ANDROID

Un système d'exploitation pour appareils mobiles

Marc Christie
ISTIC - MMM

Plan du cours

- I - Android, les présentations
  - Contexte
  - Architecture
- II – Android et les applications
  - Cycle de vie
  - Activités
- III – Android et les interfaces
  - Layouts et Views
  - Menus et évènements
  - Fragments et malléabilité
- IV – Android et les données
  - Accès aux ressources
  - Content Providers
- V – Android et la réalité augmentée
  - Principes
  - API et outils

I - Android, contexte et architecture

Contexte

- La convergence des technologies matérielles (processeurs, écrans tactiles, GPS, boussole, accéléromètre,...), du réseau et du logiciel mène à des évolutions majeures dans le monde informatique (technologies embarquées et mobilité)

- Ces évolutions majeures s'accompagent de changements importants dans les usages

- S'en suit un marché en forte croissance des appareils mobiles:
  - smartphones
  - tablettes

- D'ou un intérêt majeur des acteurs du logiciel autour des appareils mobiles

Révolution des usages

- Les technologies mobiles ont transformé radicalement notre façon d'interagir avec le monde (à la fois dans l'utilisation et diffusion de l'information)

- Qui pensait il y a 5 ans (l'iPhone sort en 2007) que:
  - on utiliserait la géolocalisation pour retrouver ses amis au café
  - on jouerait avec des jeux en réalité augmentée sur mobile
  - on accéderait immédiatement à la TV, la radio et toute la presse

- Ou que les technologies mobiles (tablettes) réduiraient la fracture générationnelle numérique?
  - les surfaces tactiles sont facilement adoptées par les séniors, de par leur grammaire gestuelle intuitive

- Et amèneraient leur lots d'ambiguités
  - instantanéité = dépendance à l'information MAIS gain de temps
  - encore plus d'interactions sociales mais moins de présence

Impacts

- Communication vs. services:
  - Tous les acteurs (industriels, administratifs, associatifs) veulent apparaître sur ces nouveaux médias
  - Oui, c'est un phénomène de mode, mais pas uniquement
    - Il apparait souvent une vraie valeur ajoutée (service aux clients, information, interaction, …)
    - Couplage effectif de l'intérêt du client (le service rendu) et de l'intérêt de l'acteur (qui utilise le service, que fait-il, quand et où?)

Industries du logiciel

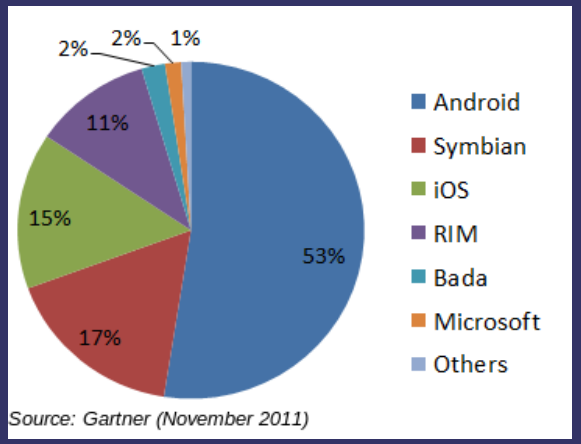
Industries du logiciel:

- Besoins significatifs de développement sur terminaux mobiles
  - Portage d'applications existantes
  - Accès distants aux données existantes
  - Nouveaux besoins et nouvelles modalités
  - Lien avec les réseaux sociaux

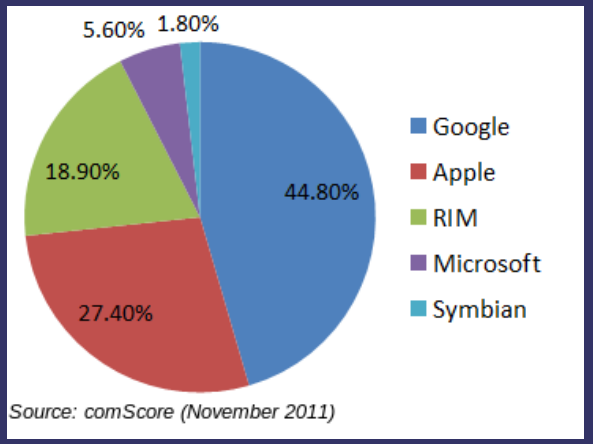
- Problématique du portage d'application entre terminaux (Blackberry, iPhone, WP7, Android)
  - Et on ne va pas dans le sens de la simplification (Samsung Bada OS en 2010, WP8 en 2012, arrivée de Baidu Cloud en Chine en 2012, marché d'un milliard de téléphones), Arrivée d'Alibaba en 2012 (Chine également)

Ampleur du phénomène

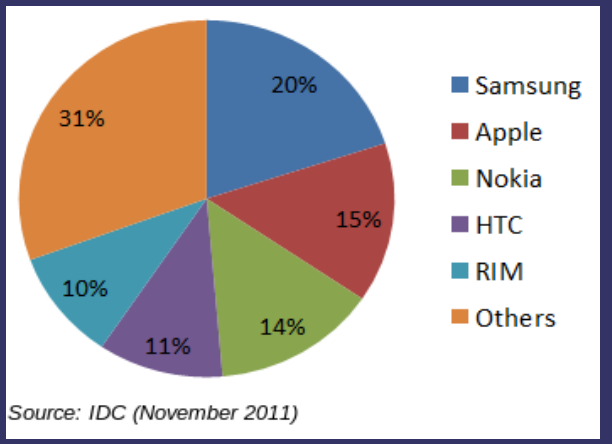
- Mondial (source Gartner)
  - 172 Milllions de Smartphones vendus en 2009
  - 297 Millions de Smartphones vendus en 2010 + (172%)
    - 19% du marché mobile de 1.6 Milliard d'appareils
    - En augmentation de 72% par rapport à 2009
  - IDC estime 982 Millions de terminaux seront vendus en 2015



Rép. par OS – Mondial



Rép. par OS USA



Rép. Par fabricant - Mondial

- 1.1 Milliard de Smartphones en usage dans le monde (fin 2012)
- 3.3 Milliards d'ici 2018 (et x12 en volume de données entre 2012 et 2018)

Positions

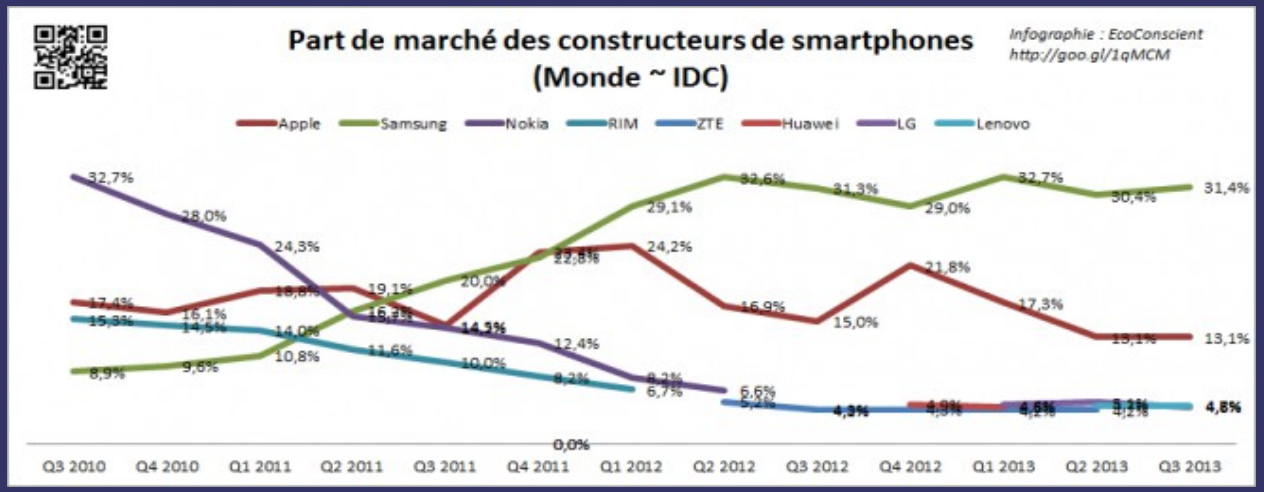
- Dans l'Europe des 5

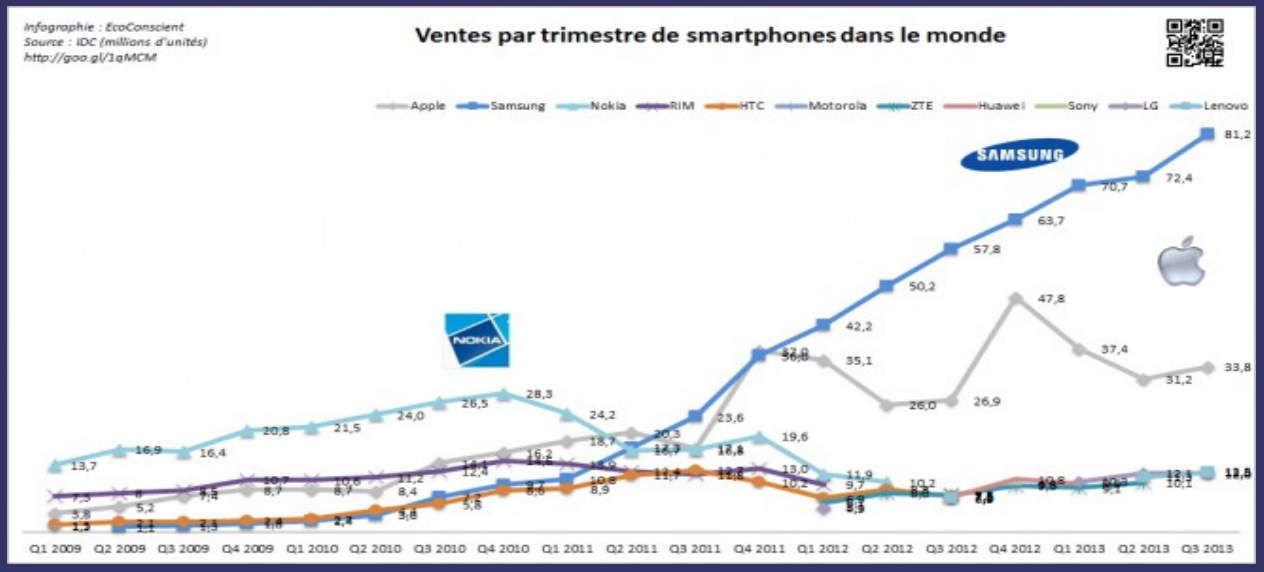
Classement des plateformes de smartphones dans l'Europe des 5 en pourcentage d'utilisateurs de smartphones Moyenne sur une période de 3 mois se terminant en Juillet 2011 vs Juillet 2010 Total Europe des 5 (Allemagne, France, Italie, Espagne et Royaume-Uni) des abonnés mobiles, 13 ans et + Source : comScore MobiLens			
Smartphone Platform Plateforme de Smartphone	**Share (%) of EU5 Smartphone Users Pourcentage des utilisateurs de Smartphones dans l'Europe des 5**		
	Juillet 2010	**Juillet 2011**	**Point de changement**
Total Utilisateurs de Smartphones	*100,00%*	*100,0%*	*0,0*
Symbian	53,9%	37,8%	-16,1
Google	6,0%	22,3%	16,2
Apple	19,0%	20,3%	1,2
RIM	8,0%	9,4%	1,5
Microsoft	11,5%	6,7%	-4,8

Source: Comscore, Aout. 2011

Positions

- Dans le monde (part de marché)





Android

- Android tient une position particulière dans ce phénomène:
  - Arrivée tardive, mais pénétration très rapide du marché (de 0 à 50% des ventes mondiales en moins de 4 ans, et arrive à 80% en oct. 2013).
    - un seul OS, mais différents constructeurs (contrairement à iOS ou RIM)
    - offre matérielle et logicielle très importante
    - plateforme ouverte et facilement accessible pour le développement
    - couplage avec d'excellents terminaux (Samsung galaxy)
  - Intégration complète de la suite logicielle de Google (mail, maps, streetview, docs, google+,...)

“ Nous avons désormais activé plus de 900 millions d'appareils Android dans le monde et nous avons jusqu'à 1,5 million d'activations chaque jour. C'est assez étonnant compte tenu du fait que le premier smartphone Android a été lancé il y a mois de cinq ans. " Larry Page (Juillet 2013)

Android

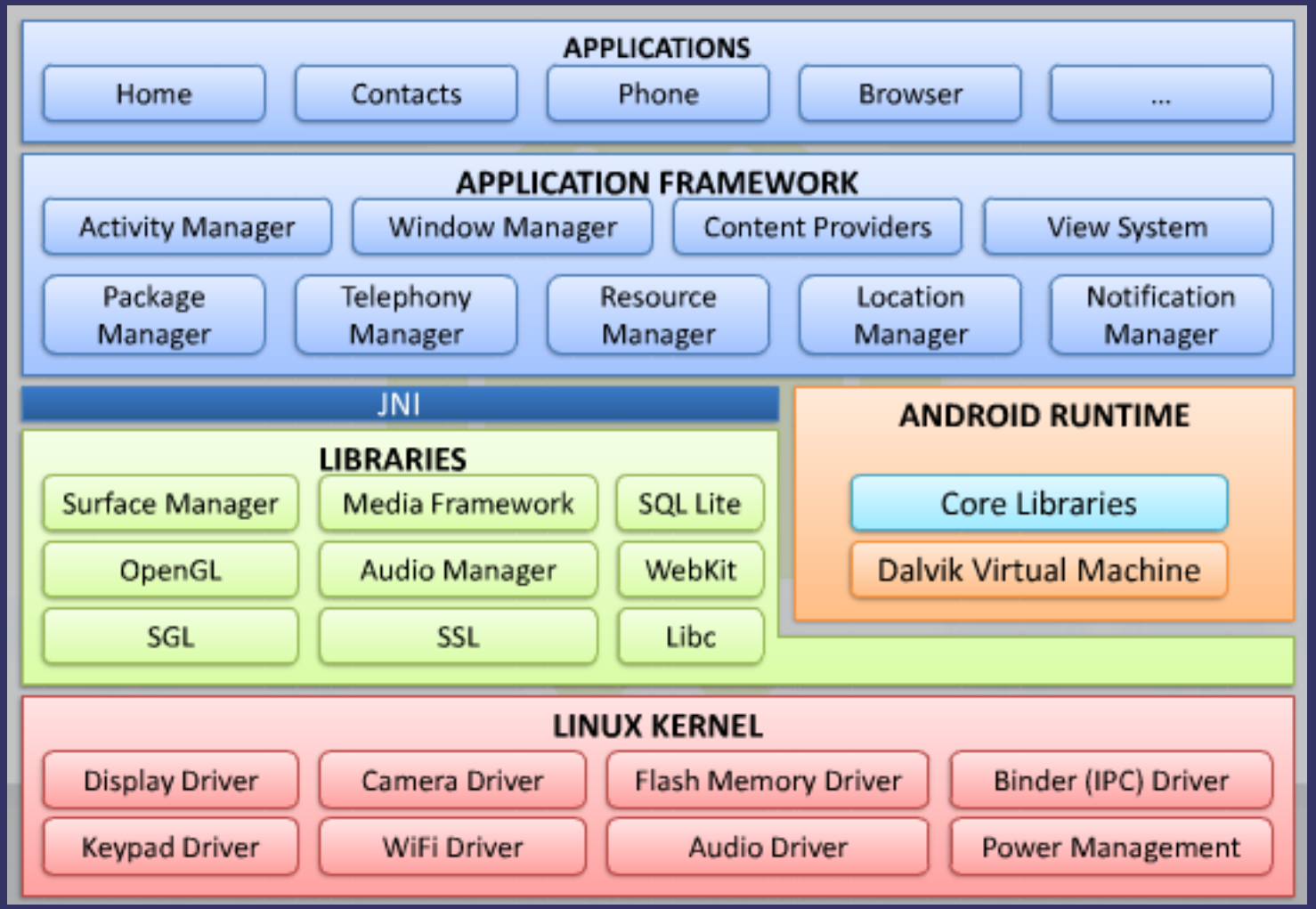
- Android OS = système d'exploitation pour appareils mobiles, par Google

  - OS **gratuit** (utilisateurs) et **ouvert** (licence Apache 2)
  - Intégration des applications Gmail, Google Maps & Street View, Agenda, Talk, Lattitude...
  - Coût limité pour les constructeurs (10 $ / appareil)
  - Le projet Android est entièrement financé par les revenus de la publicité provenant d'Android

- **sortie officielle** annoncée par Google en Nov 2007 premier Android Phone (HTC Dream Sep 2008), il y a 4 ans seulement... après un rachat de la société Android Inc. par Google en 2005.

- distribution des applications sur un Android Market (20 000 applications disponibles en déc. 2009, > 200 000 applications en mai 2011, 500 000 en oct 2012)

Architecture



android

Android en détails

- Un noyau
- Des librairies
- Un cadre applicatif
- Une collection de fonctionnalités
- Un SDK, un émulateur, une intégration dans Eclipse

- Cadre applicatif basé composants
  - Réutilisabilité et remplacement des composants
- Intègration d'une machine virtuelle Java (Dalvik)
  - Une machine virtuelle par application
  - Communication asynchrone entre composants
- Browser intégré
- Capacités d'affichage
  - Librairie graphique 2D
  - Librairie graphique 3D basée sur OpenGL ES 2.0
- Gestion de données SQLite
- Support de multiples média (audio/vidéo)
- GSM/Wifi/EDGE/3G
- Camera, GPS, boussole, accéléromètre
- Un environnement de développement sous Eclipse

- 12 Millions de lignes de code
  - 3 Millions en XML, 2.8 Millions en C
  - 2.1 Millions en Java, 1.75 en C++

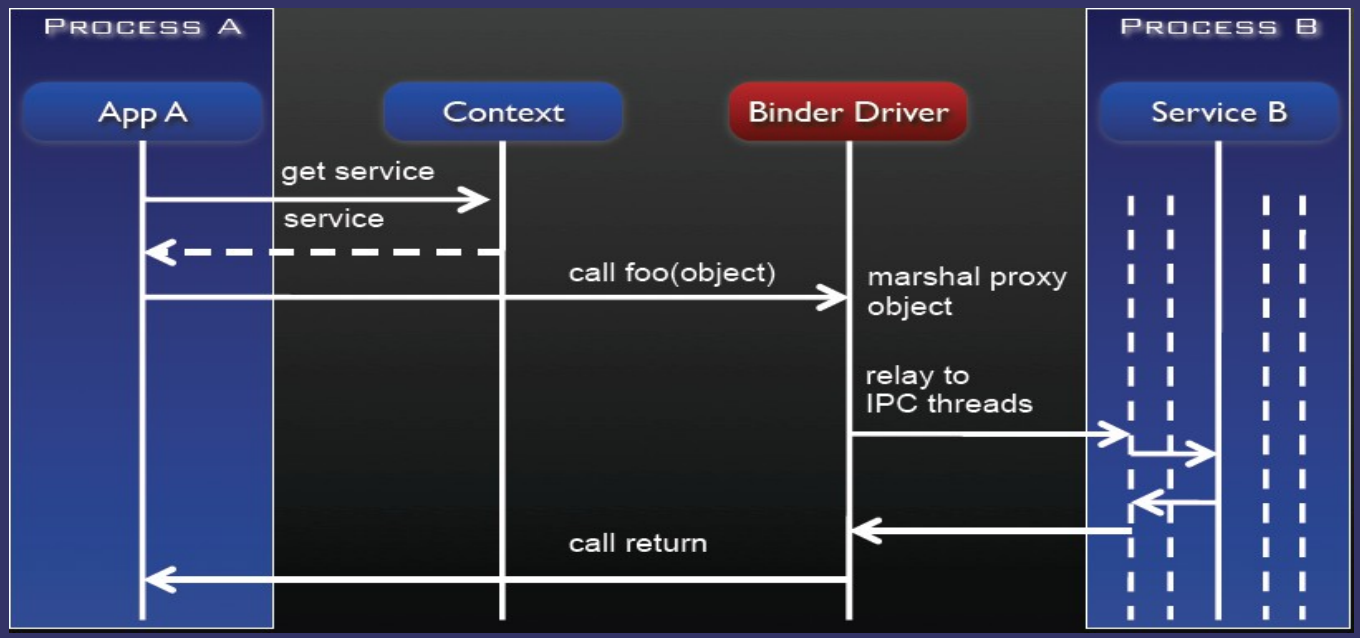
Noyau Android

- Architecture
  - Noyau Linux 2.6, patché, et une glibc spécifique
  - Sysètme de fichiers FAT32
  - Support des protocoles réseaux (TCP/IP, UDP)

- Specificités:
  - Alarm: pour réveiller les périphériques
  - Ashmem: partage de mémoire entre processus
  - Binder: communication inter-processus
  - Power manager, kernel debugger, logger

Noyau Android

- Binder
  - Une communication spécifique interprocessus (IPC) pour renforcer la sécurité
  - Chaque application est lancée dans un processus différent et les applications communiquent via un Binder (communication asynchrone)
  - Le Binder prend en charge le partage de données et les accès concurrents

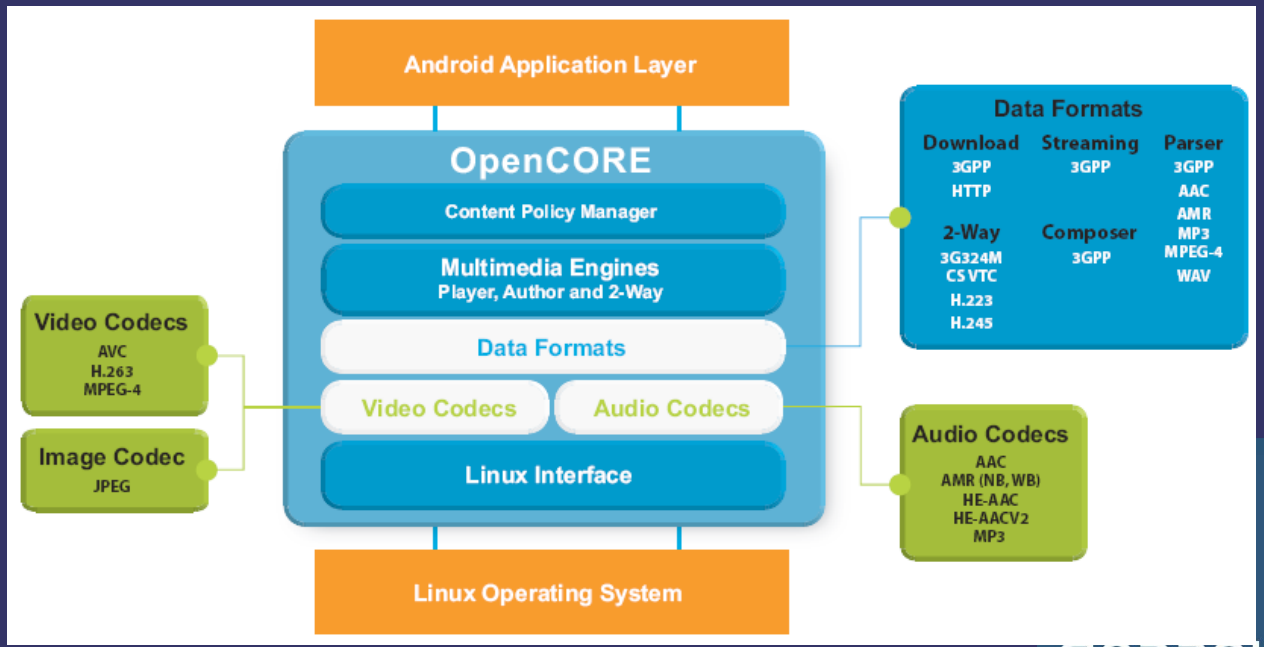


Librairies Android

- Développement d'une libc spécifique (Bionic libc)
  - 200ko de moins que la glibc (une instance par processus)
  - Pas de STL
  - Pas d'exceptions C++

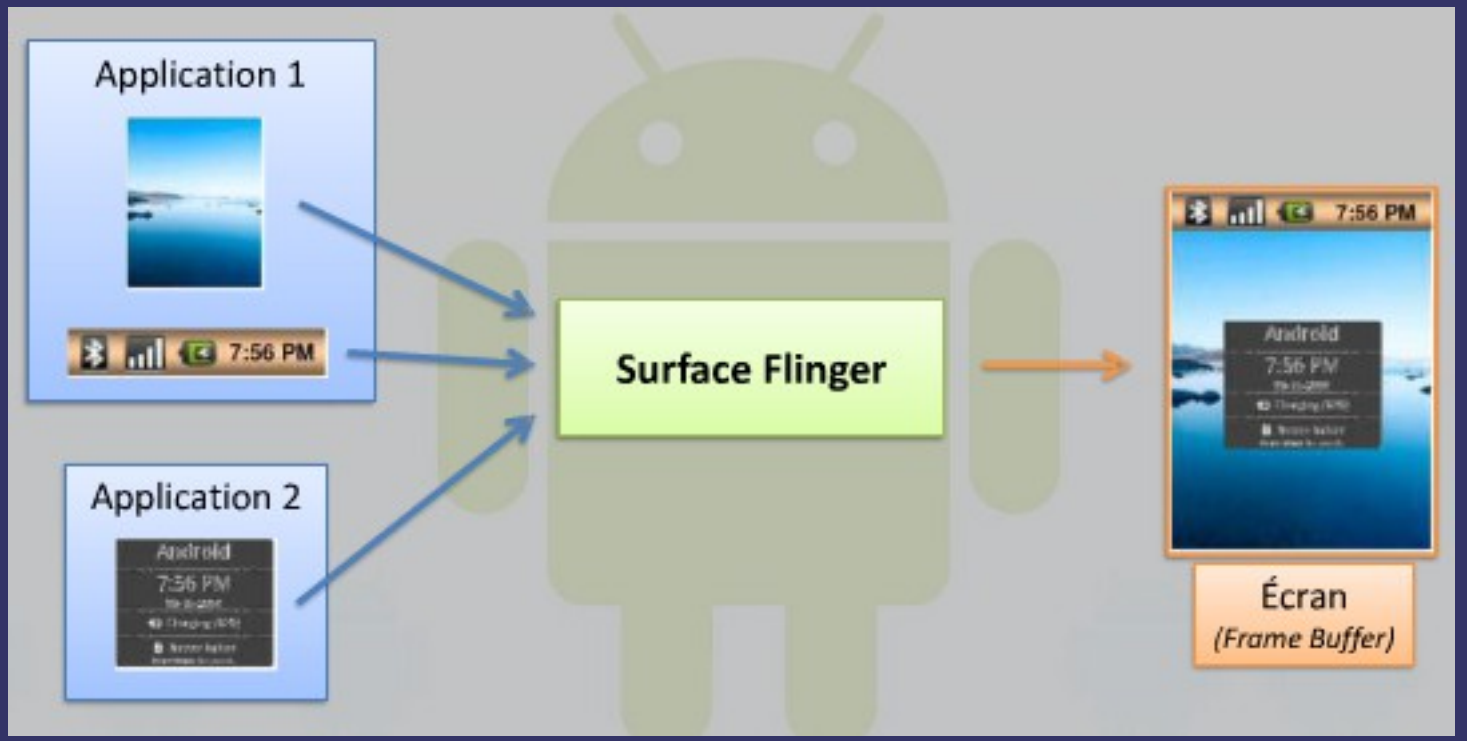
- Webkit: moteur de rendu pour navigateur web

- Media Framework: support des standards audio et video



Exemple de librairie Android

- Le surface Manager



Android Runtime

JVM Dalvik: machine virtuelle Java spécifique pour les applications Android

- Une nouvelle instance de machine virtuelle est lancée pour chaque process → **indépendance des applications**
- Le code est compilé dans des fichiers .dex (Dalvik Executables)
- Dalvik Core class library = base proche de Java 5 SE, construite sur un sous-ensemble de d'Harmony (projet Apache)
  - Optimisation du code (pour des processeurs 250-500 MHz)
  - Empreinte mémoire réduite (20MB disponible pour les applications)

Android Runtime

API Java SE 5 supporté

- java.io - File and stream I/O
- java.lang (except java.lang.management) - Language and exception
- support
- java.math - Big numbers, rounding, precision
- java.net - Network I/O, URLs, sockets
- java.nio - File and channel I/O
- java.security - Authorization, certificates, public keys
- java.sql - Database interfaces
- java.text - Formatting, natural language, collation
- java.util (including java.util.concurrent) - Lists, maps, sets, arrays, collections
- javax.crypto - Ciphers, public keys
- javax.net - Socket factories, SSL
- javax.security (except javax.security.auth.kerberos, javax.security.auth.spi, and javax.security.sasl)
- javax.sound - Music and sound effects
- javax.sql (except javax.sql.rowset) - More database interfaces
- javax.xml.parsers - XML parsing
- org.w3c.dom (but not sub-packages) - DOM nodes and elements
- org.xml.sax - Simple API for XML

API Java SE 5 non supporté

- java.applet
- java.awt
- java.beans
- java.lang.management
- java.rmi
- javax.accessibility
- javax.activity
- javax.imageio
- javax.management
- javax.naming
- javax.print
- javax.rmi
- javax.security.auth.kerberos
- javax.security.auth.spi
- javax.security.sasl
- javax.swing
- javax.transaction
- javax.xml (except javax.xml.parsers)
- org.ietf.*
- org.omg.*
- org.w3c.dom.* (sub-packages)

Android Runtime

JVM Dalvik:

- Une architecture basée sur une machine à registre (type machine de Turing) plutôt qu'une machine à pile
  - Les machines à pile nécessitent des instructions particilières pour charger les arguments
  - Les machines à registre ont des “opcode” plus complexes (spécification des registres source et destination)

- La JVM possède une empreinte mémoire réduite
- La JVM est optimisée de façon à faire tourner plusieurs JVM en paralllèle
- Les performances restent en deça d'un Java Embedded
- Dalvik Turbo Virtual Machine permet d'améliorer les performances

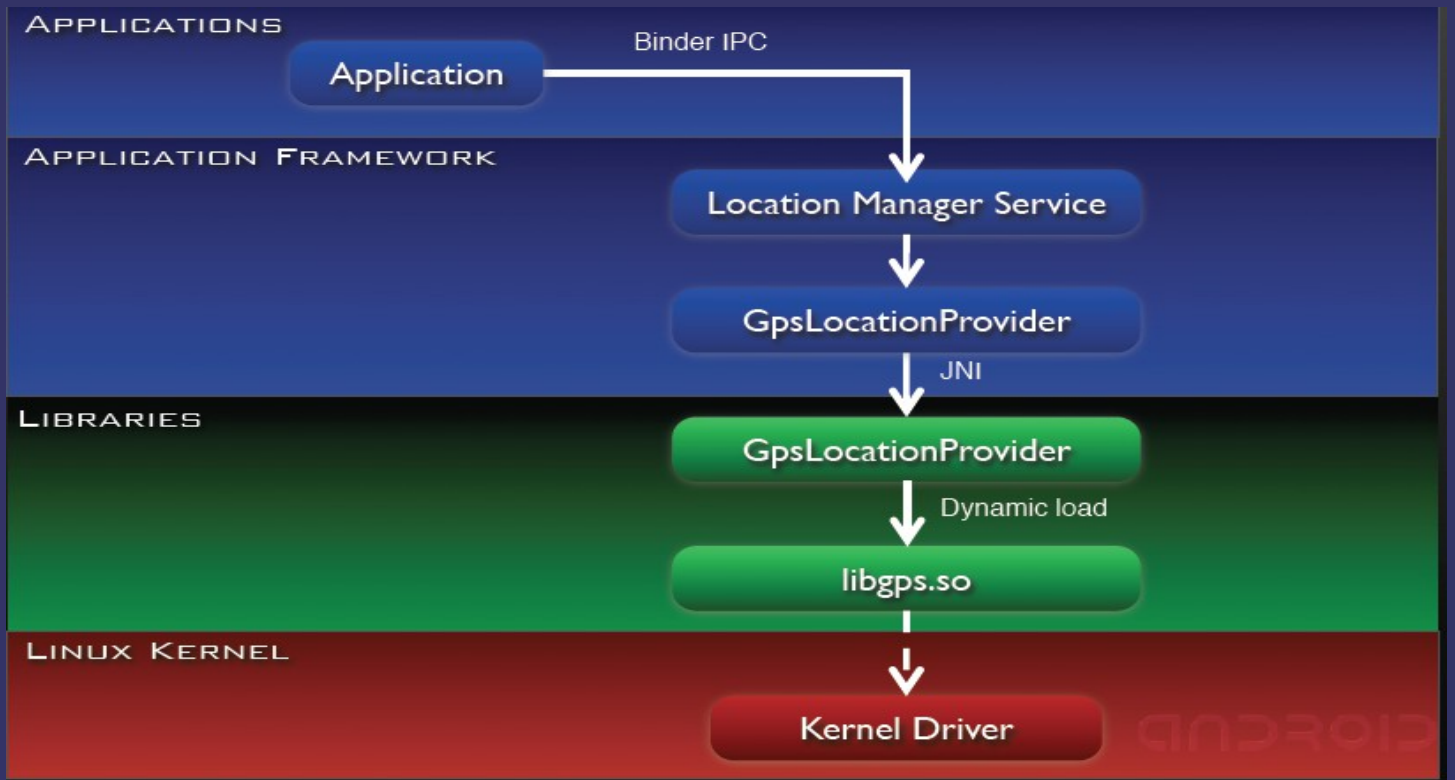
Un framework applicatif

- Les **vues** (“*Views*”) qui permettent de construire une application (widgets + layouts)
  - Listes, grilles, text edit, boutons, browser web,...
- Les “*Content Providers*” qui autorisent l'application à **accèder à des données** d'autres applications, ou à partager leurs données
- Un “*Resource Manager*” offrant l'**accès à des données** autres que du code (images, fichiers d'interface, …)
- Un “*Notification Manager*” qui autorise l'application à afficher des **alertes** dans la barre de statut
- Un “*Activity Manager*” qui prend en charge **le cycle de vie** de l'application et la navigation entre les applications

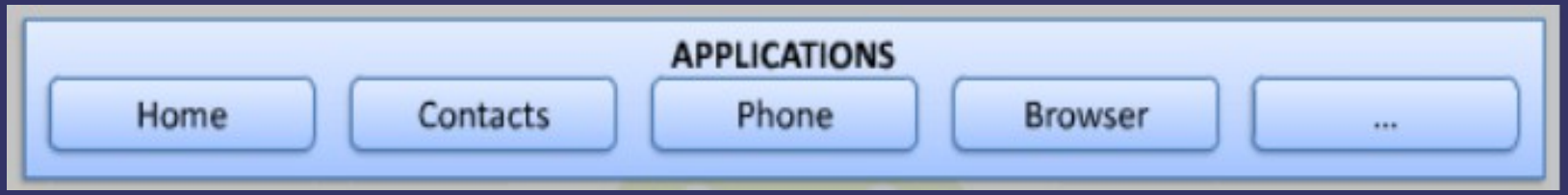
Fonctionnement

Exemple d'appel au service de localisation GPS:

- A partir du contexte, récupération du LocationManager
- Interrogation du GPSLocationProvider
- Appel JNI au code C++ (chargement dynamique du libgps.so)
- Appel au noyau



Demo + Application “Hello World”



- Présentation rapide du SDK et du plugin Eclipse ADT
- Illustration avec l'application “Hello World”

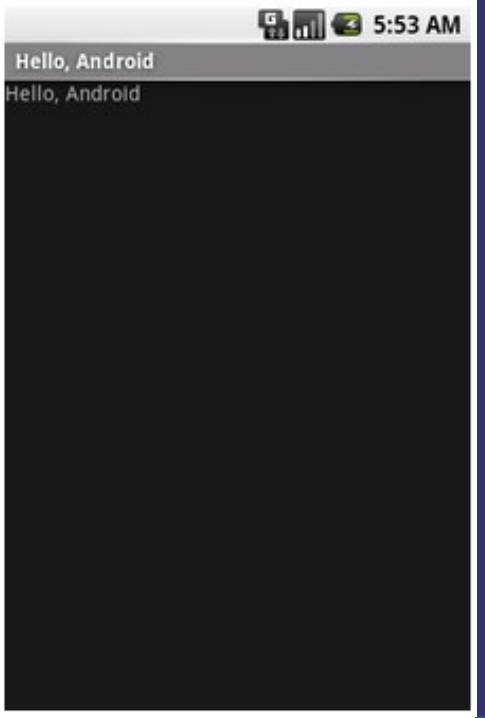
“Hello World 1”

- Par dérivation d'une classe “Activity”:

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```



“Hello World 2”

- Les vues peuvent être définies en XML (et accessibles dans les ressources)

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/textview"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:text="@string/hello"/>
```

  - **android:id** = identifiant unique qui permet ensuite l'accès au composant à partir du code Java
  - **android:text** = texte associé au composant (et défini dans un fichier de ressources, ici “strings.xml”) → permet l'international-isation...

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android! I am a string resource!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```

“Hello World 2”

- L'application se lance alors en utilisant les données générées par les ressources :

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

- **R.layout.main** = accès à la ressource définie dans le package R.layout (fichier de vue “main.xml” compilé)

“Hello World 2”

- exemple du fichier de ressources généré à partir des données XML

```
package com.example.helloandroid;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int textview=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

“Hello World 2”

- Chaque application est pilotée par un fichier de configuration (**AndroidManifest.xml**)
- ...et une arborescence de répertoires contenant les sources et les ressources (automatiquement traités par SDK Android)

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            main.xml
            info.xml
        values/
            strings.xml
```

Grands principes

- Chaque vue est une application (un Activity)
  - Coté vues:
    - développement des IHM statiques en XML
    - développement des IHM dynamiques en Java
  - Coté métier
    - développement essentiellement en JAVA et XML (exceptionnellement C+/C++)
- Les vues communiquent par message asynchrone (qu'on appele Intent)
  - pour le lancement de vues
  - pour l'accès à des services/à des données

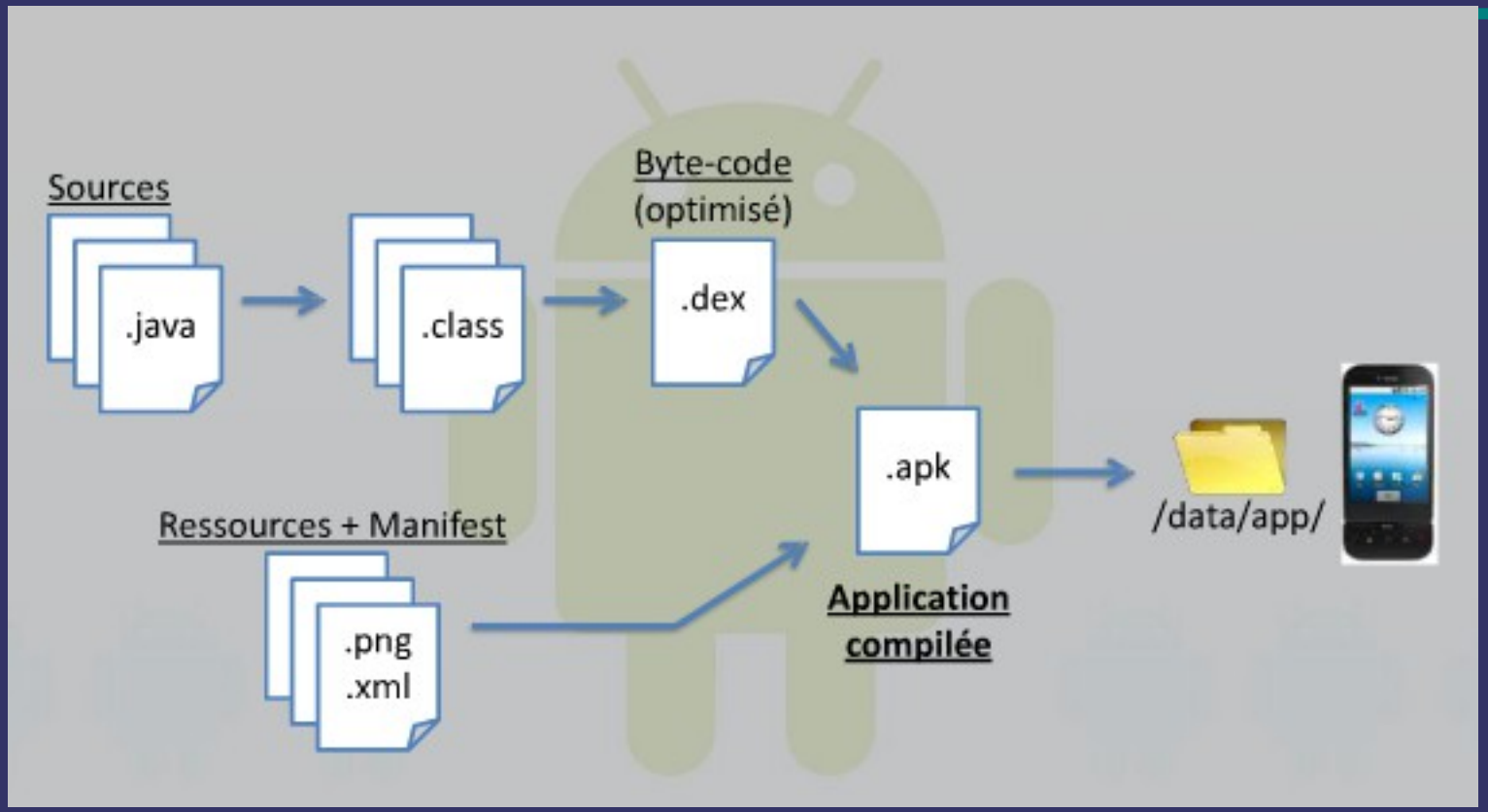
II - Les applications Android

Les applications Android

- Les applications sont écrites en **JAVA** et **paramétrées** par des fichiers de configuration (AndroidManifest.xml)
- Le code Java + ressources sont **compilées** dans un **Android Package** (extension .apk)
  - le package sert au **transport et à l'installation** de l'application
  - une application équivaut à un package (un fichier .apk)
- Chaque application fait tourner **un processus** Linux. L'OS se charge du lancement de l'application et de son cycle de vie

- **IMPORTANT**:
  - Chaque processus possède **sa propre machine virtuelle**
  - Chaque application possède **un identifiant unique Linux** (dans le système de permissions linux):
    - par défaut, une application ne peut accèder qu'à ses propres ressources (icones,...)
    - mais chaque application peut exporter/accèder à d'autres données (par un "*ContentProvider*")

Déploiement d'une application



- “AndoidManifest.xml” déclare les propriétés de l'application:
  - Composants, échanges entre composants, droits...

Android est orienté composants

- Une application est construite à l'aide d'un **ensemble de composants**
- Les applications communiquent de façon **asynchrone** avec des messages (les *“intents”*)

- Chaque composant rend public un ensemble fonctionnalités: ses “***capacities***” et peut utiliser une ou plusieurs “***capacities***” d'un autre composant
  - Évite de re-développer des composants
  - Facilite la maintenance (applicative et évolutive)
  - Le composant demande à démarrer la fonctionnalité d'un autre composant (eg afficher une image, afficher une carte geolocalisée)

- Un composant n'a pas un seul point d'entrée, mais une collection de points d'entrées (“capacities”)

- Android distingue quatre types de composants:
  - Les activités
  - Les services
  - Les récepteurs (“*Broadcast Receivers*”)
  - Les fournisseurs de contenu (“*Content Providers*”)

Composant: “Activity”

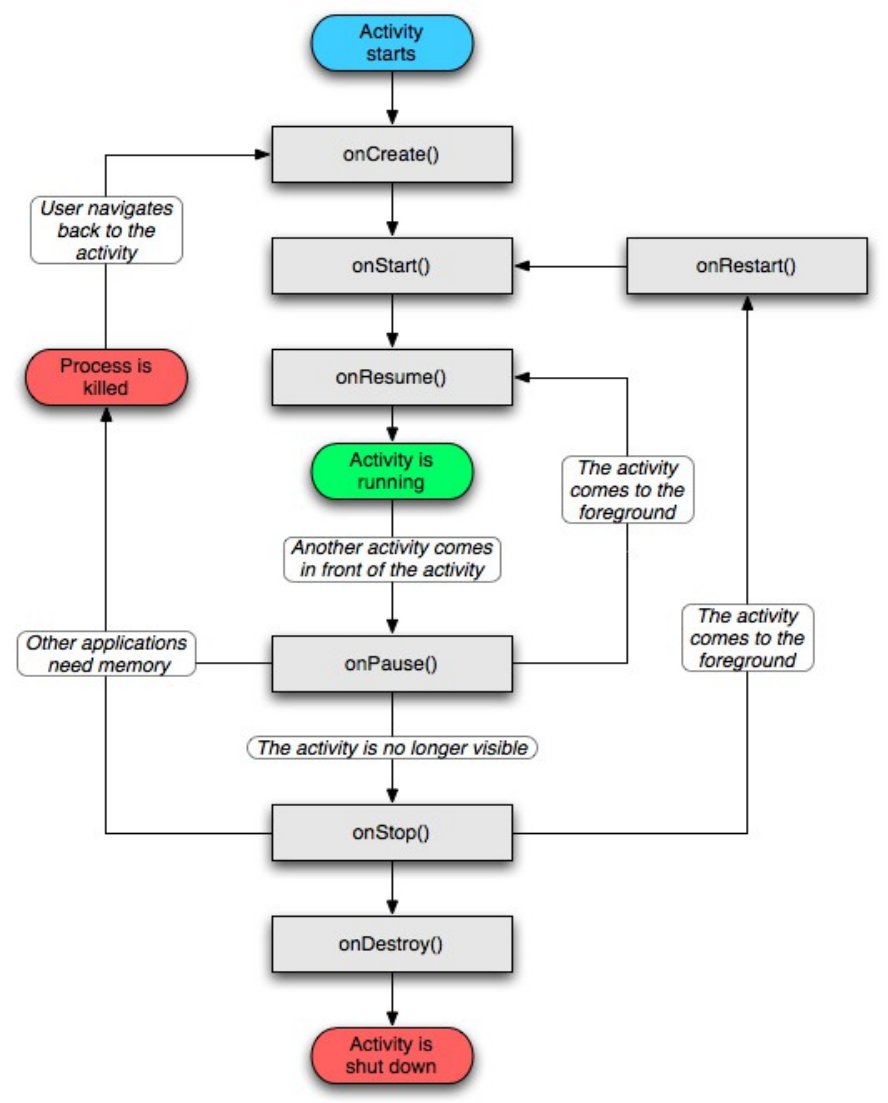
- Un composant “A*ctivity*” représente une application avec une **interface utilisateur**
  - Exemples: affichage d'un ensemble de contacts, saisie d'un e-mail, sélection d'une image (c'est le cas du Hello World)

- Les “A*ctivities*” fonctionnement **individuellement** (et dérivent d'une classe *Activity*)
- Une application peut être constituée d'une ou plusieurs “*activities*”. L'application est en charge de démarrer les suites d'activités

- Chaque “*activity*” possède un espace d'affichage
  - Sur une partie ou tout l'écran
  - Par dessus ou à la place des autres écrans

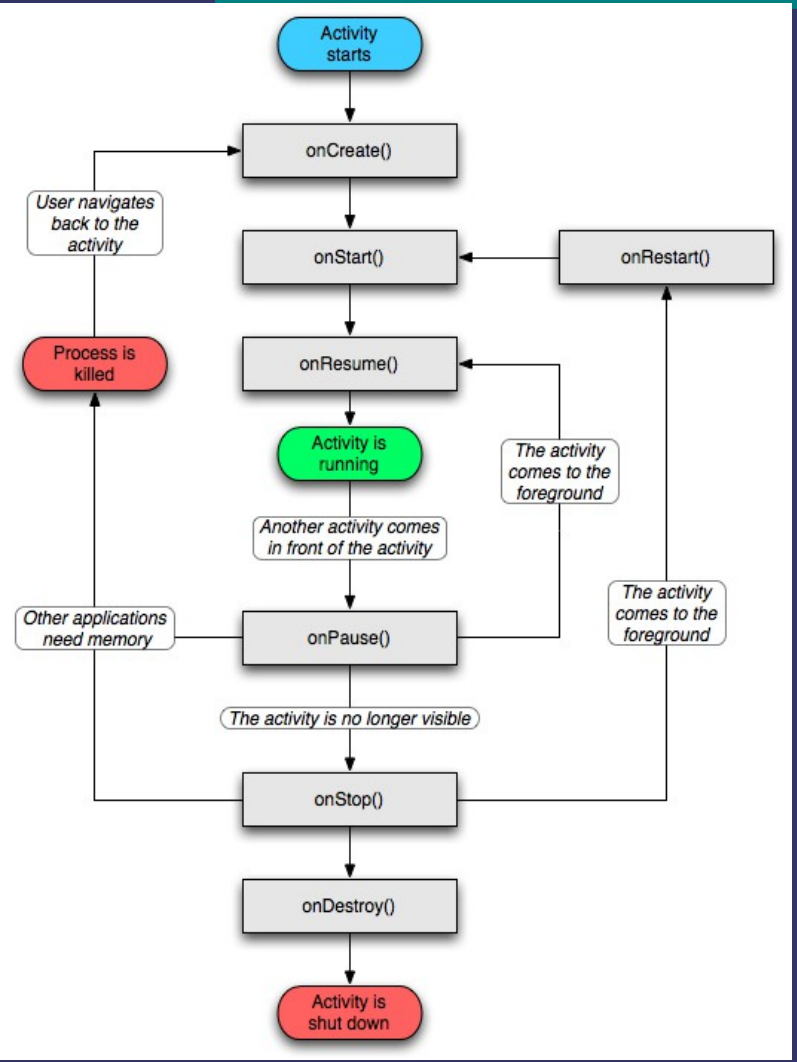
- L'espace d'affichage est organisé comme une **hiérarchie de vues** (“*Views*”)
  - Chaque vue contrôle une partie de l'écran et contrôle les entrées utilisateurs et les affichages
  - Une vue est soit un “*widget*”, soit un “*layout*” (cf partie Android et les Vues)

Cycle de vie des Activity



- Une activité a 3 états:
  - Active (affiché à l'écran)
  - En pause (visible, mais non accessible à l'utilisateur, e.g. en cas de pop up)
  - Arrêtée (non visible, mais état maintenu tel quel)
- Le système peut décider de détruire et de relancer les activités arrêtées (eg. pour libérer de la mémoire)
- Trois cycles à prendre en compte
  - le cycle de **l'application** (onCreate() → onDestroy)
  - le cycle **visible** (onStart() → on Stop())
  - le cycle **actif** (onResume() → onPause())

Cycle de vie des Activity



- Méthodes de cycle de vie d'**Activity**:
  - **onCreate(Bundle state)**: démarre l'application
  - **onRestart()**: relance l'application après un arrêt
  - **onStart()**: relance l'application après un arrêt
  - **onResume()**: relance l'application après changement de focus
  - **onPause()**: met en pause (perte de focus)
  - **onStop()**: arrête l'activité (plus visible)
  - **onDestroy()**: détruit l'activité
- **onSaveInstanceState(Bundle state)**: permet la sauvegarde du contexte
  - par défaut l'OS sauvegarde la vue courante et l'état des widgets
  - en surchargeant la méthode, on peut redéfinir/étendre la sauvegarde
  - l'objet Bundle est un mapping clé/objet qui permet la sauvegarde du contexte
- Le contexte est recharché dans **onCreate()** ou alors avec o**n-RestoreInstateState()**
- Un changement de langue de l'OS, un changement de fuseau, un changement d'orientation (portrait/paysage) détruit et relance l'application

Composant: “Services”

- Un composant “*Services*” n'a pas de vue
- Un service est un processus en taĉhe de fond qui offre des *“capacities”*:
  - e.g. synchronisation de données

- Le service peut être démarré et/ou arrêté par une “*Activity*”
- Le service peut être controlé en continu (accès ou fourniture de données, contrôle du service)

- **Exemple:** un lecteur de musique
  - Démarré par une activité tiers
  - Tourne en tâche de fond
  - Est contrôlé par ses “*capacities*” (lecture/pause/suivant/etc...)

Composant: “BroadcastReceiver”

- Ces composants recoivent et traitent des messages broadcastés à l'ensemble des applications:
  - Changement de time-zone, batterie faible, changement de langue, …
- Ces composants peuvent émettre les messages
- Ces composants n'ont pas de vues
  - ...mais peuvent démarrer une interface (“*Activity*”)
  - ...ou interagir avec le “*NotificationManager*”
- Chaque récepteur dérive de “*BroadcastReceiver*”

Composant: “Content Providers”

- Permet de rendre disponibles des données d'une application à d'autres applications (e.g. carnet d'adresses)
  - données stockées dans le système de fichiers local
  - données stockées dans une base **SQLite**
  - ou tout autre source de données (e.g. Service web)

- Le “*Content Provider*”:
  - déclare des points d'entrées pour les données
  - s'interface avec un “*ContentResolver*”
  - les autres applications interrogent le “*ContentResolver*” (et non le “*ContentProvider*”)

Activation des composants:

- L'OS assure le **lancement des composants** lors de l'appel à leurs “capacities”

- Activation:
  - Les “*ContentProviders*” sont activés directement par les “*ContentResolver*” sur la requête d'une application (e.g. accès à une entrée d'un carnet d'adresse)
  - Les autres composants sont activés par des messages asynchrones: les “*Intents*” (**Command Pattern**)
  - “*Intent*” contient le nom de l'activité et les données relatives au message

Communication asynchrone par "Intents"

- Les applications Android tournant sur des instances différentes de JVM, il est nécessaire de les faire communiquer

- Android propose un système de communication asynchrone (les Intent)
  - **Command Pattern** (transformation d'un appel de méthode en objet pour une execution ultérieure)
- Les intents contiennent :
  - les commandes à executer
  - les données à transmettre
  - le composant cible ou pas ...

Mécanique des “Intent”

- Les “Intent” permettent d'établir des liens en temps-réel entre les applications
  - En respectant les type des intent (broadcast intent n'est transmis qu'aux broadcast receivers, …)
  - Les “Intent” peuvent viser:
    - une application en particulier (**Intent Explicite**)
    - une application non nommées (**Intent Implicite**)
  - Un système de Filtrage (IntentFilter) permet aux applications de filters les Intent les concernant

Le principe des “Intent” (Activity)

- Une “A*ctivity*” se démarre en envoyant un intent avec la méthode:
  - **startActivity(Intent, ...)**
  - **startActivityForResult(Intent, ...)**
- L'activité récupère l'intent avec la méthode:
  - **getIntent()**
- Si un résultat est attendu, celui-ci est transmis par Intent, et récupérable par l'appelant avec:
  - **onActivityResult()**

Le principe des “Intent” (Service)

- Un service est démarré (ou contrôlé) en passant en paramètre un Intent à : **Context.startService()**

- Un intent peut lier un service à une application (pour maintenir une connection en continu):
  - En transmettant un Intent à **Context.bindService()**
  - En appelant des méthodes spécifiques définies par le service
  - Exemple: lecteur de musique

Le principe des “Intent” (Broadcast)

- Une application peut démarrer un broadcast en transmettant un Intent à :
  - Context.sendBroadcast(Intent, ...)
  - Context.sendOrderedBroadcast(Intent, ...)
  - Context.sendStickyBroadcast(Intent, ...)

- Les composants “broadcastés” récupèrent l'intent par la méthode:
  - onReceive(Intent)

Un exemple d'Intent (1)

- Lancement du navigateur Web (le layout)

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

<Button android:id="@+id/Button01" android:layout_width="wrap_content
" android:layout_height="wrap_content" android:text="Open Browser" an
droid:onClick="openBrowser"></Button>
</LinearLayout>
```

Un exemple d'Intent (2)

- Lancement du navigateur Web (le code)

```
package de.vogella.android.intent.browser;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;

public class OpenBrowser extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
        }

        public void openBrowser(View view) {
                Intent i = new Intent("android.intent.action.VIEW", U
ri.parse("http://www.vogella.de"));
                startActivity(i);
        }
}
```

- On ne sais pas **quelle activité** sera executée (c'est un **Intent implicite**)
  - ici, toute application acceptant une commande de type VIEW sur une URL peut être lancée

Un autre exemple d'Intent (1)

- Lancement d'une activité en particulier (Intent Explicite)

```
public class MainActivity extends Activity implements View.OnClickListener {
...
  public void onClick(View view){
        if(view == mButton1){
            // creation d'un intent pour appeler une autre activité (SecondaryActivity)
            Intent intent = new Intent(getApplicationContext(),SecondaryActivity.class);

            // ajout de données supplémentaires dans l'intent
            intent.putExtra(“myKey”, “Hello second activity”);

            //lancement de la seconde activité, en demandant un code retour
            startActivityForResult(intent, 0);
        }
    }
}
```

Un autre exemple d'Intent (2)

- .... et dans la seconde activité...

```
public class SecondaryActivity extends Activity  {
...
   public void onCreate(Bundle savedInstanceState){
        ....
        // l'activité est démarrée (puisque l'envoi d'intent l'a démarrée)

         Intent intent = new Intent();

         // ajout de données supplémentaires dans l'intent
         intent.putExtra(“returnKey”, “Hello first activity”);

         // envoi du resultat
         setResult(RESULT_OK,intent);

        // arret de l'activity ici
        finish();
    }
}
```

Un autre exemple d'Intent (3)

- .... et de retour dans la première on sur-charge la méthode onActivityResut(...)

```
public class MainActivity extends Activity implements View.OnClickListener {
…

  protected void onActivityResult(int requestCode, int resultCode, Intent intent){
        super.onActivityResult(requestCode, resultCode, intent);
        Bundle extras = intent.getExtras();
        mEditText1.setText(extras != null ? extras.getString("returnKey"):"nothing returned");
    }
}
```

Les Objets “Intent”

- Un intent est un objet contenant les données suivantes:
  1) Le nom du composant destinataire (optionel):
     Exemple: “**com.example.project.app.FreneticActivity**”

     - où **com.example.project.app** représente le nom du package défini dans le fichier Manifest

     - et **FreneticActivity** le nom de la classe (ici dérivant d'Activity)
     - Le nom est optionel:
       - s'il existe, l'activité correspondante est démarrée si nécessaire et le message est traité
       - s'il n'existe pas, Android détermine une application cible à partir des autres informations de l'Intent (Intent Resolution)
     - Le nom est spécifié par **setComponent()**, **setClass()** ou **setClassName()**

Les Objets “Intent”

2) Une action a éxecuter (ou l'action qui s'est déroulée dans le cas d'un broadcast intent)

- Il existe des actions par défaut (constantes)

Constante	Composant cible	Action
ACTION_CALL	Activity	Démarre une conversation
ACTION_EDIT	Activity	Affiche des données à éditer
ACTION_MAIN	Activity	Démarre une activité
ACTION_SYNC	Activity	Synchronise les données avec un serveur
ACTION_BATTERY_LOW	Broadcast receiver	Informe du faible niveau de batterie
ACTION_HEADSET_PLUG	Broadcast receiver	Connexion/Déconnexion d'un casque
ACTION_SCREEN_ON	Broadcast receiver	Mise en route de l'écran
ACTION_TIMEZONE_CHANGED	Broadcast receiver	Changement de time zone

Les Objets “Intent”

3) Un champ de données:

- Uniquement de type URI pointant sur les données
- Avec un mimetype caractérisant la nature de la donnée

Exemples:

- ACTION_EDIT content:
- ACTION_CALL tel:
- ACTION_VIEW http:

- Accès par setData() / setType()

Les Objets “Intent”

4) Un champ ***Category*** qui définit le type de composant qui est concerné par l'”Intent”

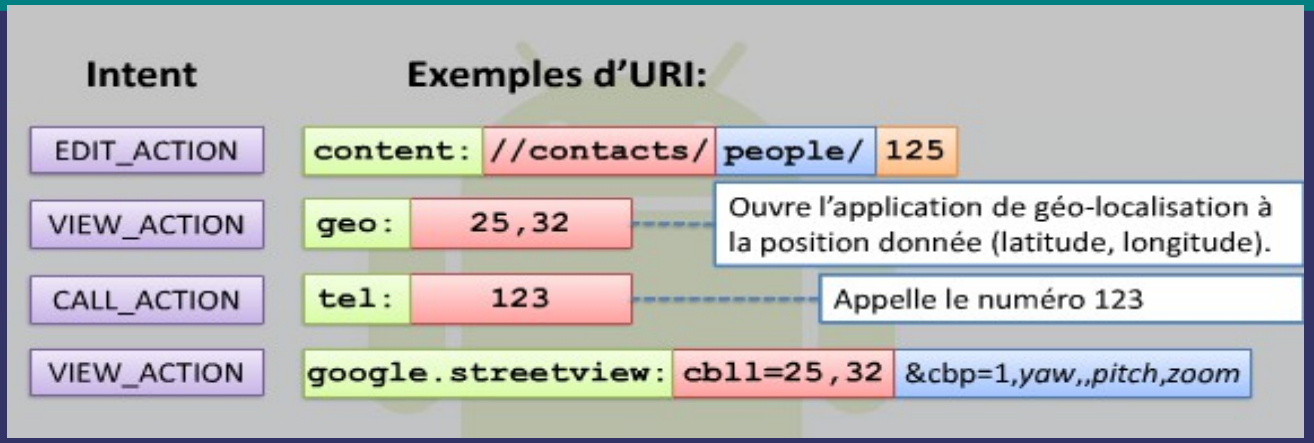
- Un Inent peut avoir plusieurs catégories (cf addCategory(), deleteCategory(), getCategories())

5) Un champ ***Extras*** qui contient des couples clé/valeur pour stocker des informations additionnelles

- eg. ACTION_HEADSET_PLUG state=plugged

6) Un champ ***Flags*** qui contient des options spécifiques aux besoins des applications

Exemples d'Intent



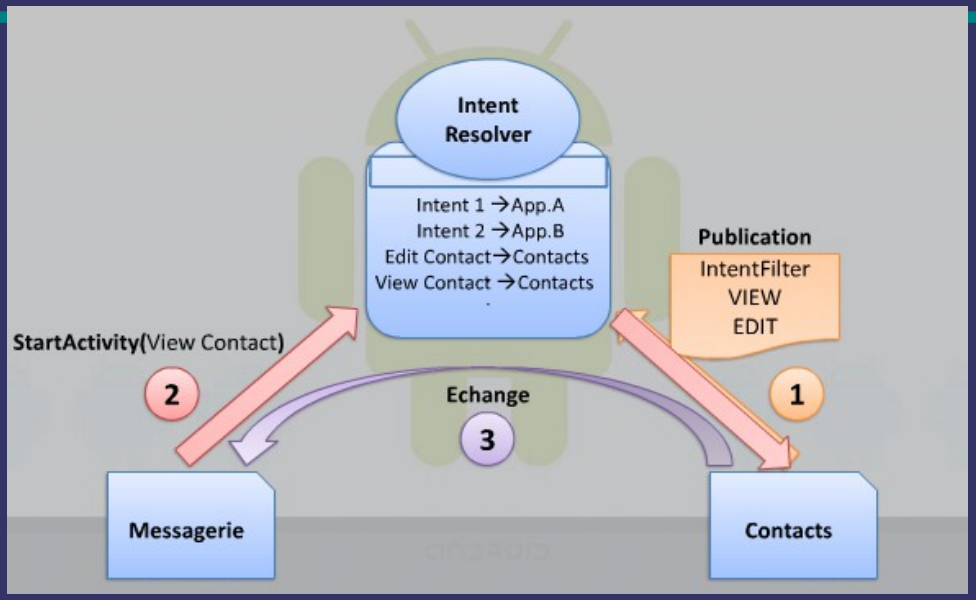
- L'URI peut référencer une entrée dans une table définie par un ContentProvider (_ID=125)

- L'URI peut appeler des services standard:
  - Tel: execute l'appel
  - Geo: geolocalise

Résolution des “Intent”

- On rappelle qu'il existe deux classes d'Intent
  - **Intent explicites**: le composant cible est identifié par son nom
    - généralement réservés aux communications internes (entre les composants d'une meme application)
    - ...mais les applications externes ne sont pas nécessaire-ment livrées avec leur nom
  - **Intent implicites**
    - Ne nomment pas de cibles (généralement réservés à l'acti-vation de composants extérieurs)
    - Android détermine:
      - le meilleur composant pour prendre l'intent en charge (activité ou ser-vice)
      - les meilleurs composants (broadcast receivers)
    - Technique:
      - chaque composant “appelé” possède des filtres (Intent Filters)
      - les filtres définissent les capacités du composant et limitent les types d'intent qu'ils peuvent traiter

Intent Filters



1) A la création, le contact déclare ses données (ContentProvider)
2) La messagerie démarre l'activité View Contact
    - L'Intent Resolver établit le lien avec le CP Contacts
    - Les données sont récupérées dans le CP Contacts
3) Et retransmises à la messagerie

Intent Filters

- A chaque composant est associé des Intent Filters (IF):
  - Un IF définit une capacité du composant et une restriction sur les intent traités par cette capacité
  - Un IF renseigne les champs action, data et category
- Le composant n'est activé que lorsque l'intent passe au travers de l'Intent Filter

- Les IF sont définis dans le fichier manifest (AndroidManifest.xml)

- Les trois catégories de filtres portent sur les trois champs:
  - Action
  - Category
  - Data

Filtrage par les actions

- Exemple

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

- Pour passer le filtre l'action de l'INTENT doit être identique à l'action du filtre; ici le filtre accepte tous les intent dont l'action est SHOW_CURRENT, SHOW_RECENT, SHOW_PENDING
- Il faut au moins une action par filtre (sinon aucun intent n'est transmis)
- Un intent sans action est automatiquement transmis! (sauf si le filtre ne déclare aucune action)

Filtrage par les catégories

- Exemple:

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```

  - Chaque catégorie de l'intent à filtrer doit nécessairement apparaître dans l'intent filter pour passer le filtre.
  - Les intent implicites ont par défaut une catégorie “android.intent.category.DEFAULT”, il faut donc ajouter cette catégorie au filtre.

Filtrage par les données

- Exemple:

```
<intent-filter . . . >
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    . . .
</intent-filter>
```

- Chaque élément nomme un MIME type et une URI
- L'URI est décrite par: scheme, port, host et path
  - content://com.example.project:200/folder/etc
    - Scheme = “content”
    - Host = “com.example.project”
    - Port = “200”
    - Path = “folder/etc”
- La comparaison s'effectue sur chacun des attributs présents dans le filtre
- Filtres et Intent peuvent utiliser des joker (*): eg 'image/*' ou 'video/*'

Exemples:

```
<data android:mimeType="image/*" />
```

- Autorise le composant à traiter tout type d'image (stockée localement, via un content provider ou via le web)

```
<data android:scheme="http" android:type="video/*" />
```

- Autorise le composant à traiter tout type de video (provenant d'un site internet)

```
<intent-filter . . . >
    <action android:name="code android.intent.action.MAIN" />
    <category android:name="code android.intent.category.LAUNCHER" />
</intent-filter>
```

- Autorise le composant être démarré à partir d'un autre composant (MAIN) ou à être lancé par le lanceur d'application (LAUNCHER)

AndroidManifest.xml

Chaque application doit possèder un fichier AndroidManifest.xml à la racine:

- Détermine la classe java relative à l'application
- Décrit les composant de l'application (activities, services, borad-casts, et content providers)
- Décrit les capacités des applications (Intent)
- Décrit les permissions de l'application (eg. Accès à des données protégées de l'API)
- Déclare des options de profiling (pour le développement et les tests)
- Déclare les librairies liées à l'application

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>

        <uses-library />

    </application>

</manifest>
```

AndroidManifest.xml

- Les éléments <manifest> et <application> sont indispensables
- Les noms des applications (Activity, Service, BroadcastReceiver et ContentProvider) correspondent à des classes Java de l'application

```
<manifest . . . >
    <application . . . >
        <service android:name="com.example.project.SecretService" . . . >
            . . .
        </service>
        . . .
    </application>
</manifest>
```

- Les ressources sont identifiées par `@[package:]type:name`

ex:

```
<activity android:icon="@drawable/smallPic" . . . >
```

L'icone “smallPic”(.png) est de type drawable (et est stockée dans le répertoire drawable)

BILAN Architecture Android

- Une approche par composants:
  - Distinction des applications en fonction de leurs services (avec ou sans vue, broadcastés ou non,...)
- Une communication asynchrone par *“Intent”*
  - Permet l'appel à des ressources ou l'appel à des fonctionnalités
- Cycle de vie applicatif spécifique
  - Applications en pause/resume/stop

Application Géo Localisation

Le code:

```
public class GPS extends Activity
{
    private LocationManager lm;
    private LocationListener locationListener;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---use the LocationManager class to obtain GPS locations---
        lm = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);

        locationListener = new MyLocationListener();

        lm.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            0,
            0,
            locationListener);
    }
}
```

Application Géo Localisation

- Le listener de positions...

```
private class MyLocationListener implements LocationListener
{
    @Override
    public void onLocationChanged(Location loc) {
        if (loc != null) {
            Toast.makeText(getBaseContext(),
                "Location changed : Lat: " + loc.getLatitude() +
                " Lng: " + loc.getLongitude(),
                Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onProviderDisabled(String provider) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onStatusChanged(String provider, int status,
        Bundle extras) {
        // TODO Auto-generated method stub
    }
}
```

Application Géo Localisation

- Affichage d'une carte par le widget **com.google.android.maps.MapView**

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

<com.google.android.maps.MapView
    android:id="@+id/mapview1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="apisamples" />

</LinearLayout>
```

Application Géo Localisation

- Dans Activity.onCreate(), il faut récupérer le contrôlleur de la mapView

```
mapView = (MapView) findViewById(R.id.mapview1);
mc = mapView.getController();
```

- Dans LocationListener.onLocationChanged(), il faut transmettre les coordonnées au controlleur

```
GeoPoint p = new GeoPoint(
        (int) (loc.getLatitude() * 1E6),
        (int) (loc.getLongitude() * 1E6));
mc.animateTo(p);
mc.setZoom(16);
mapView.invalidate();
```

mapView.invalidate demande le rafraichissement de la carte

Application Géo Localisation

- Et le fichier Manifest ...

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.GPS">

    <uses-permission
        android:name="android.permission.INTERNET" />

    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">

        <uses-library android:name="com.google.android.maps" />

        <activity android:name=".GPS" 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>
```

Ajout de l'accéléromètre

- En dérivant de **SensorEventListener**
  - En implémentant **onSensorChanged(SensorEvent event)**

```
public class SensorActivity extends Activity, implements SensorEventListener {
    private final SensorManager mSensorManager;
    private final Sensor mAccelerometer;

    public SensorActivity() {
        mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }

    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    public void onSensorChanged(SensorEvent event) {
    }
}
```

Ajout de l'accéléromètre

- En dérivant de **SensorEventListener**
  - En implémentant **onSensorChanged(SensorEvent event)**

```
public void onSensorChanged(SensorEvent event)
{
    // alpha is calculated as t / (t + dT)
    // with t, the low-pass filter's time-constant
    // and dT, the event delivery rate

    final float alpha = 0.8;

    gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
    gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
    gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

    linear_acceleration[0] = event.values[0] - gravity[0];
    linear_acceleration[1] = event.values[1] - gravity[1];
    linear_acceleration[2] = event.values[2] - gravity[2];
}
```

Encore de nombreuses possibilités

- Réalité augmentée sur Android
  - Beaucoup de frameworks disponibles dont Qualcomm AR:

    https://developer.qualcomm.com/develop/mobile-technologies/augmented-reality

Encore de nombreuses possibilités

- Jeux 3D
  - Machines multi-core, multi-GPU
  - Avec sorties HDMI

Conclusion

- Domaine moteur
  - Evolution rapide des applications embarquées
  - Contexte de répartition/ubiquité des données
  - Interconnexion des réseaux / données / usages
  - Cadres professionnels et personnels

- Plateforme complexe mais documentée
  - Les exemples et tutoriels sont simples et pédagogiques
  - La maîtrise de la plateforme nécessite un investissement important
  - Difficulté de **l'hétérogénéité** des plateformes (IHM,...) et de la **malléabilité** des interfaces