Introduction au développement d'applications mobiles avec Androïd

James EAGAN

TELECOM ParisTech

université PARIS-SACLAY

Agenda

Introduction

Applications mobiles

Anatomie d'une application Androïd

Composants d'interface (widgets)

Outils de développement

Objectifs

À la fin de ce module, vous :

- Comprendrez les contraintes particulières aux applis mobiles
- Pourrez créer une appli Androïd
- Saurez comment intégrer une appli avec d'autres

Qu'est-ce qui est spécial d'applis mobiles ?

Elles sont limitées en ressources

Elles “connaissent” leur contexte

Elles sont omniprésentes

Elles sont intrusives

Attention aux ressources

Éviter le gaspillage de ressources

Personne ne veut :

Recharger son mobile plus d'une fois par jour

Faire banqueroute à cause de l'utilisation de données

Un téléphone 🐛 car une appli utilise tout le CPU

Une appli qui plante à cause d'une appli qui bouffe toute la mémoire

Remplir tout le stockage avec les données d'une appli

- **Les ressources limitées encouragent la créativité**

Considérations IU de base

Une appli tourne en plein écran

Elles doivent être réactives

Elles doivent prendre en compte l'orientation de l'écran

Elles tournent sur des matériels différents

Elles n’ont pas forcément un clavier, boutons, *etc.*

Elles peuvent être arrêtées brutalement

Applis peuvent être omniprésentes

Une appli mobile peut souvent :

- Capter des données de l'environnement (lumière, accélération, direction, GPS, …)
- Communiquer sur Internet
  - Pour échanger avec des services distants
  - Pour charger des ressources publiques
  - Pour faire un calcul à distance
  - Pour recevoir des données poussées vers le dispositif mobile
- Communiquer avec d'autres dispositifs (BlueTooth, NFC, …)

Mais elle doit être encore utilisable face à une connexion faible ou absente

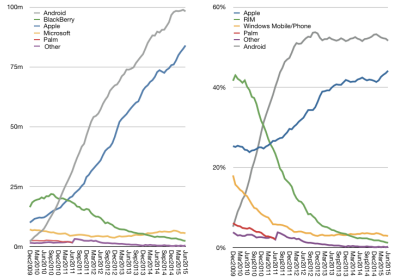
Applis peuvent être intrusives

Une appli peut avoir un accès :

- aux contacts de l'utilisateur
- aux agendas de l'utilisateur
- aux photos & vidéos de l'utilisateur
- à la géolocalisation du dispositif
- au micro du dispositif
- aux caméras du dispositif

... Et si tout ça était envoyé à un tiers méchant ?

Part du marché

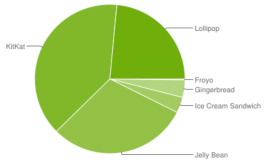


[Asymco, 10 août 2015]

Fragmentation Androïd

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	3.4%
4.1.x	Jelly Bean	16	11.4%
4.2.x		17	14.5%
4.3		18	4.3%
4.4	KitKat	19	38.9%
5.0	Lollipop	21	15.6%
5.1		22	7.9%

Data collected during a 7-day period ending on October 5, 2015.
Any versions with less than 0.1% distribution are not shown.

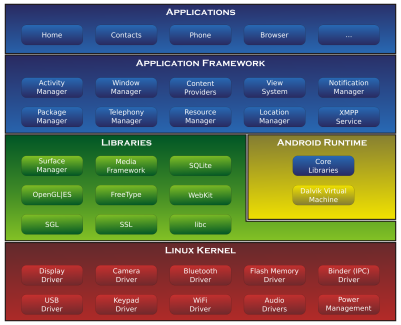


Version	Name	%	Sortie
5.1	Lollipop	8	3/2015
5.0	Lollipop	24	11/2014
4.4	KitKat	62	10/2013
4.3	JellyBean	67	7/2013
4.2	JellyBean	81	11/2012
4.1	Jelly Bean	93	7/2012

Développement en Androïd vs. iOS

	Androïd	iOS
Execution	Dalvik	native
Langage	Java / C++	Objective-C / Swift
Famille d'OS	Linux	Darwin (BSD)
App Store	$25	$99/an
Test	Emulation*	Simulation

Architecture Androïd



Machine Virtuelle Dalvik

Execute du bytecode Dalvik, traduit du bytecode Java

Possibilité d'utiliser n'importe quel langage qui compile en bytecode Java (*e.g.,* Scala, Jython, JRuby, …)

Pas possible si génération de bytecode est dynamique (*e.g.,* Clojure)

Executables, ressources mis ensemble dans un .apk

Applis Androïd vs. bureautiques

Comparée à une application bureautique, une appli Androïd :

- peut avoir ≥1 point d'entrée
- peut implicitement s'intégrer avec des services d'autres applis
- peut proposer des services aux autres applis
- peut interagir avec d'autres applis inconnues

Une appli est organisée en activités :

- Quand l'écran change, c'est souvent une nouvelle activité
- Une activité peut en lancer une autre et peut lui fournir des données
- Une activité peut recevoir un résultat d'une autre activité

Par exemple : scanner un flashcode

Interface Utilisateur

View, ViewGroup

L'IU affichée est une hiérarchie de composants

Pour commencer une activité, appelez `setContentView()` en donnant une référence vers la racine de la vue

La plupart du temps, on définit cet arbre dans un XML layout

On peut aussi le manipuler en Java

Layout en XML

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

<EditText android:id="@+id/edit_euros"
        android:inputType="numberDecimal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/euros_label" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_convert"
    android:onClick="convert" />
</LinearLayout>
```


Widgets

Androïd définit des widgets pour construire l'interface :

- Widgets de base : boutons, cases à cocher, champs de texte, …
- Widgets un peu plus complexes : horloge, zoom, date picker

Tous les widgets sont définis dans le paquetage `android.widget`

Interaction

Deux méthodes existent pour gérer une interaction :

Créer une sous-classe, puis surcharger les méthodes de rappel des événement de saisie.

Abonner des listeners à la view.

Cette méthode est conseillée.

Widget	Classe
Bouton	`Button`
Champs de texte	`EditText, AutoCompleteTextView`
Case à cocher	`CheckBox`
Bouton radio	`RadioGroup, RadioButton`
Interrupteur à bascule	`ToggleButton`
Liste déroulante	`Spinner`
Sélecteur (de date, horaire, …)	`DatePicker, TimePicker`

Bouton

Peux avoir un label, icône, ou bien les deux

XML : android:onclick

Code : setOnClickListener, View.OnClickListener

Dans le layout XML :

```
<Button …
    android:id="@+id/button_send"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Dans le code de l'`Activity` :

```
/* Appelée lorsque l'utilisateur tape le bouton */
public void sendMessage(View view) {
    // Fais qqc ici
}
```

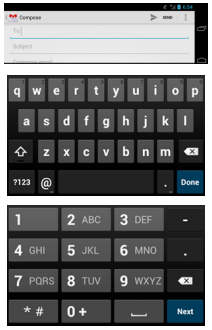
[https://developer.android.com/guide/topics/ui/controls/button.html]

Rajouter un listener en code

Alternatif, dans l'`Activity`, sans utiliser le XML :

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Fais qqc ici
    }
});
```


Champs de texte



Permet de saisir du texte

Peut utiliser un clavier adapté avec
`android:inputType : text` (clavier normal),
`textUri, textEmailAddress, phone, ...`

Options : `textCapSentences,`
`textCapWords, textAutoCorrect,`
`textPassword, textMultiLine, ...`

Peut combiner les options avec | :
`android:inputType="text|textCapWords"`

[https://developer.android.com/guide/topics/ui/controls/text.html]

Rajouter un EditText

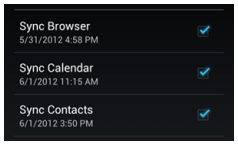
```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

Cases à cocher

Permet de choisir zéro, un, ou plusieurs options

Gére les tapes comme un bouton : `android:onClick`, `View.OnClickListener`

État peut être changé en code avec `setChecked(boolean)` ou `toggle()`



[https://developer.android.com/guide/topics/ui/controls/checkbox.html]

Cases à cocher (2)

Dans le layout XML :

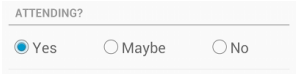
```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Dans l'`Activity` :

```
public void onCheckboxClicked(View view) {
    // La case est-elle cochée ?
    boolean checked = ((CheckBox) view).isChecked();

    // Quelle case a été cochée ?
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked) {
                // Rajouter de la viande
            } else {
                // Enlever la viande
            }
            break;
        case R.id.checkbox_cheese:
            if (checked) {
                // Rajouter du fromage
            } else {
                // Pas de fromage
            }
            break;
        // TODO: Végétarien ...
    }
}
```

Boutons radio



Pour sélectionner une option parmi plusieurs, quand on veut voir toutes les options à la fois

Gérés comme des `CheckBox`, mais avec `RadioButton` : `android:onClick`, `setChecked(boolean)`, `toggle()`

Appartiennent à un `RadioGroup`

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

[https://developer.android.com/guide/topics/ui/controls/radiobutton.html]

Adapters

On a souvent besoin de s'intégrer avec une source de données, c.f. pour afficher des résultats dans une liste.

Un adaptateur permet de relier une source de données statiques ou dynamiques à un tel widget.

Relier un adaptateur à un curseur d'une BD peut aider à efficacement gérer la mémoire et de construire les vues à la demande.

Opérations lentes

Androïd utilise un fil d'exécution pour l'IU

- Si une opération (potentiellement) de longue durée s'exécute sur le fil principal (*e.g.,* sleep, accès au réseau, BD, …), l'interface ne réponde pas.
- Aucun autre fil ne puisse modifier l'IU.

Pour exécuter une longue opération :

- Un nouveau *worker thread* est créé pour la faire.
- Le *worker thread* envoie des messages au *UI thread* via un `Handler`.
- À la réception du message, le `Handler` tourne sur l'UI thread.

Et si on n'utilisait pas de thread ?

Lorsqu'un un événement arrive (comme un tap), il est passé à l'UI thread.

Androïd surveille le temps de gestion d'un événement

Si l'événement n'est pas géré assez vite, Androïd traite l'appli comme non-réactive et propose à l'utilisateur le la fermer brutalement.

Le *strict mode* peut être activé pour signaler ce genre d'erreur

Utiliser les bons outils dès le début est fortement conseillé : `Handler`, `IntentService`, `AsyncTask`, `runOnUiThread`.

`AsyncTask` & `runOnUIThread`

La classe `AsyncTask` permet de faire une tâche lente sur un autre fil d'exécution tout en affichant une barre de progrès. Elle utilise des types génériques et peut être spécialisée aux besoins.

La méthode `runOnUIThread` de la classe `Activity` permet d'exécuter du code sur le fil principal à partir d'un fil secondaire.

Le deux solutions utilisent les mêmes mécanismes que les `Handler`s.

Sages conseilles

Éviter les méthodes longues et compliquées.

Suivez le principe du 💋 : Keep It Simple, Stupid !

Si votre solution est difficile à comprendre ou à implementer, changez de cap, elle n'est probablement pas la bonne solution.

UI Guidelines

Google a publié de très bonnes guidelines

Pour aider aux applis tierces de ressembler à celles du système

Pour aider les utilisateurs à reconnaître des symboles et idiomes communs

Pour faciliter l'adaptation aux écrans, densités, et formats différents

Suivez-les autant que possible.

Cartes

Il est possible de placer des items sur une carte :

- Des widgets Google Maps sont disponible avec un clef API (gratuit), lié à l'appli
- Mapsforge propose une API similaire basée sur OpenStreetMap

developer.android.com

[Transparents adaptés de ceux de Samuel Tardieu]