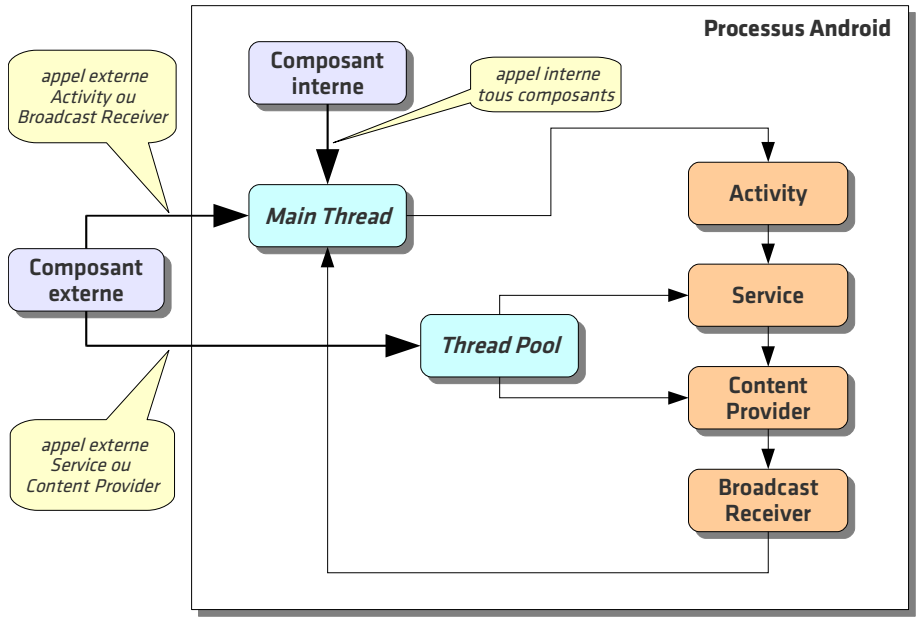
Table des matières

3.1. Composants d'une application Android

Les applications Android sont constituées de composants à couplage, liés par un manifeste.
Le manifeste (fichier AndroidManifest.xml localisé à la racine du projet) décrit les composants, leurs interactions et les métadonnées de l'application, dont notamment ses exigences en matière de plateforme et de matériel.

Il existe 7 types de composants :

- ***Activities*** : C'est la couche présentation de l'application, chaque écran est une extension de la classe Activity et utilise des *Views* pour former une interface utilisateur.
- ***Services*** : Ces composants tournent en arrière-plan pour déclencher des *Notifications* et mettre à jour les sources de données et les *Activities* visibles.
- ***Content Providers*** : Ces composants gèrent les sources de données partageables ; ils peuvent être configurés pour en autoriser l'accès à d'autres applications et pour utiliser celles qu'elles exposent.
- ***Intents*** : Ce sont des composants de communication entre applications qui permettent de diffuser des messages au sein du système ou à destination d'une *Activity* ou d'un *Service* cible.
- ***Broadcast Receivers*** : Ce sont des consommateurs de messages diffusés par les *Intents*. Ils sont à l'écoute des intentions envoyées par les autres composants applicatifs et peuvent démarrer automatiquement l'application pour répondre à un message entrant.
- ***Widgets*** : Ce sont des variantes des *Broadcast Receivers* permettant de créer des composants interactifs et dynamiques incorporables dans l'écran d'accueil.
- ***Notifications*** : Ces composants permettent d'envoyer un signal aux utilisateurs sans dérober le focus ni interrompre les *Activities* en cours.



Un processus Android peut héberger 4 types de composants :

- *Activity*,
- *Service*,
- *Content Provider*,
- *Broadcast Receiver*.

Pour les appels internes, tous les composants répondent sur le thread local.

Pour les composants clients externes, seuls les appels aux *Service* et *Content Provider* sont multithreadés. Les appels aux *Activities* et *Broadcast Receiver* importants sont routés à travers la file du thread principal.
Un appel à un *Broadcast Receiver* est surveillé pour répondre dans une durée maximale de 10 secondes. Cette restriction ne semble pas exister pour les appels externes aux *Content Provider* et *Service* ; le thread principal du client externe peut donc rester en attente jusqu'au retour de ces appels (mode synchrone)...

3.2. Notion de manifeste

Chaque projet android doit comporter un fichier XML nommé « Manifeste » (AndroidManifest.xml) stocké à la racine de la hiérarchie du projet. Le manifeste permet de définir la structure, les métadonnées, les composants et les prérequis d'une application.

3.2.1. Structure

Le manifeste est constitué d'une balise racine <manifest> avec un attribut package qui est celui du projet.

L'attribut versionCode définit la version interne de l'application sous forme d'un entier, tandis que versionName est la version sous forme de texte visible par l'utilisateur.

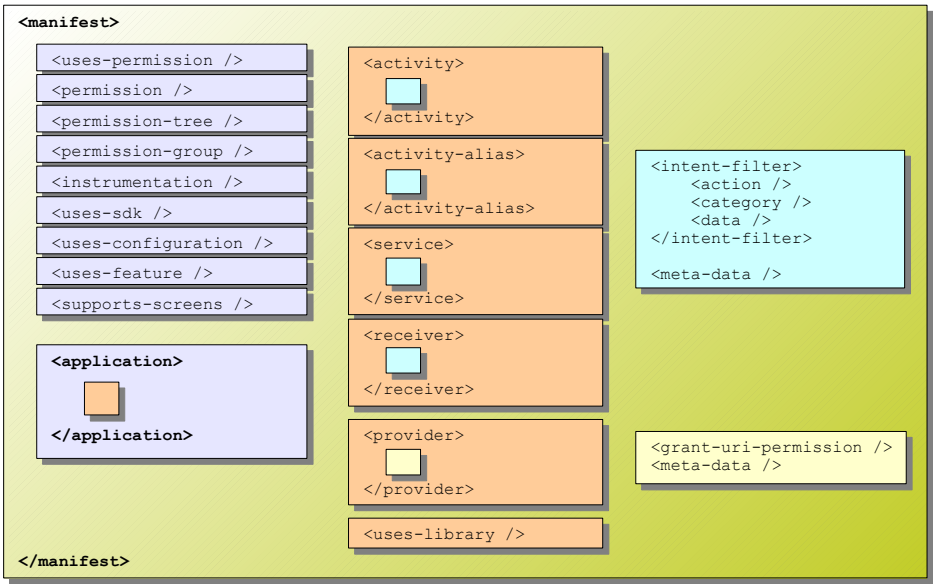
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mydomain.myappli"
    android:versionCode="1"
    android:versionName="1.0">

    [ … nœuds du manifeste … ]

</manifest>
```

nœud de manifeste type

Balises du manifeste →



Seuls sont obligatoires les éléments <manifest> et <application>... ils ne doivent apparaître qu'une seule fois.

3.2.2. Le manifeste du « Hello World »

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.alainmenu.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" android:debuggable="true">
        <activity
            android:name="com.alainmenu.helloandroid.HelloAndroid"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Le manifeste du programme de test réalisé lors de l'épisode 2 peut servir d'exemple pour décrire les principaux éléments de couplage d'une application Android...

- Balise <uses-sdk>
  - Attribut android:minSdkVersion="8" : Niveau minimal d'API pour que l'application fonctionne (précisé lors de la création du projet).
  - Attribut android:targetSdkVersion="17" : Niveau maximal d'API pour laquelle l'application est supposée fonctionner (précisé lors de la création du projet).

- Balise <application>
  - Attribut android:icon="@drawable/ic_launcher" : Définition de l'icône de l'application (fichier PNG à fournir en plusieurs formats, voir rubrique Comprendre le code).
  - Attribut android:label="@string/app_name" : Nom visible de l'application, spécifié sous forme de ressource dans res > values > strings.xml.

  - Balise <Activity>
    - Attribut android:name=".HelloAndroid" : Nom de la classe qui implémente l'activité. Ici il s'agit d e com.alainmenu.helloandroid.HelloAndroid, indiquée en relatif par rapport à l'attribut package de la balise racine <manifest>.
    - Attribut android:label="@string/app_name" : Nom de l'activité. Peut être omis, dans ce cas le nom de l'application sera utilisé (ici, ils sont égaux).
    - Balise <intent-filter> : « Intention » de démarrage de l'activité.
      - Balise <action>, attribut android:name="android.intent.action.MAIN" : L'activité démarre en tant que point d'entrée principal de l'application.
      - Balise <category>, attribut android:name="android.intent.category.LAUNCHER" : Cette catégorie fait apparaître l'activité dans le lanceur d'applications.

3.3. Cycle de vie d'une activité

Sous Android, il n'y a qu'une activité « active » à la fois, dite en avant-plan (*foreground*).
Le système d'exploitation est libre de terminer une activité qui est en arrière-plan lorsque la quantité de mémoire libre du système est trop basse. Ainsi, une application Android, lorsqu'elle n'est plus en avant-plan, doit être capable de maintenir son état pour garantir un fonctionnement transparent à l'utilisateur, elle doit notamment retrouver les valeurs et affichages précédents lorsqu'il remet l'application en avant-plan...

C'est à l'application de gérer son état, ses données, et ses ressources afin d'être prête à être interrompue ou bien terminée à tout moment !

3.3.1. Les callbacks

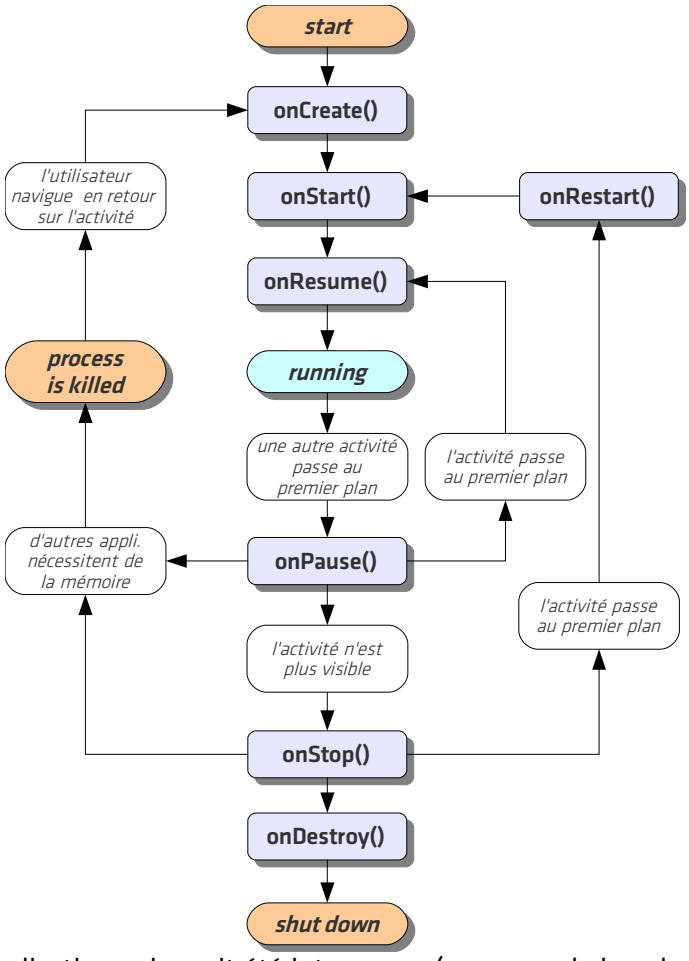
Le temps pris pour traiter la réception de ces callbacks est limité à 5 secondes sous peine de voir le message « Application Not Responding »...

Pour répondre à ses événements, des méthodes de réponse (*callback*) sont définies : onCreate(), onResume(), onPause(), onDestroy(), ...

Principaux *callbacks* et actions à entreprendre lors de leur réception :

- onCreate() : appelé quand l'application démarre ou redémarre → Initialiser les données statiques, établir des liens vers les données et les ressources, positionner l'interface avec setContentView().
- onResume() : invoqué quand une activité passe en avant-plan → Reprendre le contrôle des ressources exclusives, continuer les lectures audio et vidéo ou les animations.
- onPause() : appelé quand l'activité quitte l'avant-plan → Sauvegarder les données non encore sauvegardées, libérer l'accès aux ressources exclusives, stopper la lecture audio, vidéo et les animations.
- onDestroy() : invoqué quand l'application est fermée → Nettoyer les données statiques de l'activité, libérer toutes les ressources obtenues.

Diagramme présentant les principaux états du cycle de vie d'une activité Android →



Android pratique le *sandboxing*, technique qui consiste à séparer presque totalement les applications entres elles.

Chaque application est restreinte à des actions bien définies (accès mémoire, accès aux capteurs, ...), cette pratique permet de protéger le système au maximum en évitant de laisser les applications effectuer des actions inattendues.

La forte restriction apportée aux communications entre applications est compensée par l'existence des intentions (*intent*) qui sont à la fois des flux de contrôle et de données.

Les intentions peuvent être passés à un autre composant applicatif, de la même application ou non, de façon implicite (lire de la musique, scanner un code barre, ...) ou explicite (lancement d'une autre activité matérialisée par une classe Java).

3.3.2. Lecture du diagramme

Lorsque une application est lancée, l'activité est chargée et la méthode onCreate() est lancée. Cette méthode permet d'initialiser l'activité. Il sera aussi possible de restaurer un précédent lancement de l'application qui aurait été interrompu (par exemple lors de la rotation de l'écran).

onStart() est lancée tout de suite après onCreate() lorsque l'activité va devenir visible à l'utilisateur. Cette méthode permet par exemple de charger des données.

onResume() est lancée lorsque l'application est passé en avant-plan ; ce qui permet notamment de (re)lancer des threads, ...

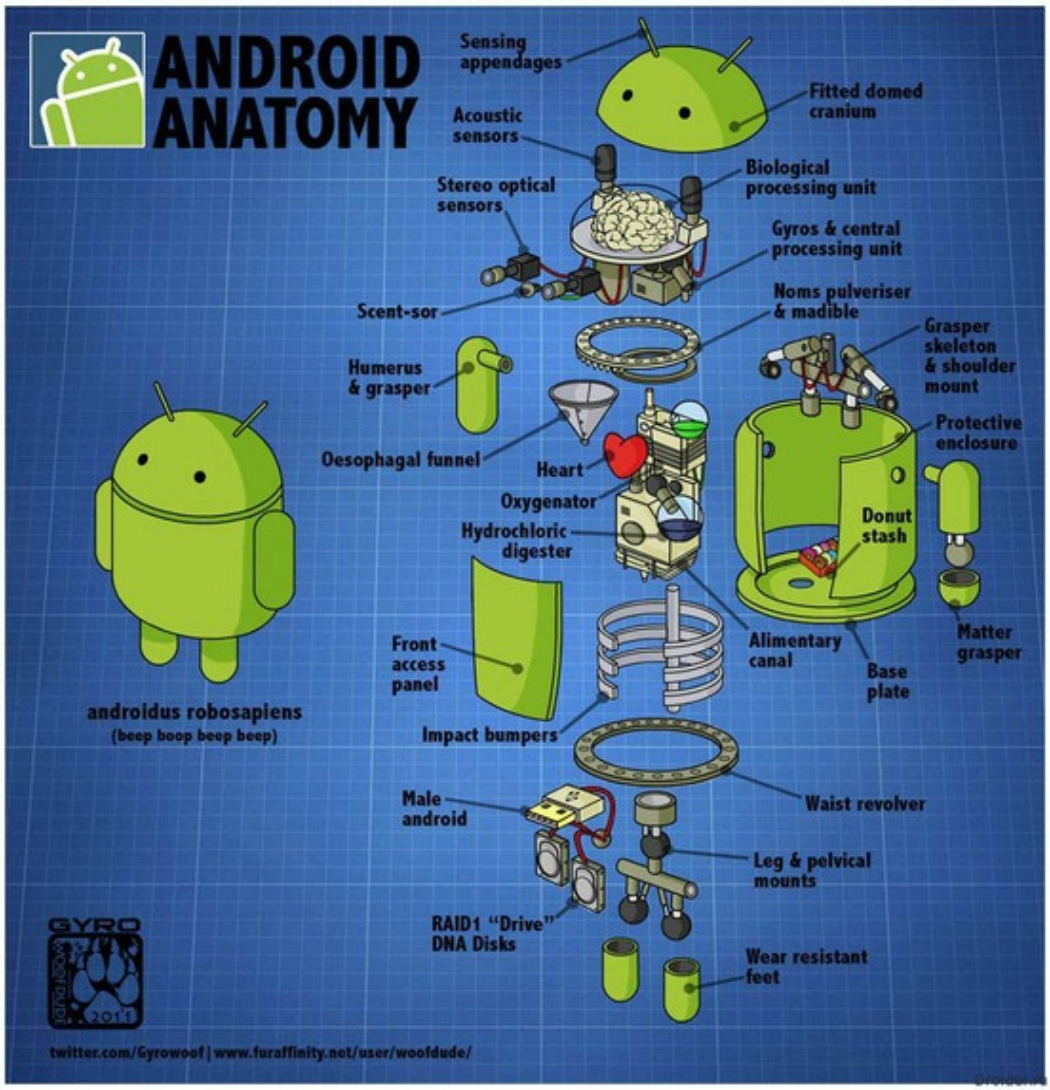
onPause() est lancée lorsqu'une autre activité va s'afficher à l'avant-plan. C'est le moment de sauver toutes les données saisies par l'utilisateur. En effet, le système peut décider par la suite de mettre fin à notre activité (par exemple à cause d'un manque de mémoire).

Le traitement réalisé par onPause() doit cependant être rapide, car aucune activité ne pourra démarrer tant que celui-ci ne sera pas fini...

onStop() est lancée lorsque l'activité n'est plus visible. Le processus passe en sommeil. Le système pourra décider de mettre fin définitivement à l'activité si nécessaire. Si l'activité utilise des ressources, celles-ci seront libérées. Que ce passe-t-il ensuite, soit l'utilisateur revient dans votre activité, dans ce cas, onRestart() sera lancée afin de réveiller votre activité. Soit votre activité n'a plus lieu d'exister et dans ce cas, onDestroy() sera lancée.

Enfin, onDestroy() met fin au cycle de vie de l'activité ; il y a libération des ressources et purge des fichiers temporaires.

Lorsqu'une activité se termine, et même si l'application n'avait qu'une seule activité, l'application reste en cours d'exécution en mémoire (pour preuve, à partir de l'écran d'accueil, un appui long sur la touche *home* permet d'afficher la liste des applications lancées...).



Références

Site Google officiel : http://www.android.com/
Site officiel de développement Android : http://developer.android.com/
Site officiel des sources Android : http://source.android.com/
Site de l'OHA : http://www.openhandsetalliance.com/
Communauté francophone autour d'Android : http://www.frandroid.com/,
et le site collaboratif associé : http://wiki.frandroid.com/wiki/Accueil
« Développement d'applications professionnelles avec Android 2 », Reto Meier – Pearson – ISBN : 978-2-7440-2452-8