

Cours Android

Développement et API

Romain Raveaux¹

¹Laboratoire LI – Polytech'Tours
romain.raveaux at univ-tours.fr

Mars 06-03, 2012

MCours.com

Sommaire

- 1 Organisation du module
- 2 Généralité
- 3 Le Système
- 4 Fonctionnement des Applications

Romain Raveaux

- **Doctorat en informatique**
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
- **Compagnon de voyage**
- Enseignements Polytech'Tours (DI et D2i)
- **Compagnon de voyage**
- **Compagnon de voyage**
- **Compagnon de voyage**

Romain Raveaux

- Doctorat en informatique
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
 - Comparaison de graphes
- Enseignements Polytech'Tours (DI et D2i)
 - Algorithmes et Graphes
 - Discrets et Graphes
 - Réseaux et Graphes

Romain Raveaux

- Doctorat en informatique
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
 - Comparaison de graphes
- Enseignements Polytech'Tours (DI et D2i)

Romain Raveaux

- Doctorat en informatique
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
 - Comparaison de graphes
- Enseignements Polytech'Tours (DI et D2i)
 - Développement Android
 - Parcours de cours
 - Réseaux et Télécom

Romain Raveaux

- Doctorat en informatique
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
 - Comparaison de graphes
- Enseignements Polytech'Tours (DI et D2i)
 - Développement Android
 - Bus de terrain
 - Parcours de graphes

Romain Raveaux

- Doctorat en informatique
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
 - Comparaison de graphes
- Enseignements Polytech'Tours (DI et D2i)
 - Développement Android
 - Bus de terrain
 - Parcours de graphes

Romain Raveaux

- Doctorat en informatique
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
 - Comparaison de graphes
- Enseignements Polytech'Tours (D1 et D2i)
 - Développement Android
 - **Bus de terrain**
 - Parcours de graphes

Romain Raveaux

- Doctorat en informatique
- 2 Masters: (a) Réseaux et Télécommunications; (b) Maths appliquées
- Recherche au Laboratoire LI
 - Comparaison de graphes
- Enseignements Polytech'Tours (D1 et D2i)
 - Développement Android
 - Bus de terrain
 - **Parcours de graphes**

Module Android

Contraintes des systèmes embarqués

- **Généralité Android**
- Organisation du système
- Android SDK
- Application

Module Android

Contraintes des systèmes embarqués

- Généralité Android
- **Organisation du système**
- Android SDK
- Application

Module Android

Contraintes des systèmes embarqués

- Généralité Android
- Organisation du système
- **Android SDK**
- Application

Module Android

Contraintes des systèmes embarqués

- Généralité Android
- Organisation du système
- Android SDK
- **Application**

Volume horaire

- 2h CM
- 6h TP

Volume horaire

- 2h CM
- 6h TP

Plan des TPs

- **Getting started (Hello World!, Débugger) (2h)**
- Communication inter-processus : Intent (2h)
- Communication inter-processus : Services (2h)
- Traitement de la vidéo
- Applications natives (C++)
- Google Maps API (Géo-localisation)
- Projets : Smart Image Gallery
- Projets : Géolocalisation des Stations Vélo

Plan des TPs

- Getting started (Hello World!, Débugger) (2h)
- **Communication inter-processus : Intent (2h)**
- Communication inter-processus : Services (2h)
- Traitement de la vidéo
- Applications natives (C++)
- Google Maps API (Géo-localisation)
- Projets : Smart Image Gallery
- Projets : Géolocalisation des Stations Vélo

Plan des TPs

- Getting started (Hello World!, Débbugger) (2h)
- Communication inter-processus : Intent (2h)
- **Communication inter-processus : Services (2h)**
- Traitement de la vidéo
- Applications natives (C++)
- Google Maps API (Géo-localisation)
- Projets : Smart Image Gallery
- Projets : Géolocalisation des Stations Vélo

Plan des TPs

- Getting started (Hello World!, Débugger) (2h)
- Communication inter-processus : Intent (2h)
- Communication inter-processus : Services (2h)
- **Traitement de la vidéo**
 - Applications natives (C++)
 - Google Maps API (Géo-localisation)
 - Projets : Smart Image Gallery
 - Projets : Géolocalisation des Stations Vélo

Plan des TPs

- Getting started (Hello World!, Débugger) (2h)
- Communication inter-processus : Intent (2h)
- Communication inter-processus : Services (2h)
- Traitement de la vidéo
- **Applications natives (C++)**
 - Google Maps API (Géo-localisation)
 - Projets : Smart Image Gallery
 - Projets : Géolocalisation des Stations Vélo

Plan des TPs

- Getting started (Hello World!, Débugger) (2h)
- Communication inter-processus : Intent (2h)
- Communication inter-processus : Services (2h)
- Traitement de la vidéo
- Applications natives (C++)
- **Google Maps API (Géo-localisation)**
- Projets : Smart Image Gallery
- Projets : Géolocalisation des Stations Vélo

Plan des TPs

- Getting started (Hello World!, Débugger) (2h)
- Communication inter-processus : Intent (2h)
- Communication inter-processus : Services (2h)
- Traitement de la vidéo
- Applications natives (C++)
- Google Maps API (Géo-localisation)
- Projets : Smart Image Gallery
- Projets : Géolocalisation des Stations Vélo

Plan des TPs

- Getting started (Hello World!, Débugger) (2h)
- Communication inter-processus : Intent (2h)
- Communication inter-processus : Services (2h)
- Traitement de la vidéo
- Applications natives (C++)
- Google Maps API (Géo-localisation)
- Projets : Smart Image Gallery
- Projets : Géolocalisation des Stations Vélo

Les règles

- Ne pas perturber le cours:
 - Silence
 - Rendre les comptes rendus de TP à temps.

Les règles

- Ne pas perturber le cours:
 - Silence
 - **Rendre les comptes rendus de TP à temps.**

L'évaluation

- Les comptes rendus

Bibliographie

- <http://developer.android.com/index.html>
- <http://www.tutomobile.fr/category/tutorial-android/>
- <http://android.developpez.com/cours/>
- Linux Mag
- Programming Android de Zigurd Mednieks, Laird Dornin et G. Blake Meike

Bibliographie

- <http://developer.android.com/index.html>
- <http://www.tutomobile.fr/category/tutorial-android/>
- <http://android.developpez.com/cours/>
- Linux Mag
- Programming Android de Zigurd Mednieks, Laird Dornin et G. Blake Meike

Bibliographie

- <http://developer.android.com/index.html>
- <http://www.tutomobile.fr/category/tutorial-android/>
- <http://android.developpez.com/cours/>
- Linux Mag
- Programming Android de Zigurd Mednieks, Laird Dornin et G. Blake Meike

Bibliographie

- <http://developer.android.com/index.html>
- <http://www.tutomobile.fr/category/tutorial-android/>
- <http://android.developpez.com/cours/>
- **Linux Mag**
- Programming Android de Zigurd Mednieks, Laird Dornin et G. Blake Meike

Bibliographie

- <http://developer.android.com/index.html>
- <http://www.tutomobile.fr/category/tutorial-android/>
- <http://android.developpez.com/cours/>
- Linux Mag
- **Programming Android de Zigurd Mednieks, Laird Dornin et G. Blake Meike**

Présentation Générale



ANDROID

Qu'est-ce que c'est ?

- Souvent présenté comme l'alternative de Google à l'iPhone
- Système d'exploitation pour terminaux mobiles
- Basé sur Linux
- Open Source (licence Apache)

Fonctionnalités 1/2

- Framework applicatif avec réutilisation et remplacement possible des composants
- DVM : Dalvik Virtual Machine (machine virtuelle optimisée pour les périphériques mobiles)
- Navigateur intégré basé sur le moteur WebKit (OpenSource)
- Librairie 2D dédiée
- Gestion de la 3D basée sur une implémentation d'OpenGL ES 1.0 (avec support de l'accélération matérielle)
- Base de données SQLite
- Gestion des écrans tactiles et du Multitouch

Fonctionnalités 2/2

- Multimédia : support de la plupart des formats classiques d'images, de vidéos et audios (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- Téléphonie GSM (selon hardware)
- Bluetooth, EDGE, 3G et WiFi (selon hardware)
- Caméra, GPS, compas et accéléromètre (selon hardware)
- Environnement de développement riche incluant :
 - Un émulateur (avec une interface de contrôle)
 - Des outils de débogage
 - Outils de profiling mémoire et performance
 - Un plugin pour l'IDE Eclipse

Historique

- Développé par la startup Android Inc.
- Juillet 2005 : Rachat par Google
- Novembre 2007 : Open Handset Alliance
 - Texas Instruments, Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile
 - Décembre 2008 : ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, Softbank, Sony Ericsson, Toshiba Corp, Vodafone

Terminaux Visés

- Téléphones portables (HTC, Samsung, Motorola...)
- Netbook/Smartbook (HP Airlife 100, Acer Aspire D250...)
- Tablette Multimedia (Archos, Samsung Galaxy Tab, ...)
- Automobile (Continental AutoLinq : Tesla, Ford...)
- Mais aussi : GPS, Réfrigérateur, Machine à laver...

Et ça ressemble à quoi ?



Mobiles Disponibles

En 2008 : HTC Dream / G1



Mobiles Disponibles

En 2009 : Une quinzaine (HTC, LG, Samsung, Motorola...)



Mobiles Disponibles

En 2010 : De très nombreux mobiles



Concurrents

- Apple iPhone OS : un des leaders en téléphonie, fermé...
- Windows Phone 7 : En progression avec la chute de windows mobile 6, fermé...
- Palm : précurseur, en perte de vitesse, PalmPré ?
- Blackberry : plutôt dédié entreprise mais se démocratise
- Symbian : passage en open source octobre 2009

Mais la plupart de ses concurrents n'ont pas la flexibilité d'Android qui ne se destine pas qu'aux téléphones mobiles !

Parts de Marché

**Worldwide Smartphone Sales to End Users by Operating System in 3Q10
(Thousands of Units)**

Company	3Q10 Units	3Q10 Market Share (%)	3Q09 Units	3Q09 Market Share (%)
Symbian	29,480.1	36.6	18,314.8	44.6
Android	20,500.0	25.5	1,424.5	3.5
iOS	13,484.4	16.7	7,040.4	17.1
Research In Motion	11,908.3	14.8	8,522.7	20.7
Microsoft Windows Mobile	2,247.9	2.8	3,259.9	7.9
Linux	1,697.1	2.1	1,918.5	4.7
Other OS	1,214.8	1.5	612.5	1.5
Total	80,532.6	100.0	100,041,093.3	100.0

Source: Gartner (November 2010)

Les versions...

- Versions d'Android :
 - 1.5 : Cupcake (avril 2009)
 - 1.6 : Donut (septembre 2009)
 - 2.0/2.1 : Eclair (Octobre 2009)
 - 2.2 : FroYo (Mai 2010)
 - 2.3 : Gingerbread (Novembre 2010)
 - 3.0 : Honeycomb (Février 2011)
 - 4.0 : Ice Cream Sandwich (ICS) (Décembre 2011)
- Remarques :
 - Évolution très rapide !
 - Problématique de déploiement

Cupcake 1.5

- 30 Avril 2009
 - Linux Kernel 2.6.27
 - Possibilité d'enregistrer et de regarder des vidéos
 - Upload de vidéos vers Youtube et d'images vers picasa directement depuis le téléphone
 - Un nouveau clavier avec saisie prédictive
- Support du Bluetooth A2DP et AVRCP
 - Possibilité de se connecter automatiquement à un kit bluetooth
 - Ajouts de widget
 - Ajout des dossiers sur le Home
 - Transition d'écrans animées



Donut 1.6

- 15 Septembre 2009
- Linux Kernel 2.6.29
- Nouvelle version du market
- Refonte de la camera et de la galerie (suppression multiple...)
- Mise à jour de la recherche vocale. Réponses plus rapides et meilleure intégration (appel de contacts..)
- Amélioration des recherches pour intégrer les bookmarks, l'historique, les contacts et le web depuis l'écran de démarrage
- Support de nouveaux protocoles de communication
- Support des écrans en WVGA
- Amélioration des performances
- Framework "Gesture" disponible
- Navigation Google Gratuite



Eclair 2.1

- 26 Octobre 2009
 - Linux Kernel 2.6.29
 - Optimisation des performance
 - Support des écrans de taille et de résolution différentes
 - Interface revue
 - Nouvelle interface pour le navigateur et support d' HTML5
 - Nouvelle liste des contacts
 - Intégration de Google Maps 3.1.2
- Support de Microsoft Exchange
 - Support du flash intégré pour l'appareil photo
 - Zoom digital
 - Amélioration du Multitouch
 - Amélioration du clavier virtuel
 - Bluetooth 2.1
 - Live Wallpapers



Froyo 2.2

- 20 Mai 2010
- Linux Kernel 2.6.32
- Amélioration générale de l'OS (vitesse, memoire...)
- Mise en place de JIT
- Integration du moteur JavaScript V8 de chrome dans le navigateur
- Amelioration du support de Microsoft Exchange
- Mise à jour du "Launcher"
- Support du Hotspot Wi-Fi
- Mise à jour du Market et mise à jour automatiques
- Passage rapide d'un clavier d'une langue à une autre



Gingerbread 2.3

- Novembre 2010
- Linux Kernel 2.6.xxx
- Support des technologies NFC (Near Field Communication)
- Client SIP amélioré



HoneyComb 3.0

- Février 2011
- Orienté Tablette
- Prise en charge du multi-coeurs



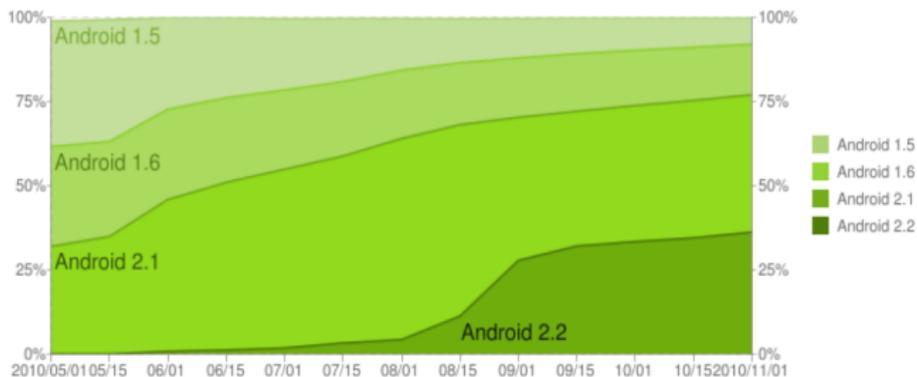
Ice Cream Sandwich

- Déblocage par reconnaissance de visage.
- Amélioration de la navigation internet avec le navigateur Chrome.
- Non support du flash.
- Gestion des form factory sans touches tactiles.



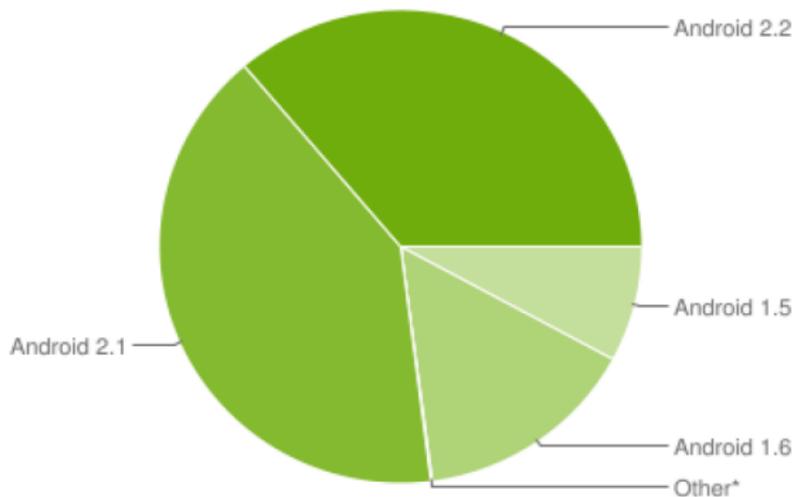
Évolution des Versions

- Au 1er Novembre 2010



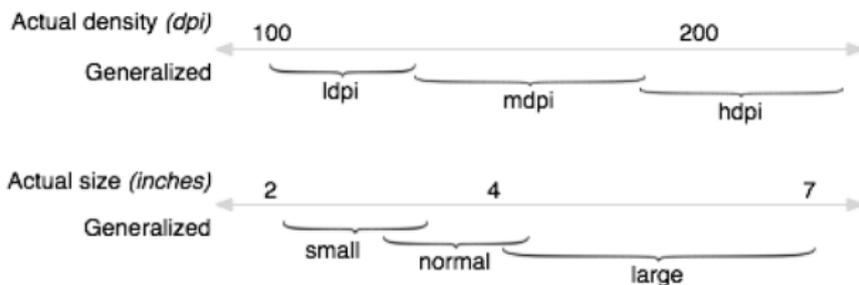
Répartition des Versions

- Au 1er Novembre 2010



Evolution des écrans

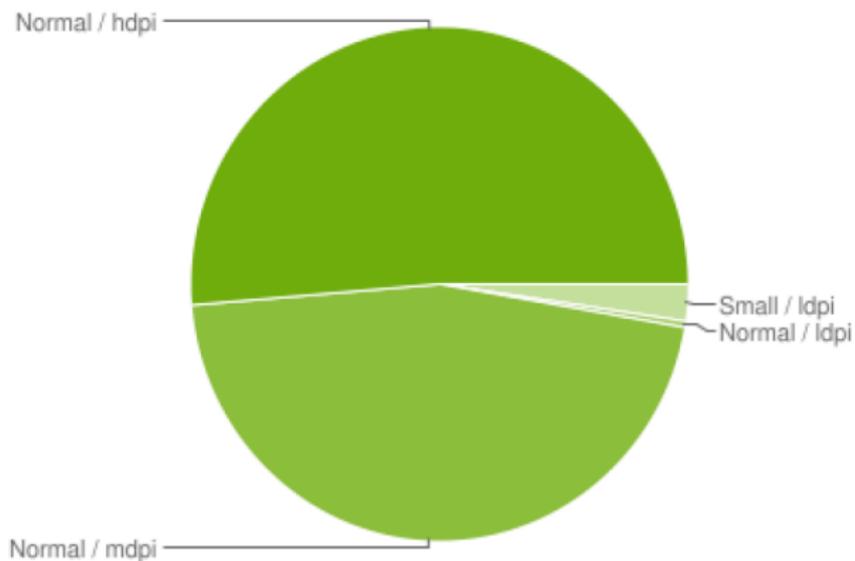
- Au 1er Novembre 2010



	Low Density	Medium Density	High Density
Small	2.3%		
Normal	0.4%	45.9%	51.2%
Large			

Répartition des écrans

- Au 1er Novembre 2010



Android Market

- Système standard de téléchargement d'applications
- Pas de vérifications des applications
- Navigation laborieuse :
 - Par catégorie
 - Recherche par mots clés
 - Par gratuit / payant
- Classement enfant, adolescent, adulte...
- Nécessite un terminal certifié (camera, 3G, compas...)
- Gestion des autorisations avant l'installation
- Possibilité de rendre payant les app.
- Des centaines de milliers d'app.

Android Market

- 25\$ pour s'inscrire en tant que développeur
- 70% du prix revient au développeur, 30% à Google
- Revenus via Google CheckOut
- Achat & vente possible selon les pays
- 57% d'applications gratuites
- App Store : 28%
- Idem Blackberry App World, Nokia Ovi Store...

Markets Alternatifs

- AppsLib (Archos) : <http://appslib.com>
- AndroLib : <http://www.androlib.com>
- Market Samsung
- ...
- Tout a fait autorisé par Google
- Libre de fonctionnement
- Accessible aux terminaux non certifiés

Sans Market

- Via les outils du SDK
- Via des applications disponibles sur le market et la carte SD

Environnement de développement

- Outils :
 - Eclipse
 - SDK Android
 - ADT : Android Development Tools (plugin eclipse)
 - AVD : Android Virtual Device
 - ADB : Android Debug Bridge

Android n'est pas Linux

- Android : un système basé sur Linux ...
- mais avec tellement de modifications - - > pas considérer comme un système Linux
- Android n'est pas un OS GNU/Linux
- Rumeur : Linux 3.3 et Android : début de fusion du noyau

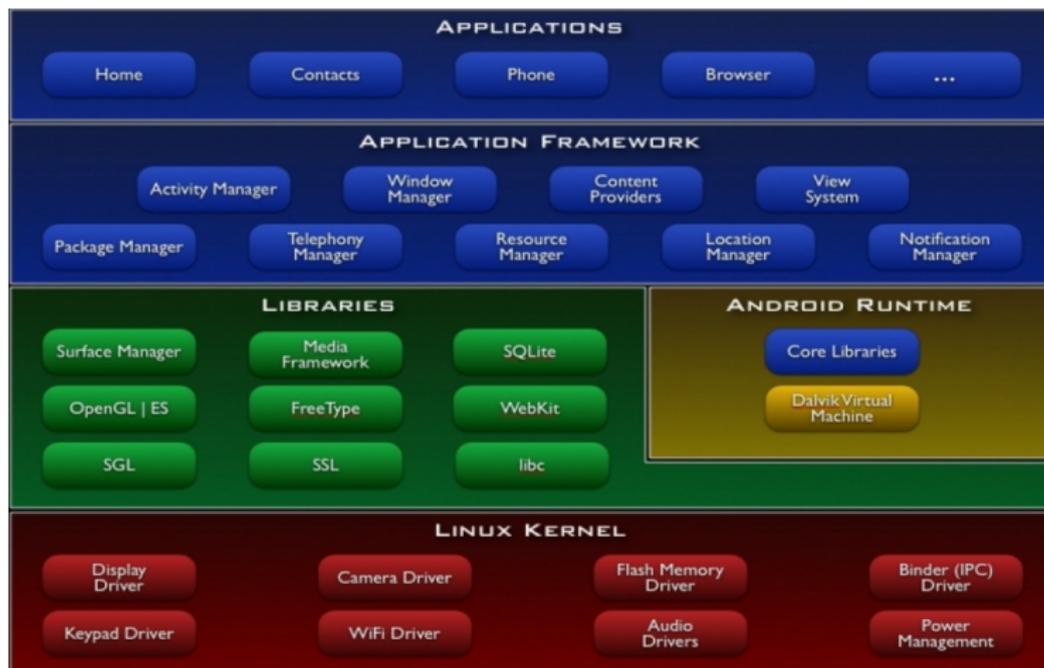
Linux sur périphériques mobiles ?

- GNU/Linux ne convient pas aux appareils mobiles
- Google a donc modifié le noyau Linux
- Android est open source.

Android taillé pour l'embarqué

- Périphériques avec peu de ressources
- Périphériques avec des ressources différentes
- Périphériques avec une utilisation bornée
 - Smart Phone, lecteur de salon, auto-radio....
- AOSP (Android Open Source Project)
 - Licence Apache

Architecture Générale



Architecture Générale

- 5 couches : noyau, bibliothèques natives, runtime, framework, application

Noyau

- Noyau Linux 2.6 (mais modifié)
- Choisi pour sa stabilité, sa maturité et l'ouverture du code
- Principal Changement : Suppression des IPC SysV remplacer par Binder
 - Binder proche de CORBA.
 - Économique en ressource dédié aux architectures qui reposent pas activement sur la gestion de processus.
- Gestion de la mémoire différente. SHM POSIX mais simplifié.
- Partage de mémoire entre processus via Binder
- Système embarqué oblige l'accès aux journaux ne peut pas se faire via `/var/log/*`
- Intégration d'un logger

Noyau

- En standard par de fonction pour terminer l'application
- Viking Killer (Out Of Memory Management)
- Pour tâche de tuer processus quand la mémoire vient à manquer

Noyau Linux

**Display
Driver**

**Camera
Driver**

**Flash Memory
Driver**

**Keypad
Driver**

**WiFi
Driver**

**Audio
Driver**

**Binder (IPC)
Driver**

**Power
Management**

Noyau Linux

- Android repose sur un noyau Linux version 2.6
- Gestion de la sécurité
- Gestion de la mémoire
- Gestion des processus
- Gestion réseau
- Drivers
- ...
- Ce noyaux agit comme une couche d'abstraction entre le matériel et le restes des couches applicatives.

Compatibilité Linux

- Noyau dérivé de Linux mais a été modifié par Google :
- Pas de système X-Window nativement
- Ne supporte pas toutes les libraires GNU standards
- Difficulté de porter toutes les applications (ou librairies) compatibles linux.
- Mais le support de X-Window reste néanmoins possible
- Le Code de google n'est pas reversé dans le noyau linux car Android forme un nouvel arbre de développement.

Bibliothèques Natives

- Elles fournissent un accès direct aux ressources du système
- Une couche d'abstraction au framework Java Android

Librairies

**Surface
Manager**

**Media
Framework**

SQLite

OpenGL | ES

FreeType

WebKit

SGL

SSL

Libc

Librairies

- Android inclus un ensemble de librairies C/C++
- Utilisées par les applications Android
- Accessibles au développeur via le SDK
- Quelques unes de ces librairies
 - Librairie Système C : une implémentation dérivé de l'implémentation BSC des librairies standard C (libc)
 - LibWebCore : Un moteur de navigateur internet moderne utilisé autant pour navigateur android que pour les vues web intégrables
 - SQLite : un système de gestion de base de données relationnel léger et puissant disponible pour toutes les applications.

Bibliothèques Natives : Bionic Libc

- Elles ne reposent pas sur la classique GNU Libc.
- Sa propre bibliothèque C appelée Bionic Libc.
- Pas l'ensemble des fonctions POSIX.
- Bionic Libc ne prend en charge que les architectures ARM et x86.
- Bon support ARM au revoir Power PC ou MIPS
- Les threads sont incompatibles avec POSIX

Bibliothèques Natives (les connues)

- SQLite
- WebKit
- FreeType

Bibliothèques Natives : Google made

- Le media framework : codec, compression, lecture, écriture.
- Surface Manager : Dessiner à l'écran s'interface avec le noyau par *framebuffer*.

Librairies

- ...
 - Librairies MultiMedia : basées sur "PacketVideo's OpenCORE". Intègre le support de la lecture et de l'enregistrement de nombreux formats audio, vidéo et image (MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG...)
 - Surface Manager : gère l'accès et l'affichage des différentes vues (2D ou 3D) composant les applications
 - SGL : Le moteur de rendu pour l'imagerie 2D
 - Librairie 3D : Une implémentation basée sur l'API OpenGL ES 1.0. Intégrant à la fois l'accélération matérielle (si disponible) et l'accélération logicielle.
 - FreeType : Librairie de rendu de police bitmap et vectorielles.

Android Runtime

Libraries Core

**Dalvik Virtual
Machine**

Android Runtime

- DVM : Dalvik Virtual Machine
 - Ecrite par Dan Bornstein
 - Dalvik : village de pêcheurs en islande
 - Une sorte de JVM optimisée pour les systèmes limités en mémoire et en puissance.
 - Exécute les applications ".dex" compilés depuis le code automatiquement par le SDK avec l'outil "dx"
 - Utilise du ByteCode spécifique et non du ByteCode Java
 - Optimisée également pour être "multi-instance" sur un seul terminal.
- Aout 2010 : Oracle (Java) porte plainte envers Google pour leur implémentation de Dalvik qui serait basé sur le code source de java... Procès repoussé en 2012.

Android Runtime : Compilation

- Deux passages :
 - .JAVA vers .CLASS
 - Concaténation des .CLASS en .DEX
- Une application c'est :
 - Le bytecode DEX
 - des ressources (images, sons...)
- Le tout regroupé dans un package .APK

Android Runtime

- Android inclus un ensemble de bibliothèques de base proposant ainsi la quasi totalité des fonctionnalités disponibles dans le langage de programmation Java.
- Chaque application sous Android utilise sa propre instance d'une DVM.
 - Pas de problème d'interaction entre les applications
 - Espace protégé
 - Pas de risque de plantage général
 - D'où la nécessité d'une VM optimisée !

Android Runtime

- Au démarrage d'Android:
 - Une machine virtuelle est lancée afin de pré-charger presque 2000 classes.
 - Zygote
- Les instances de Dalvik initiées par le lancement d'applications sont des *forks* de Zygote.
- Un cache est mis en place dès le démarrage pour accélérer le chargement du *bytecode* DEX.

Android Runtime

- Une machine virtuelle JAVA reposant sur un système GNU/Linux ne serait pas utilisable.
- Un mécanisme de compilation à la volée (JIT) permet d'accélérer l'exécution.
- Les Core Libraries intègrent l'API standard JAVA J2SE 1.5.
- Des fonctionnalités sont enlevées : toolkit SWING, fonctions d'impression.

Android Runtime

- Code Natif :
- Codage via le Android NDK
- JNI permet le pont entre le natif et Dalvik
- Peu utilisé sauf pour les jeux (habitude de programmeurs)
- Permet des gains de performance parfois. Cela dépend de l'application.

Framework Applicatif

- Framework écrit en Java.
- Fournit tout ce que les applications ont besoin.
- API du framework décrite dans la documentation du SDK
- Éléments du framework :
 - Activity Manager : cycle de vie des applications (backstack).
Assure le multi tâche
 - Package Manager : Manipulation du format .apk
 - Window Manager : utilise Surface Manager.
 - Ressource Manager : Tout ce qui n'est pas du code.
 - Content Manager : Partage des données entre processus
 - View System : équivalent d'un toolkit GTK+. Gère le rendu HTML
 - Telephony Service : fournit l'accès aux services GSM, 3G, GPRS
 - Location Service : fournit l'accès à la gestion du GPS.
 - Bluetooth Service
 - Wifi Service
 - Sensor Service

Framework Applicatif

- Framework écrit en Java.

Framework Applicatif

**Activity
Manager**

**Window
Manager**

**Content
Providers**

**Package
Manager**

**Telephony
Manager**

**Resource
Manager**

**Location
Manager**

**Notification
Manager**

**View
System**

Framework Applicatif

- Plateforme de développement Ouverte
 - Permet des application riches et variées
 - Acces au matériel
 - Acces aux informations de localisation
 - Lancement de services de fond
 - Mise en place d'alarmes, de notifications
 - ...

Framework Applicatif

- Plateforme de développement Ouverte
- Architecture conçue pour simplifier la réutilisation des composants
- Publication des capacités des applications
- Les autres applications peuvent utiliser ces capacités
- Chargé facilement les apps.

Framework Applicatif

Une application est composée d'un ensemble de services et de systèmes incluant :

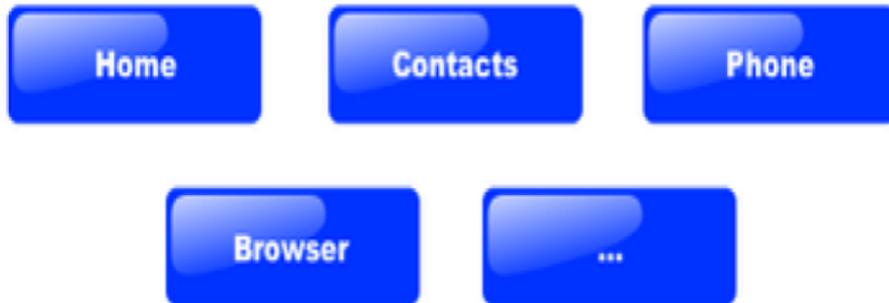
- Un ensemble de vues "Views" utilisées pour construire l'application (listes, grilles, zone de saisies, boutons ou encore navigateur web intégrable)
- "Content Provider" permettant aux applications d'accéder aux données d'autres applications (Contacts...) ou de partager leur propres données.
- "Resource Manager" permettant d'accéder a des ressources tel que des chaines de caractères, des images ou des "layout" (le tout paramétrable selon de multiples critères : taille de l'écran, internationalisation...)

Framework Applicatif

Mais aussi :

- "Notification Manager" permettant à chaque application d'utiliser la barre de statut générale pour y intégrer ses propres informations.
- "Activity Manager" : composant qui gère le cycle de vie d'une application et fournit les outils de navigation applicative.

Applications



Applications

- 2 parties :
 - Les activités : des fenêtres interactives
 - Les services : tâches de fond.
- Les applications tournent dans leurs SandBoxes
- Communications entre applications : Les "intent"
- Intent = intention : formule une demande
- Plusieurs composants peuvent répondre à un "intent" .

Applications

Dernière couche sur Android

Plusieurs sont intégrées dans le système :

- Ecran "Home"
- Gestion des Emails
- Gestion des SMS/MMS
- Gestion de la téléphonie
- Google Maps...
- Applications supplémentaires installables
- Toutes les applications sont écrites via le même SDK !

Généralités

- Les applications sont écrites en Java
- Le code compilé "dex" ainsi que les ressources (images, layout...) sont regroupés dans une archive au format "apk" par les outils du SDK
- Cette archive "apk" est un tout permettant la distribution et l'installation de l'application sur n'importe quelle plateforme android.

Indépendance

- Chaque application Android est isolé des autres à plusieurs niveaux :
 - Chaque application tourne sur son propre process Linux. Ce processus est lancé par Android dès qu'une partie du code nécessite une exécution et inversement tue les processus dont il n'a plus d'utilité.
 - De plus chaque process utilise sa propre machine virtuelle Dalvik. Ainsi chaque application possède son propre environnement.
 - Chaque application est associé à un unique Linux User Id. Ainsi les fichiers d'une application ne sont pas visibles par les autres applications. (mais il existe des moyens de partager ces ressources, par exemple via les Content Provider)
 - Il est possible de forcer deux application de partager le même user ID (et donc de partager des fichiers nativement). Il est également possible donc d'utiliser la même VM et le même processus Linux.

Indépendance

- Un des aspect les plus important d'Android est la réutilisabilité
 - Chaque application peut utiliser des " morceaux d'autres applications" (si elle le permettent)
 - Par exemple si votre application permet de retoucher des photos et que vous désirez publier cette photo vous pouvez utiliser toutes les applications déjà présentes pour réaliser cette tâche (facebook, picasa, mail ...). Et sans utiliser le code de cette application tierce juste en appeler la partie intéressante.
- Ainsi le système doit être capable :
 - De lancer n'importe quelle partie exposée d'une application sans en lancer la totalité
 - Donc les application Andoid n'ont pas de point d'entrée global (méthode main()). Mais sont composés d'éléments indépendants ou chacun peut être lancé individuellement.

Éléments Fondamentaux

- Activity
- Service
- BroadcastReceiver
- ContentProvider
- Intent

Les Activity

- Une activité ("Activity") = une IHM pour une action utilisateur précise :
 - Liste d'éléments parmi lesquels l'utilisateur peut choisir
 - Affichage d'une image avec un titre
 - Affichage d'un calendrier pour choisir une date
- Exemple d'une application de SMS :
 - Une activité pour choisir un contact
 - Une autre pour écrire le message
 - Une autre pour afficher un historique d'échanges.
- Chaque activité est indépendante des autres
- Une activité doit hériter de la classe : `android.app.Activity`

Les Activity

- Une application est donc un ensemble d'activités
- On doit définir quelle est la première activité à exécuter lors du lancement de l'application
- Pour naviguer dans l'application chaque activité doit elle-même lancer l'activité suivante.

Les Activity

- Chaque activité est assignée à une fenêtre
 - Plein écran
 - Fenêtre flottante
- Une activité peut aussi posséder des sous fenêtres
 - Pop-up ...

Les Activity

Le rendu d'une activité est défini par :

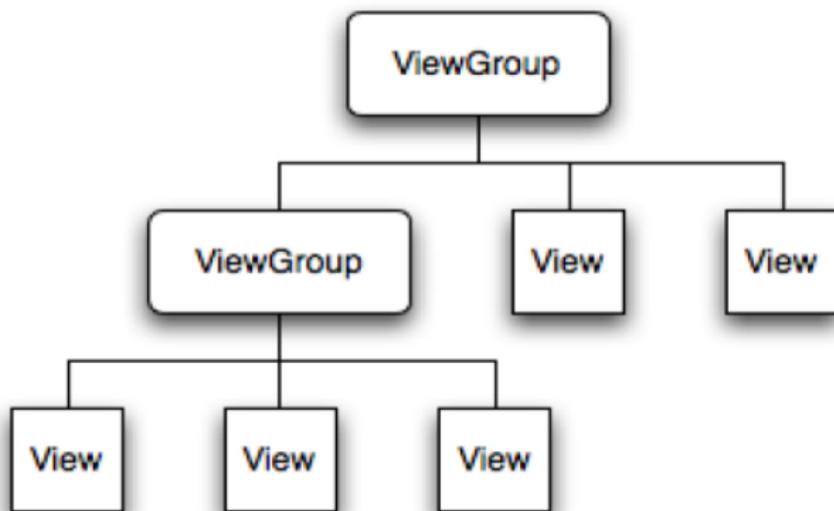
- Une ou un ensemble de vues
- Les vues héritent de la classe `android.view.View`
- Chaque vue contrôle une zone rectangulaire de l'activité
- L'organisation est défini par un arbre de "Layout" ou chaque feuille est une vue.
- Un grand nombre de vues standards sont proposées (combobox, zone de texte, bouton...)
- Possibilité de définir des vues personnalisées

Les Activity

- Les layouts :
 - Agents de placement
 - Plusieurs layouts sont proposés en standard
- Possibilité de définir ses propres Layout
- Les layout sont utilisable via des fichier XML ou via le code Java

Les Activity

- Pour résumer :
- Une fois l'arbre de vues défini on utilise la méthode suivante :
 - `Activity.setContentView()`



Service

- Un service ne possède pas d'interface
- Tourne en arrière plan en continue (ou presque)
- Exemple :
 - Lecture de musique
 - Collecte de données affichables dans une activité
 - Suivi GPS
 - Vérification de mise à jour
 - ...

Service

- Lancement d'une application musicale
 - Démarrage de l'activité de "choix de chanson"
 - L'utilisateur lance la musique
 - Le service diffuse cette musique
 - L'utilisateur peut quitter l' "application" en fermant l'activité
 - La musique continue à être diffusée !

Service

- Pour communiquer avec un service il faut :
 - S'y connecter (il se lance si il était arrêté)
 - Utiliser l'interface que présente ce service
 - - Exemple : Play(), Pause(), next() ...
- Un service s'exécute dans un Thread et donc ne bloque pas le reste du terminal quand il tourne en fond.

Broadcast Receiver

- Les broadcast receiver sont :
 - Des éléments inactifs qui attendent un évènement
 - Il y a des évènements système :
 - - Batterie faible
 - - Changement de langue du système
 - - L'utilisateur a pris une photo
 - - ...
- Il est possible de définir ses propres évènements
- Héritent de la classe `android.content.BroadcastReceiver`

Broadcast Receiver

- Une application peut contenir plusieurs receivers : un par événement important
- Les receivers n'ont évidemment pas d'interface
- Ils peuvent lancer des activités en cas de besoin
- Ils peuvent également utiliser le NotificationManager pour signaler quelque chose à l'utilisateur (préférable)
 - Icône, vibration, alerte sonore, clignotement diode...

Content Provider

- Les content provider permettent de partager du contenu entre les applications
- Une application s'en sert pour rendre public certaines de ses données
- Les données sont donc exposées dans une classe héritant de `android.content.ContentProvider`
 - Methode `query()`
 - `Insert()`
 - `Update()`
 - `delete()`...

Content Provider

- Les autres applications n'accèdent pas directement à la classe de ContentProvider
- Utilisation d'un ContentResolver qui va rediriger les requêtes vers le provider voulu
- Si l'on tente d'accéder à une ressource d'une application n'étant pas en cours d'exécution le système Android se charge de la lancer avant.

Intent

- Les content providers sont activés par une requête d'un content resolver
- Mais les 3 autres systemes (Activity, Service, BroadCast Receiver) sont activés par des messages asynchrone appelés "Intent"
- Un intent dérive de android.content.Intent
- Un intent possède une action et un contenu particulier

Intent

- Pour les activités et les services il nomme l'action désirée et précise l'URI des données sur lesquelles agir.
 - Afficher / image
 - Editer / texte
 - ...
- Pour les broadcast receivers il se contente de nommer l'action à annoncer
 - Batterie faible
 - ...

Intent

- Les Intents et les activités :
 - Lancement en passant un Intent en paramètre à une des méthodes suivantes :
 - * `Context.startActivity()`
 - * `Activity.startActivityForResult()`
 - L'activity peut accéder à celui ci avec :
 - * `getIntent()`
 - Si le système doit envoyer des nouveaux intent :
 - * Appel de `onNewIntent()` sur l'activité
 - En cas de résultat attendu
 - * Appel de `onActivityResult()` sur l'activité appelante

Intent

- Les Intents et les services :
 - Lancement en passant un Intent en paramètre à la méthode suivante :
 - * `Context.startService()`
 - * Le système appliquera ensuite la méthode `onStart()` en précisant cet Intent en paramètre
 - Connexion en passant un Intent en paramètre à la méthode suivante :
 - * `Context.bindService()`
 - * Le système appellera ensuite la méthode `onBind()` en précisant cet Intent en paramètre

Intent

- Les Intents et les Broadcast receiver :
 - Une application voulant envoyer un évènement va utiliser une des méthodes suivantes :
 - * `Context.sendBroadcast()`
 - * `Context.sendOrderedBroadcast()`
 - * `Context.sendStickyBroadcast()`
 - Le système va alors appeler la méthode `onReceive()` sur tous les broadcast receivers intéressés en passant en paramètre l'Intent.

Intent

- La catégorie
 - Une chaîne de caractère précisant quel type de composant peut gérer l'intent.
 - Plusieurs catégories peuvent être précisées.
 - Exemples :
 - * CATEGORY_BROWSABLE : Le contenu peut être affiché dans le navigateur
 - * CATEGORY_HOME : L'activité est de type Home
 - * CATEGORY_LAUNCHER : L'activité est lancable par le launcher et donc doit y être présente
 - * CATEGORY_PREFERENCE : l'activité est un panneau de préférences

Intent

- Quelques exemples :
 - ACTION_VIEW content://contacts/people/1 – Affiche les informations sur le contact 1
 - ACTION_DIAL content://contacts/people/1 – Affiche le mode d'appel rempli avec les informations du contact 1
 - ACTION_VIEW tel:123 – Affiche le mode d'appel rempli avec "123". (ACTION_VIEW s'adapte donc au contenu)
 - ACTION_DIAL tel:123 – Idem
 - ACTION_EDIT content://contacts/people/1 – Permet de modifier les informations du contact 1
 - ACTION_VIEW content://contacts/people/ – Affiche la liste des contacts (le choix d'un de ces contacts générera un Intent pour afficher ce contact)

Manifest

- Fichier XML
- Précise l'architecture de l'application
- Chaque application doit en avoir un
- AndroidManifest.xml à la racine du projet

Manifest

- Contenu :
 - Précise le nom du package java utilisant l'application. Cela sert d'identifiant unique !
 - Il décrit les composants de l'application
 - - Liste des activités, services, broadcast receivers
 - - Précise les classes qui les implémentent
 - - Précise leurs capacités (à quels intents ils réagissent)
 - - Ceci permet au système de savoir comment lancer chaque partie de l'application afin de satisfaire au principe de réutilisabilité.

Manifest

- Contenu suite :
 - Définit les permissions de l'application
 - - Droit de passer des appels
 - - Droit d'accéder à Internet
 - - Droit d'accéder au GPS
 - - ...
 - Précise la version d'Android minimum nécessaire
 - Déclare les bibliothèques utilisées
 - Déclare des outils d'Instrumentation (uniquement pour le développement)

Manifest

- Conventions :
 - Seuls deux éléments sont obligatoire
 - - < *manifest* > : contient le package, la version... Englobe tout le fichier
 - - < *application* > : décrit l'application et contiendra la liste de ses composants.
 - Les données sont passées en tant qu'attribut et non en tant que contenu
 - Tous les attributs commencent par " android:" (sauf quelques un dans < *manifest* >)

Manifest

- Les ressources
 - Au lieu de contenir les données en tant que tel le fichier manifest peut faire appel à des ressources
 - `< activityandroid : icon = "@drawable/smallPic" ... >`
 - Ces ressources sont définies dans le répertoire "res" de l'application.

Manifest

- Permissions
 - Une application ne peut pas utiliser certaines fonctionnalités sauf si il le précise dans le fichier Manifest
 - Il faut donc préciser les permissions nécessaires grace à :
`< uses – permission >`
 - Il existe des permission standard :
 - - android.permission.CALL_EMERGENCY_NUMBERS
 - - android.permission.READ_OWNER_DATA
 - - android.permission.SET_WALLPAPER
 - - android.permission.DEVICE_POWER
- Il est possible de définir ses propres permissions

Manifest

- Intent Filter
 - Ils informent le système à quelle intent les composants peuvent réagir
 - Un composant peut avoir plusieurs filtres
 - Editeur de texte
 - - Filtre pour éditer un document existant
 - - Filtre pour initier un nouveau document
 - Un filtre doit posséder une "action" qui définit à quoi il correspond

Manifest

● Exemple

```
<?xml version="1.0" encoding="utf-8" ?>
- <manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.kahriboo.menu3d" android:versionCode="1" android:versionName="1.0">
- <application android:icon="@drawable/icon1" android:label="@string/app_name"
  android:debuggable="true" android:theme="@android:style/Theme.Translucent.NoTitleBar">
- <activity android:name=".MainActivity" 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>
<uses-sdk android:minSdkVersion="6" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.VIBRATE" />
</manifest>
```

Activity & Task

- Résumé :
 - L'application A doit afficher une carte
 - A prépare l'intent avec les données nécessaires
 - A appelle `startActivity()` avec cet intent
 - Le système trouve l'application B qui sait gérer cet Intent
 - L'application B affiche la carte
 - L'utilisateur ferme cette carte (bouton back)
 - L'application A reprends la main

Activity & Task

- Du point de vue de l'utilisateur :
 - 1 seule application (A et B sont confondues)
- Du point de vue du système :
 - 2 applications
 - 2 DVM
 - 2 process
 - 1 Tâche = 1 Application au sens utilisateur.

Activity & Task

- Une tâche :
 - Est une pile d'activités
 - La première est celle qui a été initiée par l'utilisateur
 - Les activités peuvent provenir de différentes applications
 - L'ensemble forme un tout
 - - Mis en arrière plan en même temps
 - - Remise au premier plan dans son ensemble
- Comportement par défaut modifiable via le manifest et le tag "`< activity >`" et ses flags

Processus & Threads

- Quand le premier composant d'une application nécessite une exécution Android démarre un nouveau processus Linux pour gérer ce composant
- Chaque composant peut préciser dans la Manifest (via l'attribut "process") si il doit s'exécuter dans un nouveau processus ou si il doit partager un processus existant
- Deux composant de deux applications peuvent aussi partager le même processus si :
 - Elle utilisent le même Linux User ID
 - Elles sont signées par la même autorité

Processus & Threads

- Attentions pour les composant utilisé dans le même processus
 - Ne pas faire de longues opérations lors des appels par le Systeme (`View.onKeyDown()`) sinon cela bloquera tout le reste des composants.
 - Penser à utiliser des Threads pour les traitements longs.
 - Utiliser la classe classique Java de Threads
 - Android fournit aussi des classes utilitaires pour simplifier l'utilisation des Threads

Cycle de Vie

- Une activité possède trois états :
 - Active (running) : Quand l'activité est au premier plan et recoit les actions utilisateur.
 - Paused : Quand elle est toujours visible mais n'a pas le focus (autre activité transparente par dessus ou activité ne prenant pas tout l'écran)
 - - Toujours vivante
 - - Mais peut être tuée en cas de ressources très limitées
 - Stopped : Quand elle n'est plus visible
 - - Toujours vivante
 - - Mais sera tuée dès que des ressources seront nécessaires.

Cycle de Vie

- Le système tue les activités en état "stopped" (ou "paused") de deux manières :
 - En appelant la méthode finish()
 - En tuant le processus tout simplement
- Quand l'activité sera a nouveau demandée :
 - Doit être complètement reconstruite
 - Doit Potentiellement recharger son dernier état

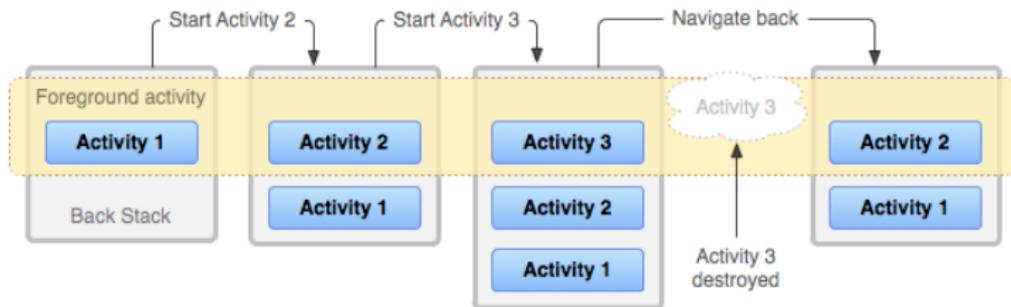
Cycle de Vie

- Une activité est notifiée de ses changement d'état par l'appel à ses méthodes :
 - void onCreate(Bundle savedInstanceState)
 - void onStart()
 - void onRestart()
 - void onResume()
 - void onPause()
 - void onStop()
 - void onDestroy()

Cycle de Vie

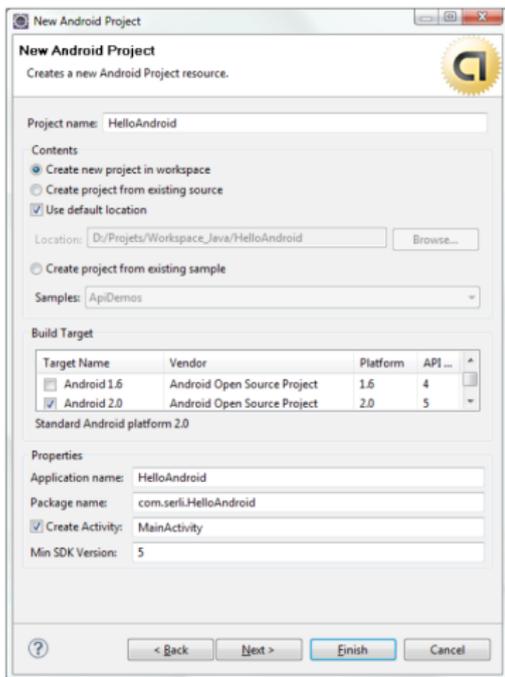
- Afin de sauvegarder le contexte le système appelle "onSaveInstanceState()" avant de rendre l'application potentiellement tuable (paused...)
- Cet appel fournit un bundle "clé/valeurs" pour que le développeur puisse sauvegarder l'état
- Au prochain appel de "onCreate()" ce bundle sera fournit
- Il est également fournit via un appel à "onRestoreInstanceState()"
- L'appel à la sauvegarde n'est faite qu'en cas de risque de terminaison de l'activité par le système et non si cela vient d'une action utilisateur (back)

Back Stack



Hello World

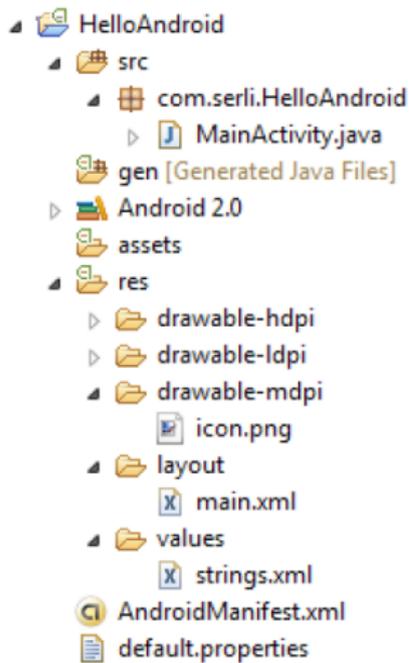
- Créer un nouveau projet :



Hello World

Organisation des dossiers :

- src : sources
- gen : code généré
- res : ressources
- drawable : images
- layout : layout
- values : constantes
- Manifest



Hello World

Manifest :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.serli.HelloAndroid"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MainActivity"
            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>
    <uses-sdk android:minSdkVersion="5" />
</manifest>
```

Hello World

Ressources (Layout, String, Images)

- On y accède par "@dossier/nom" ou "R.dossier.nom"
- Ressources alternatives : selon la langue, le sdk, l'écran...

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello World, MainActivity!</string>
  <string name="app_name">HelloAndroid</string>
</resources>
```

```
<?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"
  >
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
  />
</LinearLayout>
```

Hello World

```
package com.serli.HelloAndroid;

import android.app.Activity;

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

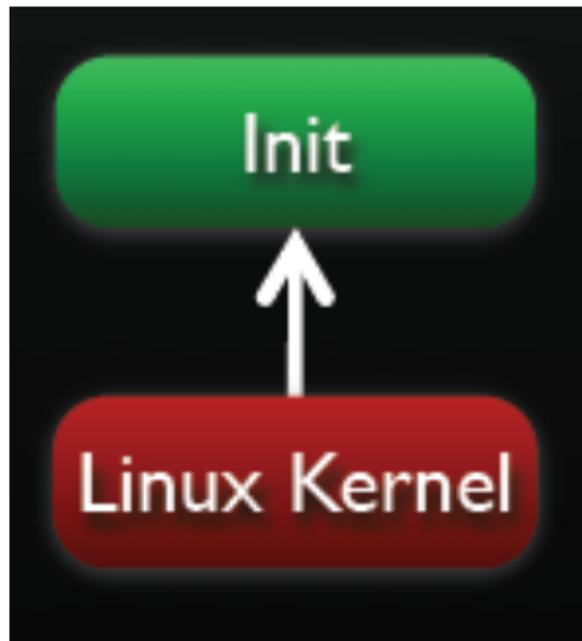
Activity

Hello World

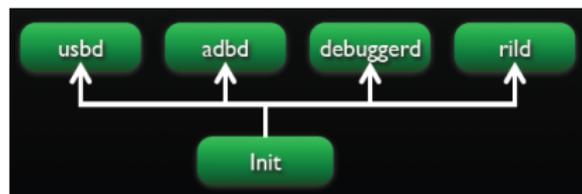
Exécution de l'application :

- Sur un émulateur
- Sur un terminal

Un peu plus sur Zygote



Un peu plus sur Zygote



Un certain nombre de daemons sont lancés :

- USB daemon (usbd)
- Android Debug Bridge (adbd)
- Debugger Daemon (debuggerd); Il permet de gérer les requêtes de debug de processus (dump memory, etc.)
- Radio Interface Layer Daemon (rild)

Un peu plus sur Zygote



- Le processus zygote :
- initialise une instance de Dalvik VM
- Pré charge les classes et écoute sur une socket pour créer des Dalvik VM
- Fork sur demande pour créer des instances de Dalvik VM pour chaque application
- Les VM créées partagent des zones mémoire communes ce qui permet de minimiser la mémoire utilisées
- Chaque VM créer par zygote est un fork d'une VM "mère", ce qui permet d'accélérer le démarrage d'une application.

Un peu plus sur Zygote

