République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri de Tizi-Ouzou
Faculté du Génie Electrique et d'Informatique
Département Informatique

Mémoire

de fin d'études

En vue de l'obtention du diplôme de Master en Informatique

Option : *Systèmes Informatiques*

Thème :

Implémentation d'un interprète de commande pour la plateforme Android

Dirigé par :

Mr M. Daoui

Réalisé par :

Melle BOUTERFA Dehia

Promotion 2011-2012

REMERCIEMENTS

Je remercie tout d'abord « DIEU » le tout puissant de m'avoir donné la santé et le courage d'effectuer ce projet de fin d'étude, dans les meilleures conditions.

La réalisation de cette thèse fut une occasion merveilleuse de rencontrer et d'échanger avec de nombreuses personnes. Je ne saurais pas les citer toutes sans dépasser le nombre de pages raisonnablement admis dans ce genre de travail. Je reconnais que chacune a, à des degrés divers, mais avec une égale bienveillance, apporté une contribution positive à sa finalisation. Mes dettes de reconnaissance sont, à ce point de vue, énormes à leur égard.

Je pense particulièrement à Mr DAOUI, mon promoteur, pour la finesse de ses attitudes sur le plan aussi bien humain que scientifique. Ses remarques successives ont permis d'améliorer les différentes versions de ce travail. Il a toujours trouvé comme promoteur le juste équilibre entre la liberté qu'il m'a laissée dans le choix des grandes orientations et dans la détermination des pistes à suivre, d'une part, et un soutien total et sans faille dans les moments délicats, d'autre part. De lui, j'ai toujours reçu non seulement les encouragements dont j'avais tant besoin, mais aussi les précieux conseils pratiques que seul un homme, ayant des qualités humaines comme lui, peut amener à prodiguer. Grâce à son approche respectueuse de la personne humaine, je me suis continuellement sentie à l'aise. Je lui en suis infiniment gré.

J'adresse mes remerciements aux membres du jury, devant qui, j'ai l'honneur d'exposer mon travail et qui ont pris la peine de lire ce mémoire pour juger son contenu.

Mes remerciements les plus chaleureux et ma gratitude la plus sincère vont à mes professeurs pour leurs passions et leurs volontés à nous donner chaque jour le meilleur d'eux même pour enrichir nos connaissances, merci pour tout ce qu'ils nous ont appris depuis notre première année jusqu'à ce jour .

A toute personne qui m'a aidé de prés ou de loin pour réussir ce modeste travail. Je ne ferais pas le pari de les énumérer sans risquer d'en omettre certains. Je m'astreigne à un devoir de reconnaissance à l'égard d'eux tous.

MERCI

In Memorium

Je dédie ce travail à mon regretté grand père « Ali » : en souvenir de ce que tu as fait et ce que tu as été pour moi durant ton existence. Je sais que tu serais bien content d'apprendre que ta « Dahbia » va obtenir un master. Que dieu t'élève au rang de ses illustres amis.

A ma chère grand mère Ouardia : en souvenir des deux ans que j'ai passés auprès de toi, les joies que tu m'as procurées, qui font que chaque jours qui passe me font regrettées le fait que je n'aie aucun souvenir de toi. Que dieu t'accueil dans sa miséricorde

Dédicaces

Au nom d'ALLAH, le Clément et qui le manifeste, louange à toi le maître des mondes. A travers ce mémoire, je tiens à remercier du fond du cœur certaines personnes qui ont eu à contribuer de prés ou de loin à sa réalisation.

A mes très chers parents, ma mère, qui est la lumière de notre maison, mon père, qui est le guide de réussite dans ma vie. Pour leur aide et leur soutien tout au long de mes études, et qui ont fait de moi ce que je suis aujourd'hui et j'espère qu'un jour je serai capable de leur rendre un minimum de ce qu'ils m'ont donné.

A ma chère « Nana »et mes grands parents maternelle « Davou » et « stsi » : votre souci permanent a été mon bonheur et ma réussite. Que Dieu vous donne longue vie pour que vous puissiez assister à d'autres réussites inchalallah.

A mes soeurs : Celya, Ouardia et Thiziri.
A mon petit frère Ali,
A tous mes amis (es) de la promotion 2O11 /2O12.

TABLES DES MATIÈRES

CHAPITRE III : CONCEPTION ET REALISATION

CHAPITRE IV : IMPLEMENTATION ET TESTES

LISTE DES FIGURES

INTRODUCTION GÉNÉRALE

L'année 2007 a été vécue par le monde de la téléphonie mobile comme le fameux big-bang qui a bouleversé l'univers.

Avant cette date, la téléphonie mobile suivait une évolution tranquille. Chaque année apportait son lot de méga pixels supplémentaires aux appareils photo intégrés, et le GPS faisait son apparition sur les modèles haut de gamme, mais les nouveaux téléphones n'étaient pas fondamentalement différents de ceux qui les avaient précédés.

Pour l'utilisateur, le téléphone était toujours un appareil aux menus obscurs comportant des options de configuration étranges qu'il valait mieux ne pas modifier . La saisie était assurée par un clavier numérique peu confortable, qui, même avec les mécanismes évolués de l'époque (complétion T9), ne permettait pas de saisir du texte rapidement.

Certains téléphones haut de gamme proposaient une interface tactile, mais elle imposait généralement l'utilisation d'un stylet, parce que ces systèmes étaient la transposition quasi directe sur un écran très petit des principes d'interface homme-machine existants sur les ordinateurs de bureau. Cliquer sur un bouton minuscule ou déplacer une barre de défilement avec un stylet alors qu'on est en train de marcher, relève de l'exploit. [4]

Et la très grande majorité des utilisateurs se contentaient des applications de base fournies avec le téléphone, souvent parce qu'ils ignoraient la possibilité d'installer de nouvelles applications, et l'offre était relativement limitée et peu intéressante. « *Le téléphone était fait pour téléphoner, envoyer des SMS et prendre des photos, et rarement beaucoup plus.* »[1]

Pour le développeur, la situation était complexe : la plupart des téléphones étaient capables d'accueillir des applications JavaME (Java Micro Edition), mais ce standard limité interdisait toute intégration avec les fonctions natives du téléphone, sans parler de l'énorme variété des implémentations et des interprétations du standard qui rendaient le développement d'une application portable digne excessivement difficile.

En juin 2007, une première révolution, qui changea radicalement le marché du téléphone mobile, eut lieu: Apple sortit l'iPhone, un téléphone d'un genre entièrement nouveau, doté d'une superbe interface tactile réellement utilisable au doigt, avec des applications simples d'usage mais très efficaces, disponibles sur un marché d'applications en ligne.

Dès ce moment, tout nouveau téléphone lui était immédiatement comparé, et chaque constructeur s'est empressé d'annoncer son futur « iPhone killer ».

En parallèle, les équipes de Google travaillaient sur un projet tout aussi ambitieux. En novembre 2007, ils annoncèrent Android avec leurs partenaires de l' Open Handset Alliance : un système d'exploitation pour téléphone mobile moderne, fondé sur Linux, avec une interface tactile similaire à celle de l'iPhone, et intégralement Open Source. Il fallut attendre octobre 2008 pour avoir un téléphone Android entre les mains, et c'était clairement une réussite. Android est le seul « iPhone killer » crédible.

Pour le développeur, Android est une bénédiction : le développement se fait en Java, pas un « sous-Java » comme c'est le cas avec JavaME, mais un environnement complet, reprenant la majore partie de la librairie du JDK et y ajoutant tout ce qui permet de construire très facilement des applications tactiles, graphiques, communicantes, géolocalisées, etc. Et les outils de développement fonctionnent sur un ordinateur quelconque, avec une excellente intégration dans Eclipse.

Android ouvre le développement d'applications mobiles à l'énorme population des développeurs Java, ce qui ne peut que favoriser l'émergence d'un écosystème très riche.

Dans le cadre de ce mémoire, nous présenterons la plateforme Android en décortiquant son architecture qui est comme nous le découvrirons par la suite d'une grande élégance : en permettant la décomposition d'une application en « activités » communicantes, l'ensemble des applications présentes sur notre téléphone peuvent coopérer, et de nouvelles applications peuvent enrichir celles qui sont déjà présentes.

Nous verrons par la suite comment développer et intégrer une application à la plate-forme, nous avons pris comme cas d'étude, et qui est donc le thème de ce mémoire, implémentation d'un interprète de commande à la plate-forme android.

Dans cette optique, nous avons structuré notre mémoire comme suit :

- **Une introduction générale** : dans laquelle on a introduit les grandes lignes de notre travail.
- **Chapitre 1:** présentation de la plate-forme android, son architecture, les différents supports, etc
- **Chapitre 2** : sera consacré a la présentation des interprètes de commandes, en général, et du shell d'Unix, en particulier.
- **Chapitre 3** : consacré à comment développer une application, et donc un interprète de commande, pour la plate-forme android.
- **Chapitre 4:** le dernier chapitre sera consacré à l'implémentation, nous y présenterons l'environnement de développement, les outils qui ont servi à la réalisation de notre application, et nous terminerons par la présentation de ses fonctionnalités à travers ses différentes interfaces.
- **Conclusion générale et perspectives d'avenir** : pour récapituler les points cruciaux de notre travail et donner quelques perspectives,
- **Annexe** : pour présenter quelques éléments en relation avec le développement pour la plate-forme android

1. Introduction :

La téléphonie mobile a connu une explosion dans les années 2000 mais aucune révolution n'a semblé arriver depuis : les appareils tendaient à tous se ressembler, les innovations n'avaient plus vraiment de saveur ; les applications étaient difficiles d'accès de par leur mode de distribution et souvent peu performantes à cause des faibles capacités des appareils.

Depuis 2007, avec l'arrivée de l'Iphone d' Apple, la téléphonie est entrée dans une nouvelle ère. Les Smartphones sont dotés d'une puissance plus importante et d'espaces de stockage conséquents. *Les téléphones tendent à devenir des objets artistiques, presque de reconnaissance sociale [1]*, et possèdent des fonctionnalités qu'aucun téléphone ne pouvait espérer auparavant : connexion haut débit, localisation GPS, boussole, accéléromètre, écran tactile souvent multipoints, marché d'applications en ligne... Autant de qualités permettant de créer des applications innovantes et de les distribuer en toute simplicité.

La plate-forme Android apporte tout cela au consommateur, mais surtout, elle affranchit le développeur de nombreuses contraintes par son ouverture ; elle permet à n'importe quel développeur de créer ses applications avec un ticket d'entrée quasi nul. Le Framework et le système d'exploitation et outils associés ont un code source ouvert, leur accès est gratuit et illimité. Plus besoin de négocier avec le constructeur du téléphone pour qu'il vous laisse développer sur sa plate-forme.

2. Définition :

D'après Google, Android est une plate-forme libre destinée aux appareils mobiles comprenant un système d'exploitation, des applications, et les couches intermédiaires nécessaires pour faire fonctionner le tout.

Pour faciliter la tâche aux développeurs, il a été prévu dès le départ qu'Android serait facilement programmable. C'est le langage Java qui a été retenu et Google fournit aux développeurs de nombreux outils pour écrire et mettre au point leurs programmes : émulateur pour les tests, SDK Java et plugin Eclipse pour le développement, ainsi qu'une documentation fournie, disponible sur internet (www.android.com).

Google fédère autour d' Android tout un écosystème d'un quarantaine de développeurs , d'utilisateurs et de constructeurs de matériels représentant une alliance spécialisées dans les applications mobiles (Open Handset Alliance).

3. L'Open Handset Alliance :

Dès son origine, la démarche de Google a été d'ouvrir le développement d'Android en rassemblant autour de lui et au travers de l'Open Handset Alliance (OHA) un maximum de sociétés. Les membres de ce consortium sont très variés : nous y trouvons des fabricants de téléphones connus tels que Sony Ericsson, Samsung ou Motorola, des opérateurs de téléphonie comme Sprint, T-Mobile ou NTT DoCoMo, des sociétés Internet, Google évidemment mais aussi eBay, des constructeurs de puces électroniques Intel, nVidia, ou encore des acteurs du marché du GPS comme Garmin. Toutes ces entités se retrouvent donc au sein de cette alliance, pour des raisons qui leur sont propres, pour œuvrer au développement d'Android.

4. Architecture [9] :

Le schéma suivant illustre la structure de la plate-forme Android, sous forme de couches qui s'empilent :

Figure 1.1 : L'architecture en couche d'Android.

4.1. Le noyau linux :

Le système d'exploitation d'Android se base sur Linux. Si on veut être plus précis, c'est le noyau (« kernel » en anglais) de Linux qui est utilisé. Le noyau est l'élément du système d'exploitation qui permet de faire le pont entre le matériel et le logiciel.

La version du noyau utilisée avec Android est une version conçue spécialement pour l'environnement mobile, avec une gestion avancée de la batterie et une gestion particulière de la mémoire.

Android est une distribution de Linux, il a le même cœur mais c'est tout. En d'autres termes on ne pourra pas lancer des applications destinées à GNU/Linux sans passer par de petites manipulations.

En regardant attentivement le schéma 1.1, on remarque que cette couche est la seule qui gère le matériel. Android en sois ne s'occupe pas de ce genre de détails. On ne veut pas dire par là qu'il n'y a pas d'interactions entre Android et le matériel, juste que quand un constructeur veut ajouter un matériel qui n'est pas pris en compte par défaut par Android, il doit travailler sur le kernel et non sur les couches au-dessus, qui sont des couches spécifiques à Android.

4.2. DES BIBLIOTHÈQUES C/C++:

Au-dessus du noyau proprement dit se loge un ensemble de librairies C/C++. Parmi celles-ci on peut noter l'incontournable *libc* (librairie système C standard) dont l'implémentation a été adaptée pour le mode embarqué.

Il y a également diverses bibliothèques graphiques assurant le rendu vectoriel des polices de caractères ou encore le rendu 2D et 3D.

Côté « média », Android ne manque de rien : il lie les formats d'images PNG et JPG, il est capable d'encoder et de décoder des flux audio et vidéo aux formats AAC,MP3, ou Ogg Vorbis.

On peut ajouter WebKit et SQLite :

- *WebKit* est le moteur du navigateur web qui peut également être manipulé par du code dans les applications. Ce moteur a été incorporé dans de nombreux navigateurs : Safari sur Mac, Konqueror (KDE Linux), Google Chrome, et surtout dans bon nombre de téléphones portables (Nokia, iPhone, Palm Pré...).

- *SQLite* est un moteur de base de données sous forme de librairie C et par conséquent destiné exclusivement à être encapsulé dans un logiciel. SQLite se trouve ainsi utilisé comme système de stockage interne dans énormément d'applications, c'est même la référence des moteurs SQL embarqués.

4.3. UN MIDDLEWARE JAVA :

Si nous remontons les couches de la plateforme, après le noyau Linux et les librairies C/C++ , nous arrivons au middleware, l'environnement d'exécution Android.

C'est cette couche qui fait qu'Android n'est pas qu'une simple « implémentation de Linux pour portables ». Elle contient certaines bibliothèques de base du Java accompagnées de bibliothèques spécifiques à Android et la machine virtuelle « Dalvik ».

Voici un schéma qui indique les étapes nécessaires à la compilation et à l'exécution d'un programme Java standard :

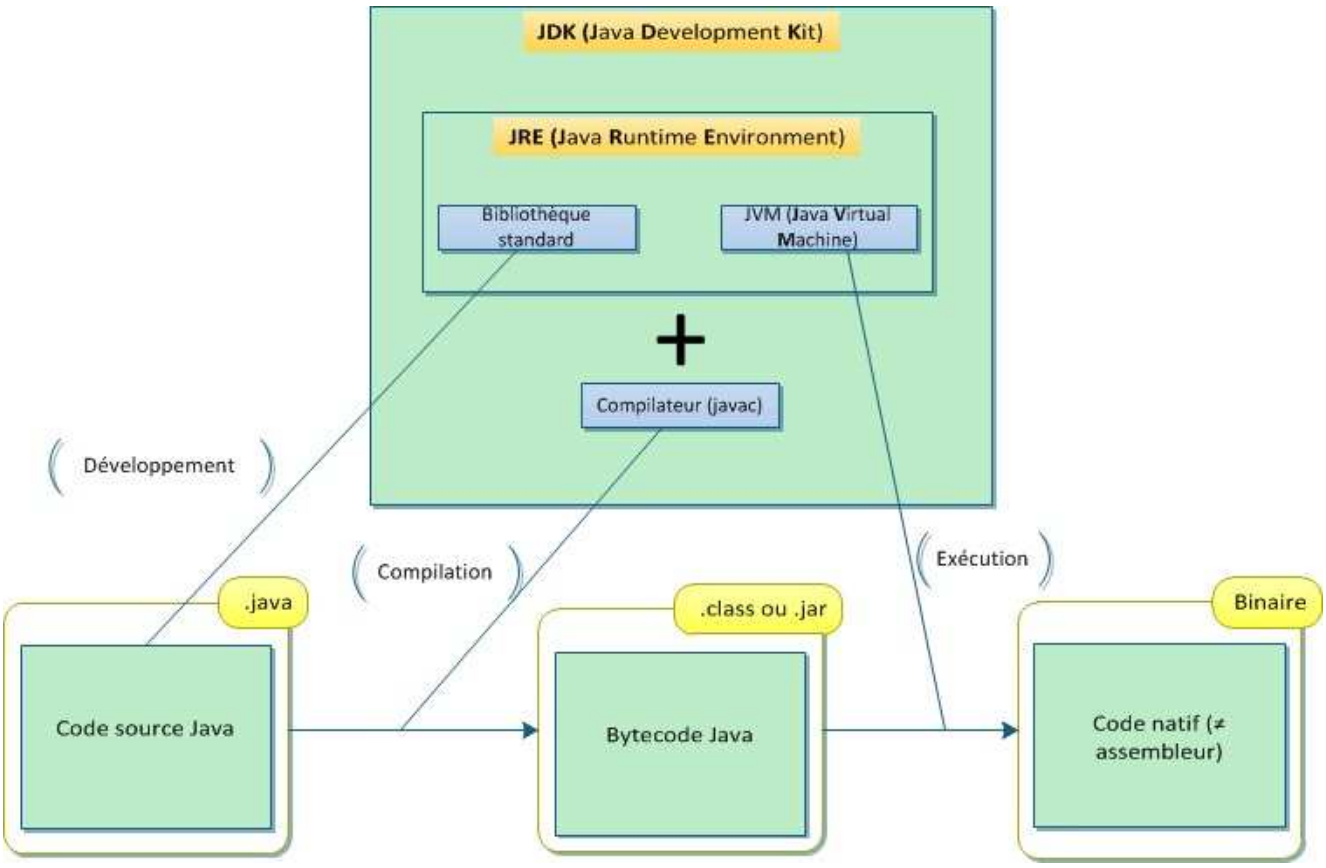


Figure 1.2 : schéma d'exécution d'un programme standard java.

Le code est une suite d'instructions stockées dans un fichier « .java » et ce fichier sera traduit en une autre suite d'instructions dans un autre langage que l'on appelle le «bytecode». Ce code est contenu dans un fichier « .class ». Le bytecode est un langage spécial qu'une machine virtuelle Java peut comprendre et interpréter. Les différents fichiers « .class » sont ensuite regroupés dans un « .jar » et c'est ce fichier qui est exécutable.

En ce qui concerne Android, la procédure est différente. En fait ce qu'on appelle Java est certainement une variante particulière de Java qui s'appelle « Java SE ». Or, pour développer des applications pour Android, on n'utilise pas vraiment Java SE.

La version de Java qui permet le développement Android est une version réduite amputée de certaines fonctionnalités qui n'ont rien à faire dans un environnement mobile. Par exemple, la bibliothèque graphique Swing n'est pas supportée, on trouve à la place un système beaucoup plus adapté. Android n'utilise pas une machine virtuelle Java ; une machine virtuelle toute étudiée pour les systèmes embarqués a été développée, et elle s'appelle « Dalvik ». Cette machine virtuelle est optimisée pour mieux gérer les ressources physiques du système. On citera par exemple qu'elle permet

de laisser moins d'empreinte mémoire et qu'elle puise moins dans la batterie qu'une machine virtuelle Java.

La plus grande caractéristique de Dalvik est qu'elle permet d'instancier un nombre très important d'occurrences de elle-même : chaque programme a sa propre occurrence de Dalvik et elles peuvent vivre sans se perturber les unes les autres.

Voici un schéma qui indique les étapes nécessaires à la compilation et à l'exécution d'un programme Android standard :

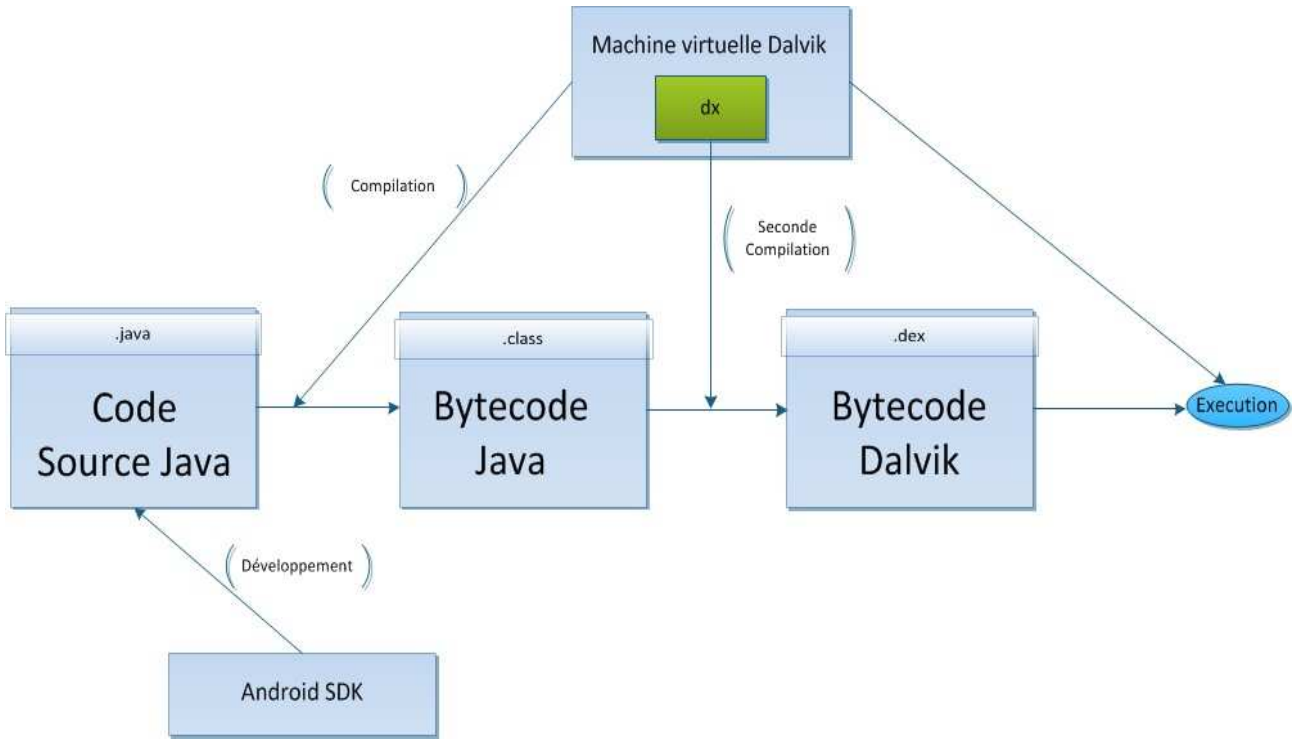


Figure 1.3:schéma d'exécution d'un programme Android standard

On voit bien que le code Java est converti en bytecode Java comme auparavant. Mais étant donné que, le bytecode java ne pouvant être lu que par une machine virtuelle java, et que Dalvik n'en est pas une, il faut donc procéder à une autre conversion à l'aide d'un programme qui s'appelle « dx ». Ce programme s'occupera de traduire les applications de bytecode Java en bytecode Dalvik, qui lui est compréhensible par la machine virtuelle.

4.4. Plate-forme Applicative :

Cette couche permet quant à elle de mutualiser au maximum les ressources entre applications Java. Elle propose également un moyen pour ces applications d'échanger des données. On retrouve par exemple dans cette couche la boîte à outils graphiques qui permet d'afficher des boîtes de dialogue,

des boutons, des menus, etc. C'est cette couche qui est rendue disponible au développeur Java au moyen d'un ensemble d'API.

4.5. Les applications :

La dernière couche, la seule finalement dont l'utilisateur aura à se préoccuper, est celle des applications. Elles sont sous forme de paquets .apk, qui permettent une installation et une désinstallation facile. Certaines sont fournies par l'équipe Android : navigateur web, gestionnaire de contacts, agenda, etc. C'est ici donc que nous intervenons, « cette stèle est celle de vos futures applications développées en Java. »[].On accède aux mêmes ressources que les applications fournies par défaut, on aura donc autant de possibilités qu'elles, on a même la possibilité de les remplacer. C'est aussi ça la force d'Android!

5. Le Système de Fichiers Android [10]:

Un système de fichiers est une façon d'organiser et de stocker une arborescence sur un support (disque, disquette, cd …). Chaque OS propriétaire a développé sa propre organisation.

Le processus de montage, avec sa commande *mount*, est le moyen de faire correspondre parties de l'arborescence et partitions physiques du disque. Il permet de plus d'affecter tout système extérieur à un répertoire créé pour cela dans l'arborescence.

Il suffira ensuite de se déplacer à ce répertoire, appelé point de montage, pour accéder à ses fichiers.

5.1. Arborescence du système Linux :

La racine est le sommet de la hiérarchie des répertoires. Il s'agit d'une arborescence logique, indépendante de l'implantation physique des divers sous-répertoires, qui peut s'étendre sur plusieurs partitions incluses sur un ou plusieurs disques, et même sur des disques réseaux. Sa structure est standard, avec des extensions imposées par les distributions.
Toute modification est de la compétence exclusive de l'administrateur, à l'exception des répertoires personnels situés dans */home*.

5.2. Liste des répertoires principaux et leurs rôles :

/ Le répertoire racine

- /bin : contient les fichiers exécutables (en binaire) (initialisation du système + commandes "essentiels")

- /boot : le noyau *vmlinuz* et les fichiers de démarrage.
- /dev : répertoire de fichiers spéciaux, qui servent de canaux de communication avec les périphériques (adaptateur réseau, cartes son etc. …)
- /etc : les fichiers de configuration du système et les principaux scripts de paramétrage
    - /etc /rc.d *scripts de démarrage du système*
    - /etc /X11 *scripts de configuration du serveur X*
    - /etc /init.d *scripts de contrôle des serveurs*
    - /etc /cron.d *description des tâches périodiques à effectuer*
    - /etc /skel *fichiers recopiés dans le rép. Personnel d'un nouvel utilisateur*
- /home : la racine des répertoires personnels des utilisateurs
- /lib : les bibliothèques et les modules du noyau
- /mnt : la racine des points de montage des périphériques.
- /opt : lieu d'installation d'applications supplémentaires (comme starOffice, java...)
- /root: répertoire personnel du super utilisateur root
- /sbin : les fichiers exécutables pour l'administration du système
- /tmp : stockage des fichiers temporaires
- /usr : programmes accessibles à tout utilisateur
- /var : données variables liées à la machine (fichiers d'impression, traces de connexions http, smb .. dans /var/log)
- /proc : ce pseudo-répertoire contient une "image" du système (/proc/kcore est l'image de la RAM)

Android, bien que basé sur le noyau Linux, ne possède pas les outils auxquels on est habitué. Le shell, par exemple, est extrêmement sommaire, et beaucoup d'utilitaires " classiques " sont absents (cp, grep, find...). De plus, la structure du système de fichier est pour le moins étrange : un répertoire /system/bin au lieu de /bin, pas de dossier */home, /etc* est un lien symbolique vers */system/etc*...

Les fichiers de configuration, que l'on s'attend à retrouver dans /etc, sont en fait pour la plupart stockés dans une base de données.

À ces répertoires sont en fait associées (au sens *mount*) les images système suivantes :

- ramdisk.img est associée à la partition racine /, au format rootfs,
- system.img est associée au dossier /system, au format yaffs2, conçu pour la mémoire flash,

- dans l'émulateur, /data est associé au fichier ~/.android/SDK/userdata-qemu.img, en yaffs2,
- enfin, /sdcard correspond sur le téléphone à la carte SD, au format FAT – sur l'émulateur,

6. Les différentes versions d'android [6]:

Les différentes versions d'Android ont toutes des noms de desserts (en anglais) depuis la sortie de la version 1.5 et suivent une logique alphabétique (de A vers Z) :

- 1.0 : *Apple Pie* (Tarte aux pommes) ou *Alpha*, Version connue uniquement ou presque des développeurs car c'est la version du Sdk distribué avant la sortie du premier téléphone Android, Sdk distribué fin 2007,
- 1.1: *Banana Bread* (Cake à la banane) ou *Beta*, Version incluse dans le premier téléphone, le HTC G1/Dream,
- 1.5: *Cupcake* (Petit Gâteau), sortie en avril 2009, dernière révision officielle en mai 2010,
- 1.6: *Donut* (Beignet), sortie en septembre 2009, dernière révision officielle en mai 2010,
- 2.0 (2.0.1): Version appelée *Éclair* au départ mais, à cause de nombreux bugs, vite remplacée par la 2.0.1 puis par la 2.1, cette version 2.0 est très peu connue,
- 2.1: *Éclair*, sortie en janvier 2010, dernière révision officielle en mai 2010,
- 2.2 (2.2.3): *FroYo* (*Frozen Yogourt*: Yaourt glacé), sortie en mai 2010, dernière révision officielle en 2011,
- 2.3 (2.3.7): *Gingerbread*(Pain d'épice), sortie le 6 décembre 2010, version actuelle pour Smartphones et petites tablettes,
- 3.0 (3.2): *Honeycomb* (Rayon de miel), sortie le 26 janvier 2011, version actuelle pour grandes tablettes et télévisions connectés,
- 4.0 (4.0.4): *Ice Cream Sandwich* (Sandwich à la crème glacée), version unifiée pour Smartphone, tablette et Google TV, fortement inspiré d'Honeycomb, sortie le 19 octobre 2011,
- 5.0: *Jelly Bean* (Jelly Bean, dragibus pour les Français), version qui devrait être installable sur les netbook dont la sortie est prévue en été ou à l'automne 2012
- 6.0: *Key Lime Pie*, nom supposé pour la version suivant Jelly Bean,

7. Les différents supports d'android [6]:

7.1. Les Smartphones :

Le premier mobile commercialisé sous Android est le HTC G1/Dream produit par la firme Taiwanaise HTC, lancé aux États-Unis le 22 octobre 2008.

Figure 1.4: Le téléphone G1.

En novembre 2009, Motorola a lancé aux États-Unis le Droid (commercialisé ailleurs dans le monde sous le nom de Motorola Milestone), le premier téléphone muni de la version 2.0 d'Android (nom de code Éclair). L'appareil qui a trouvé 250 000 acheteurs une semaine après son lancement fut lancé au Canada début février 2010.

Le 5 janvier 2010 Google annonce le Nexus One, téléphone conçu par la firme de Mountain View et sous-traité par HTC. Doté de caractéristiques alors assez impressionnantes (écran AMOLED de 3,7 pouces, processeur de 1 GHz, 512 Mo de mémoire RAM et Android 2.1), il devait également avoir l'avantage de recevoir directement ses mises à jour de Google, c'est-à-dire très rapidement. Il fut en effet le premier Smartphone à bénéficier d'Android 2.2 FroYo en juin 2010. Cependant ses ventes n'ont pas été exceptionnelles: environ 20 000 exemplaires la semaine de sa sortie.

En décembre 2010, Samsung fabrique le Nexus S pensé par Google sous Android 2.3.

Le 18 octobre 2011, Samsung et Google dévoilent le Samsung Galaxy Nexus, premier Smartphone sous Android4 "Ice Cream Sandwich". Celui-ci intègre le déverrouillage par reconnaissance faciale, l'utilisation de boutons virtuels et un système de reconnaissance vocale avancé.

Certains développeurs ont réussi à porter Android sur d'autres Smartphones comme le Nokia N9, dont le portage a été réalisé début 2012, ou l'iPhone dont le portage a été réalisé début mai 2010 par David Wang, le portage ayant été effectué avec plusieurs versions.

7.2. Les téléviseurs :

Le 5 avril 2010, la première télévision sous Android est dévoilée. Celle-ci est développée par l'entreprise suédoise People of Lava et se nomme Scandinavia. Elle possède les applications Facebook, YouTube, Google Maps et Twitter, possède un navigateur Web ainsi qu'un client de messagerie électronique.

7.3. Les Tablettes :

En septembre 2010, Samsung présente à Berlin le Samsung Galaxy Tab, tournant sous Android 2.2 (FroYo) et sortie fin 2010. Archos avec sa génération 7 de tablettes internet introduit Android (lancée en septembre 2009). Dans la même lignée, les tablettes Archos de la génération 8 (Gen 8) intègrent Android 2.2 (FroYo).

Motorola a présenté en 2011 la Xoom, première tablette bénéficiant de la nouvelle version du système mobile de Google, Honeycomb (Android 3.0).

7.4. Autoradios :

La société française Parrot SA a dévoilé en 2011 le premier autoradio tournant sous Android, la Parrot Asteroid. Cet autoradio offre notamment un adaptateur GPS, des ports USB et une connectivité Bluetooth pour contrôler la musique de son téléphone mobile.

7.5. Netbook :

La version 5.0 (Jelly Bean) devrait pouvoir être intégré aux minis portables, comme Chrome Os avec une possibilité de dual-boot entre ce dernier et Android.

8. Android coté développeur [4] :

Il n'est pas nécessaire de posséder un téléphone Android pour pouvoir utiliser la plate-forme. En effet, Google fournit un kit de développement (Software Development Toolkit ou SDK) destinée aux développeurs du téléphone, et dispensable sous différents systèmes Linux, Windows et Mac OS. Cela permet d'effectuer des tests préliminaires rapidement sur la station de développement, sans prendre aucun risque en cas de mauvaise manipulation.

Le SDK Android est composé de plusieurs éléments pour aider les développeurs à créer et à maintenir des applications :

- des API (interfaces de programmation) ;
- des exemples de code ;

• de la documentation ;

• des outils – parmi lesquels un émulateur – permettant de couvrir quasiment toutes les étapes du cycle de développement d'une application.

8.1. Développement d'application :

Sur un mobile, la capacité mémoire est limitée, la puissance CPU forcément plus réduite que celle du poste de développement. De plus, un téléphone portable possède une batterie qu'il s'agit de ménager et quiconque ayant déjà écrit des programmes pour des PDA (Personal Digital Assistant) ou des téléphones portables s'est heurté aux inconvénients de leur miniaturisation :

● Les écrans sont sous-dimensionnés.

● Les claviers, quand ils existent, sont minuscules.

● Les dispositifs de pointage, quand il y en a, sont peu pratiques ou imprécis.

En outre, les applications qui s'exécutent sur un téléphone portable doivent gérer le fait qu'il s'agit justement d'un téléphone. Les utilisateurs sont généralement assez irrités lorsque leur mobile ne fonctionne pas, et c'est la raison pour laquelle la campagne publicitaire "Can you hear me now?" de Verizon a si bien fonctionné ces dernières années. De même, ces mêmes personnes seront très mécontentes si un programme perturbe leur téléphone, pour les raisons suivantes :

● Il occupe tellement le processeur qu'elles ne peuvent plus recevoir d'appels.

● Il ne s'intègre pas correctement au système d'exploitation de leur mobile, de sorte que l'application ne passe pas en arrière-plan lorsqu'elles reçoivent ou doivent effectuer un appel.

● Il provoque un plantage de leur téléphone à cause d'une fuite de mémoire.

Le développement de programmes pour un téléphone portable est donc différent de l'écriture d'applications pour des machines de bureau, du développement de sites web ou de la création de programmes serveurs. On finira par utiliser des outils et des Frameworks différents et nos programmes auront des limites auxquelles on n'est pas habitué.

Android essaie de faciliter les choses :

● Il fournit un langage de programmation connu (Java), avec des bibliothèques relativement classiques (certaines API d'Apache, par exemple), ainsi qu'un support pour les outils auxquels on est habitués (Eclipse, notamment).

● Il offre un Framework suffisamment rigide et étanche pour que les programmes s'exécutent "correctement" sur le téléphone, sans interférer avec les autres applications ou le système d'exploitation lui-même.

Cependant, on ne code pas une application Android comme on coderait un programme Java standard destiné à être exécuté sur un poste de travail. « *Il convient de se méfier des raccourcis : le fait*

d'utiliser Java et Eclipse ne signifie pas qu'on ne doit pas adapter ses pratiques de développement ».[4]

Android est un environnement embarqué, certaines règles s'imposent donc.

8.2. Règles de programmation :

Si une application Android est une application embarquée, elle ne peut tout de même pas se classer dans la catégorie des applications temps réel critique. La plateforme intègre donc un Garbage Collector (GC), les objets créés et déréférencés seront bien éliminés de la mémoire. Néanmoins, ce qui est valide pour un programme Java classique est d'autant plus vrai sur environnement contraint. À chaque passage du GC, la machine virtuelle a tendance à marquer un temps d'arrêt. Au niveau de l'interface graphique, cela génère une impression négative, gâchant l'expérience utilisateur. Il est donc préférable de minimiser la création d'objets temporaires.

Pour améliorer les performances, il est recommandé également de déclarer « static » les méthodes pouvant l'être, c'est-à-dire celles ne modifiant pas les attributs de la classe, et de bien tagger les constantes avec le modificateur « final » .

Il est aussi plus performant d'éviter les appels de méthodes au travers d'interface. Par exemple, la « API Java Collections » définit les interfaces Map, Set, List, SortedSet, SortedSet... Plusieurs implémentations de ces interfaces sont disponibles :

Pour l'interface List, il y a, entre autres, les classes ArrayList et LinkedList. Chacune de ces implémentations fournissant donc la même API, elle offre des niveaux de performance variables selon les usages (lecture seule, modification fréquente...). Les tutoriaux officiels de Sun préconisent par conséquent de déclarer les objets avec leur interface afin de s'autoriser à changer le choix de l'implémentation si besoin est. Il faudrait alors écrire :

```
List<Object> maList = new ArrayList<Object>();
```

et non:

```
ArrayList<Object> maList = new ArrayList<Object>();
```

Sur Android, il est préférable de faire le contraire. Bien sûr, ce principe doit aussi être appliqué avec parcimonie. Si le code en question constitue une API publique, la neutralité apportée par les interfaces vaut probablement la petite perte de performance. Depuis la version 5 (JDK 1.5) de Java, un nouveau mot-clé enum a été introduit au langage. Ce mot-clé sert à définir les types énumérés remplaçant avantageusement les listes de constantes.

Avant l'ajout de ce mot-clé, la notion de type énuméré était implémentée comme ceci :

```
public final static int BANANE = 1;

public final static int POMME = 2;

public final static int ORANGE = 3;
```

Avec l'enum cela donne :

public enum Fruit{BANANE, POMME, ORANGE}

Cette dernière forme est plus robuste que la précédente car elle apporte la garantie à la compilation qu'aucune valeur autre que celles prévues ne pourra être employée pour un type énuméré donné. Les énumérés étant aussi des classes, ils possèdent certaines méthodes utiles à leurs manipulations : valueOf(arg), values().

Le revers de la médaille est qu'utiliser un énuméré est plus coûteux en mémoire et en cycles CPU que des constantes. Sur une plateforme limitée comme un téléphone, ce surcoût peut être significatif. Comme toujours, dans le cadre de la définition d'une API publique, on pourra par contre privilégier la robustesse et la clarté du code à sa performance.

Plus généralement, il est souhaitable de bien avoir conscience des instructions VM et même processeur qui se cachent derrière chaque fragment de code source. L'usage d'une construction telle que « for each » sur un objet Iterable génère par le compilateur l'appel à la méthode iterator() puis des méthodes hasNext() et next() à chaque boucle. Pour parcourir une ArrayList qui utilise comme structure de données interne un tableau, il est plus efficace de le faire directement avec la méthode get (index).

Il y a d'autres cas analogues. Par exemple, l'accès à une variable membre ou à une méthode privée depuis une classe interne non statique ne se fait en réalité qu'à travers des méthodes dites « synthetic methods », générées par le compilateur afin d'atteindre ces variables et méthodes aux visibilités réduites. Si on augmente la visibilité de ces éléments, ils deviendront accessibles depuis la classe interne et il ne sera plus nécessaire de passer par des méthodes intermédiaires autogénérées : on gagnera ainsi en efficacité.

9. Conclusion :

Ce chapitre est une introduction à la plate-forme android, à ses outils et son environnement de travail. Rappelons les points clés d'Android en tant que plate-forme :

• Elle est innovante car toutes les dernières technologies de téléphonie y sont intégrées : écran tactile, accéléromètre, GPS, appareil photo numérique etc.

• Elle est accessible car en tant que développeur on n'aura pas à acheter de matériel spécifique, ni à connaître un langage peu utilisé ou spécifique : le développement sur la plate-forme Android étant réalisé en langage Java, un des langages de programmation les plus répandus.

• Elle est ouverte parce que la plate-forme Android est fournie sous licence open source, permettant à tous les développeurs – et constructeurs – de consulter les sources et d'effectuer les modifications qu'ils souhaitent.

1. Introduction :

Actuellement, presque tous les systèmes informatiques utilisent des environnements graphiques disposant de plusieurs outils : Navigateurs Web, Outils d'exploration de fichiers et répertoires, Icônes pour applications et jeux, … etc

Toutefois, les interprètes de commande sont toujours un passage obligé pour l'utilisation (et surtout l'administration) des systèmes informatiques.

> « Un interprète de commande est une interface efficace pour passer des ordres et des commandes au système. »[5]

Les débuts des interprètes de commandes remontent aux débuts des années 60, à une époque où un écran 2 couleurs était un luxe inimaginable et où la puissance de calcul de ces ordinateurs était cent fois plus faible que celle d'une calculatrice. Gérer une interface graphique avec plusieurs couleurs ainsi qu'une souris et un certain nombre de fonctionnalités avancées qui nous paraissent aujourd'hui « normales», demandait de la puissance, L'interprète de commande était donc à cette époque la seule façon d'utiliser un ordinateur.

2. Historique [11]:

Les premiers systèmes capables d'interpréter des lignes de commandes sont donc apparus au début des années 1960, en même temps que le clavier informatique. Auparavant, les ordinateurs étaient uniquement utilisés en traitement par lots.

2.1 Sous Microsoft Windows :

Sous Windows, seule l'invite DOS existe. Elle se lance par l'utilitaire *COMMAND.COM* ou *cmd.exe*. Jusqu'à Windows 3.x, Windows n'était qu'une interface graphique du DOS, mais a commencé à proposer plus de fonctionnalités que lui à partir de Windows 95. La famille de Windows NT, jusqu'à Windows XP qui en est la version 5.1, se passe presque intégralement de la ligne de commande, et l'invite de commandes qu'elle propose n'est qu'un émulateur, largement bridé, de MS-DOS. Depuis le 24 mars 2009, PowerShell 1.0 est distribué comme une mise à jour logicielle facultative par le service Windows Update de Microsoft pour Windows XP et Vista, et est intégré nativement dans Windows 7 en version 2.0.

2.2 Sous les systèmes UNIX :

Sous UNIX, la ligne de commande a toujours été le moyen privilégié de communication avec l'ordinateur.

GNU/Linux, la famille BSD et autres dérivés d'UNIX ont hérité de cette particularité, même s'ils disposent également d'interfaces graphiques complètes.

Parmi ces dérivés, Mac OS X se présente comme un environnement essentiellement graphique, mais dispose d'un interpréteur de commandes (tcsh ou bash) qui s'active à partir du programme Terminal.

3. Interprète de commande [10]:

Un interprète de commande est un programme externe qui reçoit une commande, ou une liste de commandes sous forme d'un fichier texte, et qui les exécutent une à une.

Quand la série de commandes est fournie sous forme de liste, cette dernière est appelée Script. Il existe de nombreux interpréteurs de commande. Nous citons: Le Shell d'Unix, Sed, Awk, Perl, Python, Tcl/Tk,…etc.

4. Programme Interprète VS Programme Compilé [5]:

En informatique, un programme écrit en langage interprété n'est pas exécuté directement par la machine mais par un autre programme appelé interprète ; il doit être en fonctionnement sur la machine où l'on veut lancer un programme interprété. Au contraire, un programme écrit en langage compilé est traduit en instructions lisibles par la machine (code natif) et illisible pour un humain et peut être exécuté indépendamment de tout autre programme (à l'exception du système d'exploitation, dans la plupart des cas). Tandis qu'un programme interprété est composé de commandes compréhensibles et peut renfermer en lui sa propre documentation sous forme de commentaires. Quelques exemples de langages parfois interprétés : BASIC, PHP, JavaScript, Ruby, Perl, etc : les langages de script en général.

Il existe aussi des langages dits semi-interprétés ou semi-compilés, pour lesquels il existe un compilateur traduisant le programme non pas en « langage-machine » mais en un code intermédiaire assez analogue à de l'assembleur. Pour pouvoir exécuter ces programmes sur une machine donnée, il faut y faire tourner un interpréteur pour ce code intermédiaire. Le code intermédiaire est souvent appelé p-code, Byte Code, code objet… ; l'interpréteur, lui, peut être appelé p-machine ou machine virtuelle.

Java est sans doute le plus célèbre des langages semi-interprétés, se basant sur la machine virtuelle Java (JVM). Un autre exemple de langage semi-interprété est le Pascal dans sa version UCSD ou encore Python.

Certains langages à l'origine uniquement interprétés deviennent également des langages compilés ou semi-compilés pour des raisons de performance. Ils ne sont parfois compilés qu'au début de leur exécution, voire au fur et à mesure de celle-ci (on parle alors de compilation à la volée — ou JIT).

C'est le cas par exemple de C□ (JIT) mais aussi de langages comme Python avec des implémentations alternatives comme PyPy (JIT), Ruby avec YARV (semi-compilation), PHP avec HipHop (compilation en code natif)…

Plus exigeants en ressources et la plupart du temps moins rapides à l'exécution que les langages compilés en code natif, les langages interprétés gardent toutefois un intérêt notamment par leur facilité de mise en œuvre et la portabilité des programmes, qui peuvent, la plupart du temps, être lancés sans modification sur toute plateforme où fonctionne l'interpréteur.

Pour la suite de ce chapitre, nous nous intéresserons plus spécialement à l'interprète de commande d'UNIX (shell) vu que GNU/Linux en a hérité et que le système Android est basé sur linux.

5. Avantages des interprètes de commandes[10] :

De nos jours, avec la prolifération des **systèmes d'exploitation avec interface graphique** (Mac OS, Windows, Linux + KDe, Linux + Gnome), on pourrait se demander pourquoi on n'a pas supprimé l'interprète de commande. C'est là que beaucoup se trompent : quand on sait s'en servir, on va beaucoup plus vite qu'avec l'interface graphique. On se rendra compte à un moment qu'il y a des choses que seule l'interprète peut faire et qu'il serait de toute façon vraiment inutile de recourir à une interface graphique pour les effectuer.

- **Exemple avec le shell d'UNIX :**

En mode graphique, aller dans un répertoire qui contient beaucoup de fichiers : des fichiers texte, des images, des vidéos...et on voudra savoir combien il y a d'images JPEG dans ce dossier.

En assemblant quelques commandes :

```
ls -l | grep jpg | wc -l
  54
```

La première ligne est la commande, la seconde le résultat. Il y avait donc 54 images JPEG dans le dossier, et on a obtenu le résultat en moins d'une seconde !

On peut même enregistrer directement ce nombre dans un fichier texte :

```
ls -l | grep jpg | wc -l > nb_jpg.txt
```

... et on peut aussi envoyer le fichier nb_jpg.txt sur Internet par FTP ou par e-mail, le tout en une ligne .

Donc :

- Un interprète de commande est une interface efficace pour passer des ordres et des commandes au système
- Il est plus rapide et nécessite peu de ressources (mémoires, processeur, …etc.)

- Il permet d'exécuter des actions intelligemment et automatiquement dans des situations particulières (démarrage du système, lancement des applications, analyse des fichiers journaux, …etc)
- **il permet d'utiliser un logiciel en ligne de commande**, par exemple **MySQL** qui dispose de nombreux outils en ligne de commande, pas tous interfacés dans PHPMyAdmin.
- Automatiser des tâches d'administration : création des utilisateurs, planification des sauvegardes ou des tâches diverses...
- Mais surtout ,un interpréteur de commande est accessible à distance via internet.

6. L'accès à distance des interprètes de commande :

Une des forces de l'interprète de commande, c'est d'être accessible à distance par Internet. Il suffit qu'une machine soit connectée au réseau pour que l'on puisse s'y loger de n'importe quel ordinateur dans le monde et faire comme si on était devant notre machine, et cela est pratique pour un certain nombre de tâches :

- Surveiller l'état d'un téléchargement un peu long,
- Lancer l'exécution d'un programme pour qu'il soit prêt lorsque vous serez rentrés chez vous...
- Mais surtout, c'est comme ça que l'on administre un serveur sous Linux.

Un serveur est un ordinateur tout le temps connecté à Internet. Il permet d'offrir des services divers et variés aux internautes. Par exemple, il y a des serveurs web dont le rôle est de distribuer des pages web.

La grande majorité des serveurs tourne sous Linux. Les serveurs Windows existent aussi, mais ils sont plus rares et on apprécie en général la stabilité de Linux ainsi que la possibilité de l'administrer à distance en ligne de commande.

Pour qu' un ordinateur et un serveur puisse communiquer, il faut un protocole. C'est un ensemble de règles pour que deux ordinateurs puissent discuter entre eux... un peu comme si deux personnes devaient parler la même langue pour avoir une conversation.

Il existe plusieurs protocoles pour communiquer par Internet, mais pour ce qui est d'accéder à la ligne de commande à distance, c'est-à-dire à la console, il y en a deux principaux.

- *Telnet :* le protocole le plus basique, qui présente le gros défaut de ne pas crypter les données échangées entre nous et le serveur. Si un pirate « écoute » nos échanges par un moyen ou un autre, il pourrait récupérer des informations sensibles, en particulier notre mot de passe lorsqu'on l'envoie à la connexion. Ce moyen de connexion reste utilisé mais peu par rapport à SSH.
- *SSH* : est de très loin le protocole le plus utilisé car il permet de crypter les données et de sécuriser ainsi la connexion avec le serveur.

❖ ***PuTTY:***

Pour accéder à distance à un ordinateur sous Linux connecté à Internet, On a besoin d'un programme spécial capable de restituer la ligne de commande à distance. Ce qui est bien, c'est qu' on a pas forcément besoin d'être sous Linux pour se connecter à un autre ordinateur utilisant cet OS ; on peut très bien le faire depuis Windows, et c'est d'ailleurs la procédure qui est montrée ici.

Il existe plusieurs programmes capables de se connecter en SSH à un serveur Linux. Le plus célèbre sous Windows est sûrement PuTTY : il est gratuit, léger et ne nécessite même pas d'installation (juste un exécutable à lancer).

7. Quelques notions de base :

Figure 2.1 : la console**.**

Cette fenêtre noire représente l'interprète de commande (la console, le terminal) où l'on va écrire nos commandes.

7.1 L'invite de commande :

Ce qu'on voit à la figure 2.2, est ce que l'on appelle l'invite de commande (en anglais prompt). C'est un message qui nous invite à rentrer une commande. Il s'affiche avant chaque commande qu'on tape.

«dehia»: le premier élément est le pseudonyme. C'est le pseudonyme sous lequel on s'est logé. En effet, on peut créer plusieurs comptes utilisateurs sous Linux. Il est en général conseillé d'en générer un par personne susceptible d'utiliser l'ordinateur.

«@» : ce symbole n'indique rien de particulier. C'est le symbole « at » qui signifie « chez ». Si on lit l'invite de gauche à droite, on doit donc comprendre « dehia chez ».

«dehia-Lenovo-3000-N500» : c'est le nom de l'ordinateur sur lequel on est en train de travailler. Dans notre cas il s'appelle dehia-Lenovo-3000-N500 mais on aurait pu lui attribuer n'importe quel autre nom lors de l'installation.

La ligne d'invite de commandes se lit donc « dehia chez dehia-Lenovo-3000-N500 ».

«:» : à nouveau, ce symbole ne veut rien dire de spécial, c'est un séparateur.

«~» : représente le dossier dans lequel on se trouve actuellement. On peut naviguer de dossier en dossier dans la console et il est très utile qu'on nous rappelle systématiquement où on se trouve avant chaque commande. Le symbole ~} signifie qu'on est dans notre dossier personnel, ce qu'on appelle le « home » sous Linux ; c'est l'équivalent du dossier « Mes documents » de Windows.

«$» : ce dernier symbole est très important ; il indique notre niveau d'autorisation sur la machine. Il peut prendre deux formes différentes :

$: signifie qu'on est en train d'utiliser un compte utilisateur « normal », avec des droits limités (on ne peut pas modifier les fichiers système les plus importants). Notre compte dehia est donc un compte normal avec des droits limités ;

: signifie qu'on est en mode superutilisateur, c'est-à-dire qu'on est connecté sous le pseudonyme « root ». Le root est l'utilisateur maître qui a le droit de tout faire sur sa machine.

7.2 Notion de ligne de commande :

Une ligne de commande est une chaîne de caractères constituée d'une commande, ainsi que des arguments (paramètres) optionnels.

7.3 La commande :

Une commande correspond à un fichier exécutable du système ou bien d'une commande du shell

elle semble être un amas de lettres vide de sens ce qui est totalement faux. Tout a été minutieusement pensé, et ce dès les années 60.

- Les commandes sont courtes, abrégées. C'est pour gagner du temps et aller plus vite. Écrire pwd est moins intuitif que «diredansquelrepertoirejesuis »,
- Les commandes sont intuitives. Il s'agit bien souvent d'une abréviation de termes (en anglais) et les lettres qu'il faut taper sont généralement choisies en fonction de leur proximité les unes par rapport aux autres pour qu'on ait le moins possible à

déplacer les doigts sur le clavier (vrai plutôt pour les claviers QWERTY anglais qui sont plus adaptés pour accéder aux symboles du genre {] | #, etc).

Donc une commande est constituée d'un mot et ne contient aucun espace. Dans des cas très simples comme ceux que l'on vient de voir, il suffit juste de taper la commande pour avoir une réponse ; mais dans la quasi-totalité des cas on peut (et parfois on DOIT) rentrer des options, qu'on appelle paramètres.

7.4 Les paramètres :

Les paramètres sont des options que l'on écrit à la suite de la commande. La commande et les paramètres sont séparés par un espace, comme ceci :

Code : Console

dehia@dehia-Lenovo-3000-N500:~$ commande paramètres

Les paramètres peuvent eux-mêmes contenir des espaces, des lettres, des chiffres... . Il n'y a pas de règles véritables sur la forme des paramètres, mais les programmeurs ont adopté une « convention » pour que l'on puisse reconnaître les différents types de paramètres.

- ***Les paramètres courts (une lettre)***

Les paramètres les plus courants sont constitués d'une seule lettre précédée d'un tiret. Par exemple :

Si on doit donner plusieurs paramètres, on peut faire comme ceci :

commande -d

Ou, plus court :

commande -d -a -U -h

commande -daUh

Il faut faire attention à la casse des paramètres (majuscules / minuscules), si on écrivit -u, cela n'a en général pas du tout le même sens que –U.

- ***Les paramètres longs (plusieurs lettres)***

Les paramètres constitués de plusieurs lettres sont précédés de deux tirets, comme ceci :

commande --parametre

Si on veut mettre plusieurs paramètres longs, il faudra ajouter un espace entre chacun d'eux :

commande --parametre1 --parametre2

On peut aussi combiner les paramètres longs et les paramètres courts dans une commande :

commande -daUh –autreparametre

7.5 Entrées-sorties standard :

Lors de l'exécution d'une commande, un processus est créé. Celui-ci va alors ouvrir trois flux :

- Stdin : appelé entrée standard, dans lequel le processus va lire les données d'entrée. Par défaut stdin correspond au clavier ; STDIN est identifié par le numéro 0.
- Stdout : appelé sortie standard, dans lequel le processus va écrire les données de sortie. Par défaut stdout correspond à l'écran ; STDOUT est identifié par le numéro 1.
- Stderr : appelé erreur standard, dans lequel le processus va écrire les messages d'erreur. Par défaut stderr correspond à l'écran. STDERR est identifié par le numéro 2.

Par défaut lorsque l'on exécute un programme, les données sont donc lues à partir du clavier et le programme envoie sa sortie et ses erreurs sur l'écran, mais il est possible de lire les données à partir de n'importe quel périphérique d'entrée, voir à partir d'un fichier et d'envoyer la sortie sur un périphérique d'affichage, un fichier, etc.

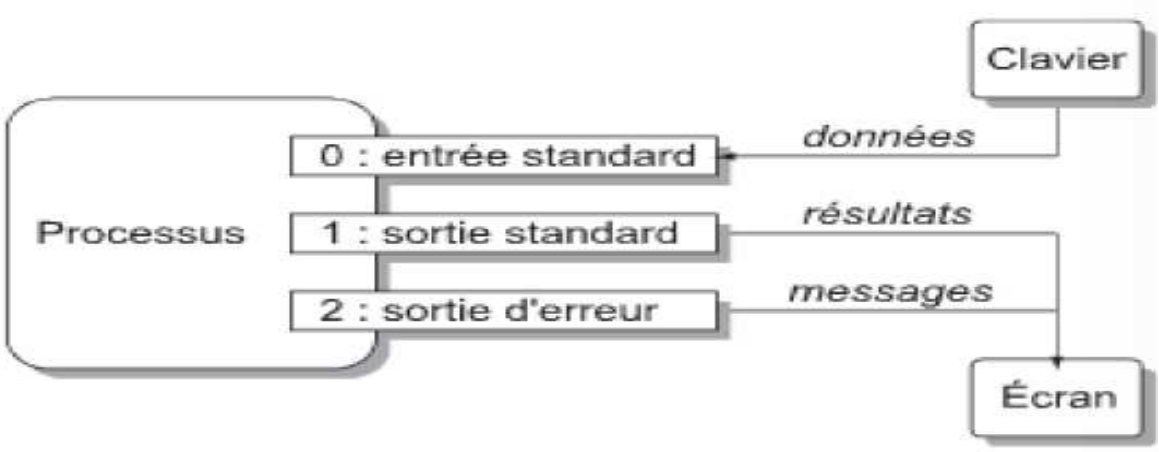


Figure 2 .2: les entrées-sorties standards.

Il est possible de réaliser une redirection de l'un ou plusieurs des fichiers d'entrées sorties

7.6 Redirections :

Linux, comme tout système de type Unix, possède des mécanismes permettant de rediriger les entrées-sorties standards vers des fichiers.

7.6.1 Les sorties standard :

Ainsi, l'utilisation du caractère «>» permet de rediriger la sortie standard d'une commande située à gauche vers le fichier situé à droite :

ls -al /home/jf/ > toto.txt

echo "Toto" > /etc/monfichierdeconfiguration

La commande suivante est équivalente à une copie de fichiers :

cat toto > toto2

La redirection «>» a pour but de créer un nouveau fichier. Ainsi, si un fichier du même nom existait, celui-ci sera écrasé. La commande suivante crée tout simplement un fichier vide :

> fichier

L'emploi d'un double caractère «>>» permet de concaténer la sortie standard vers le fichier, c'est-à-dire ajouter la sortie à la suite du fichier, sans l'écraser.

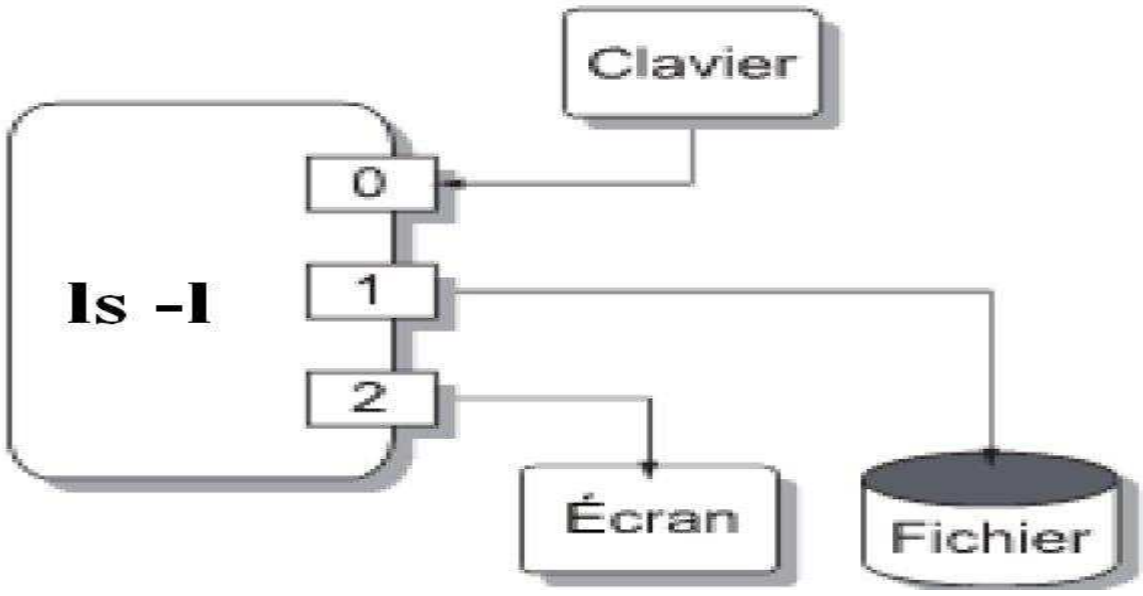


Figure 2.3:redirections de la sortie standard.

7.6.2 L'entrée standard :

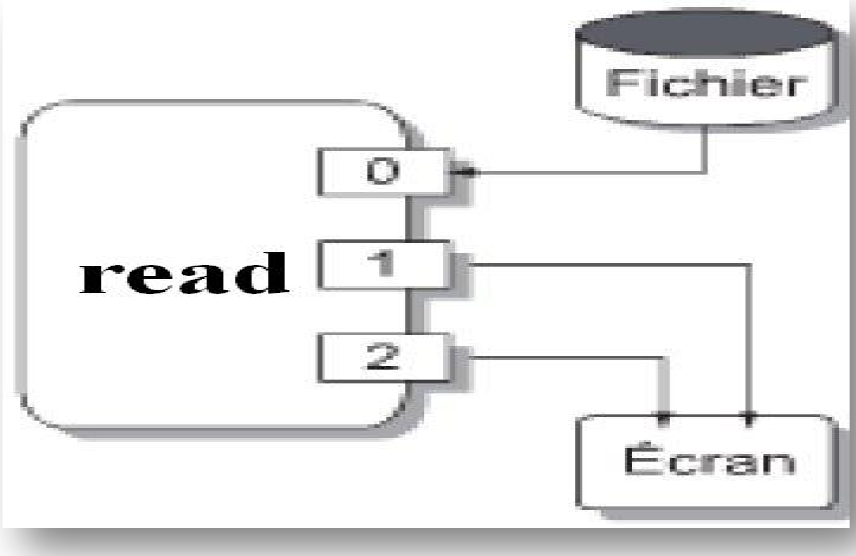
De manière analogue, le caractère «<» indique une redirection de l'entrée standard.

- **Exemple :**

read Ligne < fichier.txt lit la première ligne du fichier et l'affecte à Ligne

echo $Ligne affiche la première ligne du fichier

Enfin l'emploi de la redirection «<<» permet de lire sur l'entrée standard jusqu'à ce que la chaîne située à droite soit rencontrée. Ainsi, l'exemple suivant va lire l'entrée standard jusqu'à ce que le mot STOP soit rencontré, puis va afficher le résultat :



cat << STOP

Figure 2.4:redirection de l'entrée standard.

7.7 Redirection de la sortie d'erreur standard :

Il est même possible de rediriger Les erreurs d'une commande vers un fichier

- **Exemple :**

rm fichierInexistant 2> erreur.txt Recopie l'erreur engendrée par la tentative de suppression d'un fichier inexistant dans le fichier erreur.txt

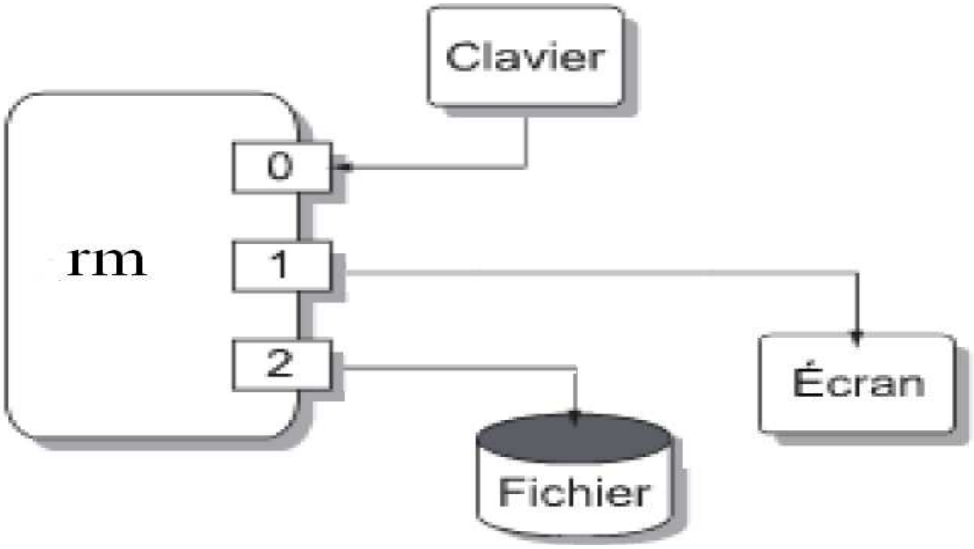


Figure 2.5: Redirection de la sortie d'erreur standard.

7.8 Pipes :

Un tube Unix, ou pipeline, ou pipe est un ensemble de processus chaînés par leurs flux standard, de sorte que la sortie d'un processus (stdout) alimente directement l'entrée (stdin) du suivant. Chaque connexion est implantée par un tube anonyme. Les programmes filtres sont souvent utilisés dans cette configuration. Douglas McLiroy a inventé ce concept pour les shells Unix et le nom anglais découle de l'analogie avec un pipeline physique. Le pipe est caractérisé généralement par le symbole |.

Par exemple :

programme1 | programme2

Le programme programme1 est exécuté par le système qui envoie les résultats au programme2 qui à son tour renvoie les résultats sur la sortie standard du système.

- **Exemple :**

ls -l | wc -l Compte le nombre de fichiers et répertoires du répertoire courant

Avec :

ls –l liste les fichiers et répertoires

wc –l compte le nombre de lignes d'un fichier

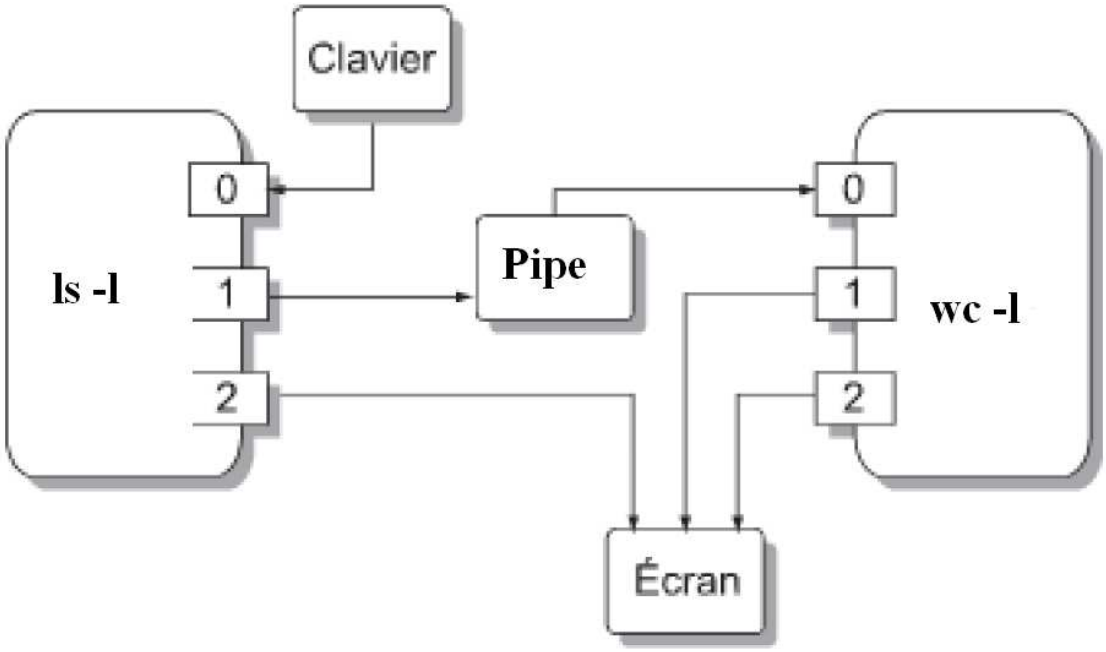


Figure 2.6 : exemple d'un pipe.

Plusieurs pipes peuvent être réunis sur une même ligne, ouvrant la voie à des connexions entre les différentes opérations, et ainsi la réalisation de script shell plus compact et mieux construit.

8. Fonctionnement de l'interpréteur de commandes (Shell) :

L'interpréteur de commandes est l'interface entre l'utilisateur et le système d'exploitation, d'où son nom anglais «shell», qui signifie «coquille».

Le shell est ainsi chargé de faire l'intermédiaire entre le système d'exploitation et l'utilisateur grâce aux lignes de commandes saisies par ce dernier.

Le travail de tout interprète de commande peut se résumer à l'algorithme très simple suivant:

TANT QUE l'utilisateur ne ferme pas la session

FAIRE

Émettre un signe d'invite

Lire la ligne courante

Exécuter la commande indiquée sur cette ligne

FIN

- **Exemple :**

dehia@dehia-Lenovo-3000-N500:~$ /bin/ls

Quand l'utilisateur appuie sur **Entrée** après avoir saisi **/bin/ls**, l'interpréteur de commandes (*shell*) crée un nouveau processus en parallèle . Ce processus invoque un appel système pour exécuter **bin/ls**. Le chemin complet, **/bin/ls** est passé à titre de paramètre ainsi que l'argument en ligne de commande (argv) et aux variables d'environnement (envp). Le gestionnaire d'appels système vérifie que le fichier existe. Si c'est le cas, il vérifie s'il est dans un format de fichier exécutable. Si le fichier est dans un format de fichier exécutable, le contexte d'exécution du processus en cours est modifié. Pour finir, quand l'appel système prend fin, **bin/ls** est exécutée et l'utilisateur voit la liste des répertoires.

Le shell est un programme comme un autre, entièrement externe au noyau. Il est donc possible de programmer de multiples shells, et les utilisateurs d'Unix ne s'en sont pas privés.

Il existe plusieurs shells, les plus courants étant sh (appelé «Bourne shell»), bash («Bourne again shell»), csh(«C Shell»), Tcsh («Tenex C shell»), ksh («Korn shell») et zsh («Zero shell»).

Le Bourne shell (sh) est l'interpréteur originel de l'environnement UNIX. À son époque, sa grande originalité était l'utilisation de tubes (caractère "|"), qui permettent de connecter la sortie d'une commande à l'entrée d'une autre. On peut ainsi écrire des commandes complexes à partir de commandes simples.

Chaque utilisateur possède un shell par défaut, qui sera lancé à l'ouverture d'un invite de commande. Le shell par défaut est précisé dans le fichier de configuration /etc/passwd dans le dernier champ de la ligne correspondant à l'utilisateur. Il est possible de changer de shell dans une session en exécutant tout simplement le fichier exécutable correspondant, par exemple :

/bin/bash

9. Les différents shells :

9.1 Shell de Bourne :

9.1.1 Variables, environnement :

La puissance d'un langage de programmation, quel qu'il soit, vient du fait qu'un même programme peut engendrer des calculs différents suivant l'environnement dans lequel il est exécuté. Cette flexibilité dépend de la présence d'instructions de contrôle et de l'existence de variables, qui permettent de désigner les objets manipulés par leur nom plutôt que par leur valeur. Pour le shell, les variables vont servir essentiellement à noter des paramètres de commandes, lesquels à leur tour sont le plus souvent des noms de fichiers. Les variables du shell de Bourne ont donc pour valeur des chaînes de caractères.

Une variable porte un nom, qui est lui aussi une chaîne de caractères. La valeur associée à la variable v se note $v. L'ensemble des associations nom → valeur existant à un instant donné forme l'environnement du processus shell.

Il existe plusieurs façons de donner une valeur à une variable. La plus simple est l'affectation, qui se note :

nom = valeur

Insistons sur le fait qu'il s'agit d'une affectation de suite d'octets. On remarquera au passage que l'affectation ne se conforme pas au schéma d'exécution donné auparavant. Après lecture, c'est le processus shell qui exécute l'affectation et non un processus fils créé pour la circonstance. Une telle commande est appelée commande interne ; il en existe quelques autres, dont la plus utilisée est cd qui permet de déplacer le répertoire de travail. Les variables sont conservées dans l'espace des données du processus shell. Lorsque ce processus lance un autre shell, les espaces de données (donc les systèmes de variables) sont indépendants, de sorte que les affectations de l'un n'influent pas sur les variables de l'autre.

9.1.2 Commandes et programmation à l'aide du shell :

Insistons sur le fait que le shell n'exécute pas directement les commandes (à l'exception des rares commandes internes). Pour le shell, une commande est un nom qui désigne un fichier exécutable. Cette disposition facilite la création de nouvelles commandes soit en combinant des commandes existantes , soit en programmant directement en C ou en tout autre langage.

En fait, tout utilisateur d'Unix un peu chevronné se crée son propre environnement de commandes.

Les commandes fournies avec le système ne sont qu'une base de départ pour cette construction. On y trouve :

— des commandes de manipulation et d'exploration du système de fichier : liste d'un répertoire (ls), copie et impression d'un fichier (cp, cat), renommage (mv), destruction (rm) ; la commande cd permet de déplacer le répertoire courant du shell qui l'exécute,

— des commandes d'exploration de l'environnement d'un utilisateur : date, ps (liste des processus actifs), who (liste des usagers connectés), etc. ;

— des commandes exécutant des traitements élémentaires sur des fichiers : sort (tri), wc (comptages), grep (recherches), etc. ;

— des outils de génie logiciel (compilateurs, interpréteurs, assembleurs, éditeurs de liens, éditeurs de texte) et des outils pour la documentation (traitement de texte nroff) ;

— des outils de communication (mail, uucp, write, readnews, etc.).

Ces commandes ont été conçues pour pouvoir interagir facilement ; par ailleurs, le shell est un langage de programmation puissant. Une tâche qui, sur un autre système, réclamerait l'écriture d'un programme important peut très souvent se réaliser par combinaison de commandes existantes.

9.1.3 Initialisation :

Lorsqu'on lance le premier shell (login shell), le nom du répertoire de travail est lu dans le fichier /etc/passwd. Le shell recherche dans ce répertoire un fichier de nom profile. Son éventuel contenu est interprété comme une suite de commandes d'initialisation. La constitution du fichier .profile est laissée au choix de l'usager ; on y trouve typiquement :

— la définition de variables d'environnement : HOME, le nom du répertoire initial, TERM, le type du terminal utilisé, et PATH, la liste des répertoires à explorer à la recherche d'une commande.

Ces variables doivent naturellement être exportées ;

— des commandes à exécuter à chaque connexion, comme par exemple la commande mail qui permet de lire le courrier.

9.2 Autres shells :

Le shell de Bourne est le plus utilisé, mais il a été conçu à une époque où le terminal standard était le Télétype. D'autres interpréteurs, plus récents, s'adaptent mieux au travail sur écran. Le shell à syntaxe C ***csh*** comporte deux innovations essentielles :

— l'historique des commandes ;

— la possibilité de faire passer une tâche d'avant-plan en arrière-plan et réciproquement à volonté.

csh conserve les plus récentes commandes et permet à l'usager d'en sélectionner une et de la

réexécuter en tout ou en partie. Dans la version tcsh, la commande sélectionnée est soumise à un éditeur de texte pleine ligne, ce qui donne la possibilité de la modifier ou de la corriger avant de la relancer.

9.2.1 Lancer un processus en arrière plan :

Pour lancer une commande quelconque, on en saisit le nom après le prompt du shell, tant que la commande n'est pas terminée, on n'a plus la main au niveau du shell, on ne dispose plus du prompt. Si la commande prend un certain temps, le shell ne nous donnera pas la main tant que la commande n'est pas terminée, on est obligé de lancer un autre shell, pour taper une autre commande. On dispose d'une technique simple qui permet de lancer une commande à partir d'un shell, et de reprendre aussitôt la main. Il suffit de rajouter un & à la fin de la commande. Celle-ci se lancera en " tâche de fond ", et on reviendra directement au prompt du shell.

En tapant une commande en tâche de fond, on aura à l'affichage :

$ ps ef &

6677 (PID du processus ramené en arrière plan)

Pour visualiser l'état d'exécution d'une commande lancée en arrière plan, on utilise la commande

$ jobs ---> cette commande donne le [numéro de tâche] (job), son [PID, son état

d'exécution (stopped, running, ...) et le nom de la commande

9.2.2 Ramener un processus en arrière/avant plan :

$ fg [N°tâche] ---> fg (foreground) permet de reprendre l'exécution de la tâche en

premier plan

$ bg [N°tâche] --->bg (background) permet de ramener l'exécution de la tâche en arrière

plan

9.2.3 Le shell de Korn :

Il reprend toutes les fonctionnalités du cshell, mais avec une syntaxe entièrement compatible avec celle du shell traditionnel de Bourne. Autrement dit, le shell de Korn, kshell, es tune extension du shell de Bourne : toute liste de commande qui fonctionne en shell de Bourne fonctionne aussi en shell de

Korn, l’inverse étant bien sûr faux. Enfin le shell POSIX est le shell normalisé de l’ISO. Pratiquement, il s’agit en fait du shell de Korn.

10. Utilité d'un interprète de commande pour Android :

Équiper un téléphone android d'un interprète de commande a plusieurs avantages :

- Android est un système basées sur le noyau linux :la plupart des commandes du shell linux devrait donc fonctionner sur android. On pourrait donc les utiliser pour manipuler les données de sa carte SD, par exemple: créer des répertoire, des fichiers,...etc. Cependant ces manipulations ne sont pas permises par le système,

- Un téléphone Android est destiné à des utilisateurs non avertis qui l'utilisent en général pour effectuer des appels, se connecter à internet, écouter de la musique,...etc. Ces fonctionnalités passent à travers des applications conventionnels chargés par le constructeur du téléphone ou installer par l'utilisateur. Ces applications s’exécutent en général, en mode utilisateur. les constructeurs désactivent pour cela le compte administrateur(root) pour ne pas endommager les applications systèmes et le téléphone lui même.

- Il se trouve qu'il est possible de réactiver le compte root, et dans ce cas, il serait possible d'installer directement des applications systèmes qui pourraient transformer le smartphone en un véritable ordinateur . Ce genre de manipulation nécessite l'utilisation d'un interprète de commande.

11. Conclusion :

L'idée force de se chapitre était de nous familiariser avec les interprètes de commandes, et plus spécialement le shell d'Unix, quand ils exécutent une commande.

La console affiche un invite de commandes au début de la ligne. Cette invite rappelle le nom d'utilisateur, le nom de la machine ainsi que le dossier dans lequel on se trouve.

On rentre des commandes dans la console pour demander à l'ordinateur d'exécuter des actions. Chaque commande peut être compléter de paramètres qui agissent comme des options pour modifier l'action de la commande.

Pour conclure, on peut dire que le rôle du shell a été de briser un cercle vicieux, qui veut que sous Unix, un processus ne peut être créé que par un processus déjà existant, qui doit contenir les instructions convenables.

Dans le chapitre suivant on s’attaquera à la création d'un shell pour le système Android.

1. Introduction :

Ce qui fait aujourd'hui le succès d'une plateforme n'est malheureusement pas toujours ses qualités mais principalement les logiciels qu'on peut y trouver. Le meilleur exemple : Windows.

Windows est en effet une plateforme décriée pour ces nombreux problèmes et pourtant elle équipe aujourd'hui plus de 90% des ordinateurs du monde. En ce qui concerne les plateformes mobiles le constat devrait être le même dans les années qui viennent car aujourd'hui les téléphones mobiles deviennent de véritables ordinateurs.

Conscient de cette situation, Google n'a donc pas hésité à proposer gratuitement un SDK (Kit de Développement) pour que tout le monde puisse facilement y développer des logiciels. Les avantages d' Android sur ce point est sans aucun doute le fait que ce SDK existe aussi bien sous Windows, MacOS X et Linux. Une véritable première dans le monde de la mobilité où les autres fabricants ne proposent qu'un SDK fonctionnant avec leur système (MacOS X pour Apple et Windows pour Microsoft).

Dans ce chapitre on va voir comment concevoir une application pour Android. Pour cela on va programmer un interprète de commande.

2. Les outils indispensables à télécharger :

Avant de commencer à développer, le SDK Android nécessite différents outils:

- Le SDK lui-même, proposé directement par Google
- Un environnement de développement

On prendra comme environnement de développement Eclipse qui, lui aussi, possède l'immense avantage de fonctionner sur différentes plateformes.

Le SDK Android est un zip, qu'il convient de décompresser.

3. Structure d'un projet [8] :

La structure d'un projet Android répond à des règles bien précises. Le meilleur moyen pour les appréhender est de commencer par créer un projet de test à l'aide du plug-in Eclipse d' Android.

Le menu Eclipse « File/New/Other... » amène à la boîte de dialogue depuis laquelle l'assistant de création de projet Android est accessible.

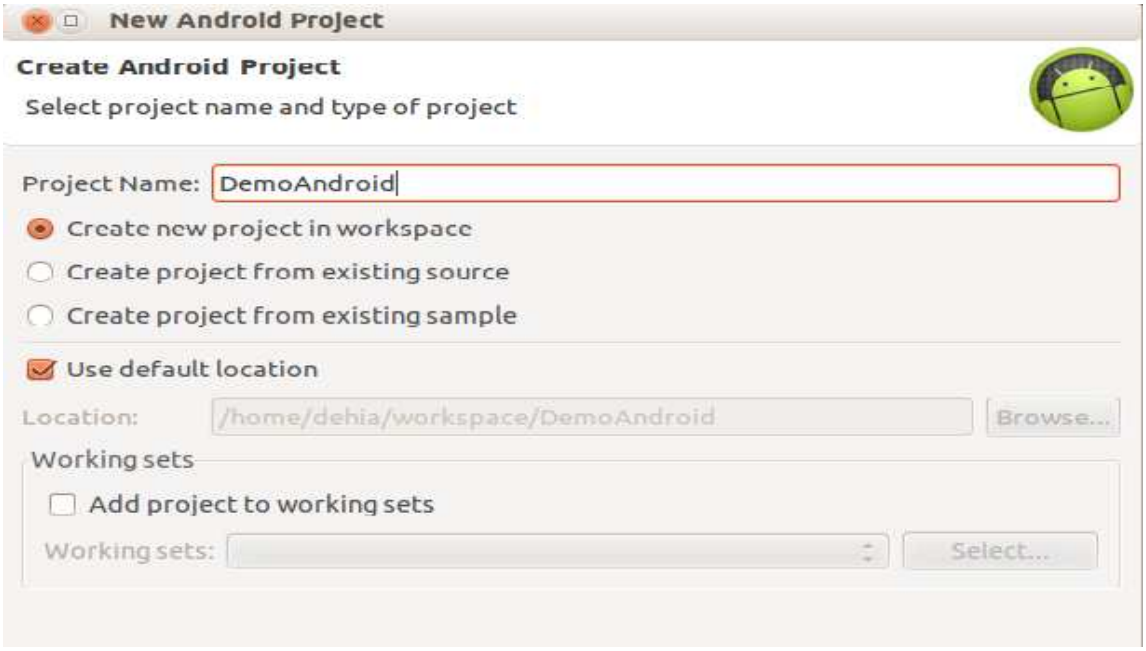


Figure 3. 1 : création du projet android.

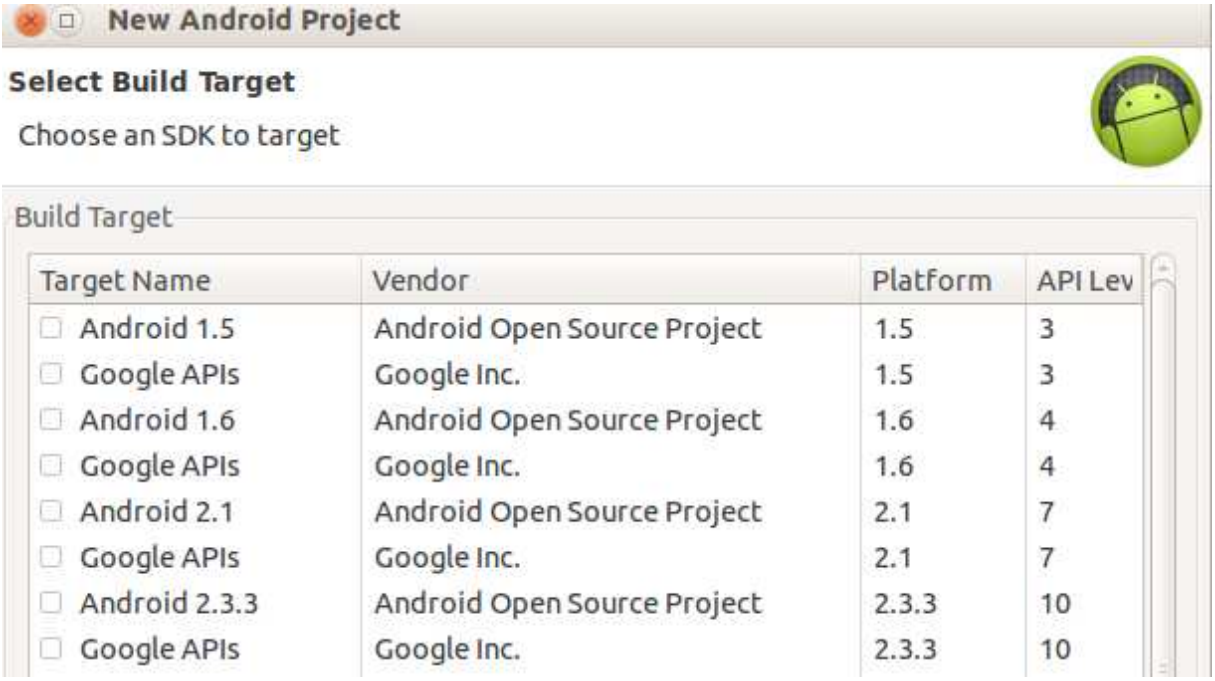


Figure 3.2 : choix du SDK pour le projet.

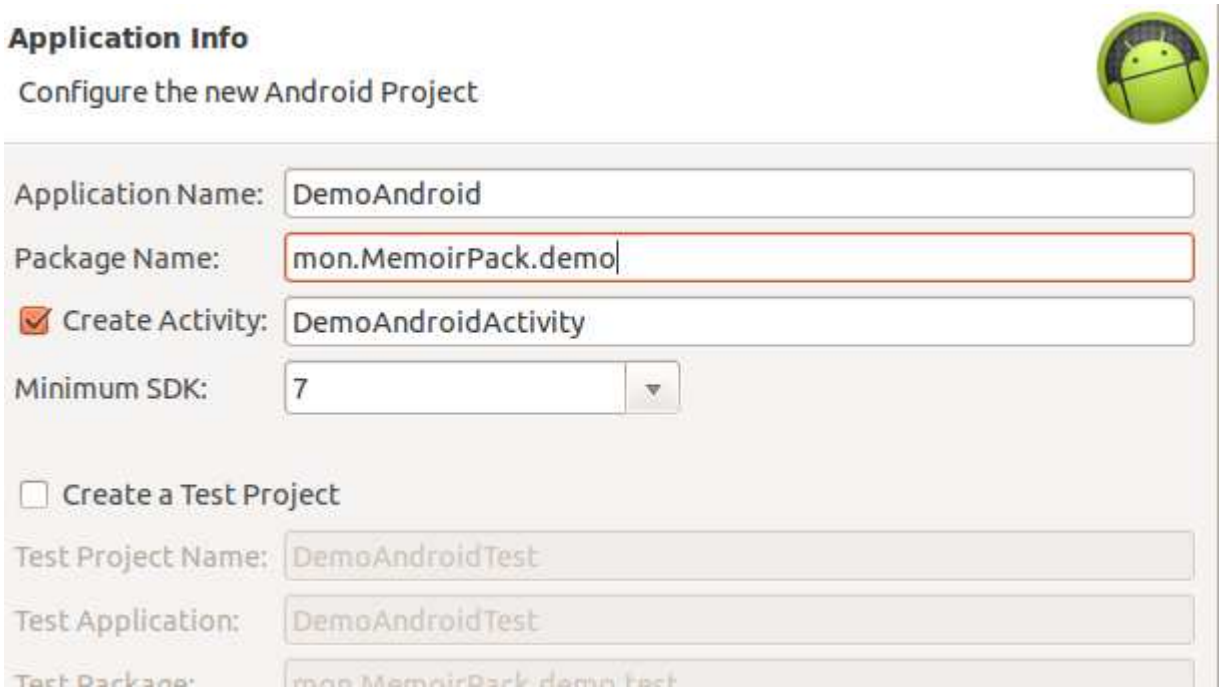


Figure 3.3 : configuration du package Name et de Activity .

- Les champs « Project name » et « Package name » ; il s'agit simplement du nom du projet Eclipse et du package Java racine de l'application.
- Le champ « Activity name » est typique à Android.

Le concept d'Activité est très important dans Android. On les verra un peu plus loin dans ce chapitre. Pour le moment, on dira qu'une Activité pourrait être définie comme le point d'entrée, équivalent de la classe contenant la méthode static main, d'une application Java de base. L'activité est aussi le point d'ancrage, où est défini le contenu visuel de l'écran.

Une fois créé, le projet possède la structure suivante :

Figure 3.4 : Arborescence initiale.

Le système de construction d'un programme Android est organisé sous la forme d'une arborescence de répertoires spécifique à un projet, exactement comme n'importe quel projet Java. Les détails, cependant, sont spécifiques à Android et à sa préparation de l'application qui s'exécutera sur le terminal ou l'émulateur. Voici un rapide tour d'horizon de la structure d'un projet,

a) Contenu de la racine :

La création d'un projet Android (avec la commande android create project ou via un environnement de programmation adapté à Android) place plusieurs éléments dans le répertoire racine du projet :

- AndroidManifest.xml est un fichier XML qui décrit l'application à construire et les composants – activités, services, etc. – fournis par celle-ci.
- bin/ contient l'application compilée.

- gen/ contient le code source produit par les outils de compilation d'Android.
- libs/ contient les fichiers JAR extérieurs nécessaires à l'application.
- src/ contient le code source Java de l'application.
- res/ contient les ressources – icônes, descriptions des éléments de l'interface graphique, etc.- empaquetées avec le code Java compilé.
- assets/ contient les autres fichiers statiques fournis avec l'application pour son déploiement sur le terminal.

b) Contenu du manifeste :

Le point de départ de toute application Android est son fichier manifeste, AndroidManifest.xml, qui se trouve à la racine du projet. C'est dans ce fichier que l'on déclare ce que contiendra l'application – les activités, les services, etc.- On y indique également la façon dont ces composants seront reliés au système Android lui-même en précisant, par exemple, l'activité (ou les activités) qui doivent apparaître dans le menu principal du terminal (ce menu est également appelé "lanceur" ou "launcher"). La création d'une application produit automatiquement un manifeste de base.

- **Structure du fichier de configuration (le manifest) :**

Un fichier de configuration est composé d'une racine et d'une suite de nœuds enfants qui définissent l'application.

```
<manifest
    xmlns:android=http://schemas.android.com/apk/res/android
    package="mon.MemoirPack.demo">
</manifest>
```

Figure 3.5: Structure vide d'un fichier de configuration d'une application.

La racine XML de la configuration est déclarée avec un espace de nom Android (xmlns:android) ainsi qu'un paquetage dont la valeur est celle du paquetage du projet.

Ce fichier est au format XML. Il doit donc toujours être :

- bien formé : c'est-à-dire respecter les règles d'édition d'un fichier XML en termes de nom des balises, de balises ouvrante et fermante, de non-imbrication des balises, etc. ;
- valide : il doit utiliser les éléments prévus par le système avec les valeurs prédéfinies.

4. Contenu d'un programme Android :

« *Le développeur d'une application classique est "le seul maître à bord". Il peut ouvrir la fenêtre principale de son programme, ses fenêtres filles – les boîtes de dialogue, par exemple – comme il le souhaite. De son point de vue, il est seul au monde ; il tire parti des fonctionnalités du système d'exploitation mais ne s'occupe pas des autres programmes susceptibles de s'exécuter en même temps que son programme. S'il doit interagir avec d'autres applications, il passe généralement par une API, comme JDBC (ou les frameworks qui reposent sur lui) pour communiquer avec MySQL ou un autre SGBDR.*

Android utilise les mêmes concepts, mais proposés de façon différente, avec une structure permettant de mieux protéger le fonctionnement des téléphones. »[1]

Une application Android est composée de plusieurs éléments qu'il faut assembler pour obtenir un tout cohérent. Plus une application est complexe, plus le nombre de pièces utilisées sera grand. Voici donc les différents pièces principales du « puzzle » Android :

- activités ;
- services ;
- fournisseurs de contenu ;
- gadgets ;
- objets Intent ;
- récepteurs d'Intents ;
- notifications.

4.1 Composants applicatifs : activité, service, fournisseur de contenu et gadgets [8]

Parmi les éléments précédents, quatre sont des composants applicatifs.

L'activité représente le bloc de base d'une application. Elle correspond à la partie présentation de l'application et fonctionne par le biais de vues qui affichent des interfaces graphiques et répondent aux actions utilisateur.

Le service est un composant qui fonctionne en tâche de fond, de manière invisible.Ses principales utilisations sont la mise à jour de sources de données ainsi que d'activités visibles et le déclenchement de notifications.

Le fournisseur de contenu permet de gérer et de partager des informations. Un même fournisseur permet d'accéder à des données au sein d'une application et entre applications.

Le gadget est un composant graphique qui s'installe sur le bureau Android. Le calendrier qui affiche de l'information ou le lecteur audio qui permet de contrôler la lecture de fichiers sont deux exemples de gadgets que l'on trouve souvent sur un écran d'accueil.

4.2 Éléments d'interaction : intents, récepteurs, notifications

Les éléments suivants permettent l'interaction entre les différents composants du système, entre les applications installées sur l'appareil ou avec l'utilisateur.

L'objet Intent : il permet de diffuser des messages en demandant la réalisation d'une action. L'accès aux autres applications et au système étant restreint par le modèle de sécurité Android, ces objets permettent aux applications de fournir ou demander des services ou des données. La transmission se fait à travers tout le système et peut cibler précisément une activité ou un service.

Récepteur d'Intents : il permet à une application d'être à l'écoute des autres afin de répondre aux objets Intent qui lui sont destinés et qui sont envoyés par d'autres composants applicatifs.

Notification : une notification signale une information à l'utilisateur sans interrompre ses actions en cours.

Filtres (d'Intents) : un objet Intent peut mentionner explicitement un composant cible.Si cette information n'est pas renseignée, Android doit choisir le meilleur composant pour y répondre. Ceci est mené à bien via une comparaison de l'objet Intent avec les filtres des différentes applications cibles. Les filtres ne se manipulent généralement pas via l'API mais sont paramétrables grâce à la balise <intent-filter> du fichier de configuration.

4.3 Permissions

Certaines opérations sont réalisables à condition d'en obtenir la permission. Ces actions sont de plusieurs formes :

- opérations pouvant entraîner un surcoût (connexion, échange de données, envoi de SMS par exemple) ;
- utilisation de données personnelles (accès aux contacts, à un compte Google,) ;
- accès au matériel du téléphone (prise de clichés, écriture sur la carte mémoire...).

Si on veut utiliser les fonctionnalités liées à de telles permissions, on devra déclarer leur utilisation dans le fichier de configuration qui décrit l'application. À l'installation de l' application, l'utilisateur disposera d'un récapitulatif de toutes les permissions demandées pour que l'application fonctionne. Il pourra alors choisir de continuer ou d'interrompre l'installation en connaissance de cause.

5. Activité et vue [5]:

5.1 Définition d'une activité

Analysons un petit peu l'architecture d'une application Android. Si on regarde par exemple un site de téléchargement : on a plusieurs écrans à l'intérieur même de l'application : la liste des applications les plus téléchargées, un système qui permet de rechercher des applications, etc. Ces différents écrans sont des activités. On sait que ce sont différentes activités parce qu'on passe d'un écran à un autre. Une application peut posséder plusieurs activités, mais n'en affiche qu'une à la fois. Cependant, on ne différencie pas des activités que parce qu'elles n'ont pas la même interface graphique.

Une interface graphique peut très bien être dynamique, par exemple à faire grossir puis rapetisser des boutons, l'interface graphique sera donc différente, mais on sera toujours dans la même activité. De plus, un programmeur peut faire plusieurs activités avec la même interface graphique pour une raison ou pour une autre.

On peut considérer une activité comme un support sur lequel se grefferont les éléments de l'interface graphique. Cependant, ce n'est pas son rôle que de créer et de disposer les éléments graphiques, elle n'est que l'échafaudage. En revanche, elle s'occupera de définir les comportements que doivent adopter ces éléments. C'est donc l'activité qui détermine le comportement d'une page, mais pas son esthétique.

De plus, une activité contient des informations sur l'état actuel de l'application : ces informations s'appellent le contexte. Le contexte constitue un lien avec le système Android ainsi que les autres activités de l'application.

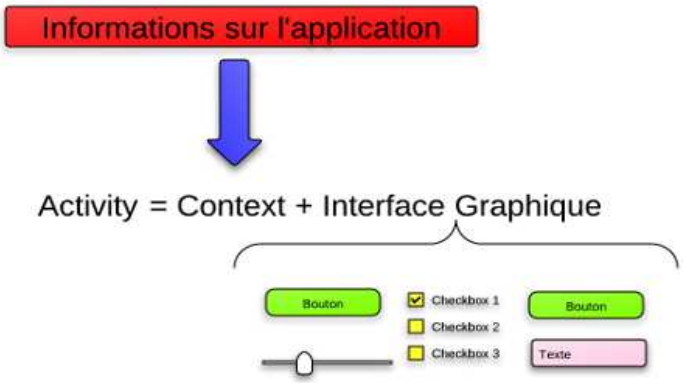


Figure 3.6: information contenu dans une activité.

Exemples :

Quand un utilisateur navigue sur internet avec son téléphone, tout en écoutant la musique qu'émet ce même téléphone. Il se déroule deux « choses » évidentes dans le système :

- une navigation permise par une interface graphique (la barre d'adresse et le contenu de la page web au moins).
- La musique, qui est diffusée en fond sonore, mais sur laquelle l'utilisateur ne peut agir à l'aide d'une interface graphique puisqu'il navigue en même temps.

Dans cette configuration, le navigateur web est une application qui affiche une activité, alors que le lecteur audio est, aussi, une application mais qui n'affiche pas d'activité. En revanche, l'utilisateur peut, s'il le désire, quitter le navigateur pour aller voir l'interface graphique du lecteur audio afin d'arrêter la musique, et là la page qui affiche les contrôles du lecteur est une activité. Le lecteur propose certainement d'autres activités, dont une qui permet à l'utilisateur de choisir la chanson qu'il veut sur la carte SD du téléphone

5.2 États d'une activité :

Quand un utilisateur reçoit un appel, il devient plus important qu'il puisse y répondre que d'écouter la chanson qu'une application diffuse. Pour pouvoir toujours répondre à ce besoin, les développeurs d'Android ont pris deux décisions :

- A tout moment une application peut laisser place à d'autres prioritaires. Si une application utilise trop de ressources système, alors elle empêchera le système de fonctionner correctement et Android pourra décider de l'arrêter.
- Une activité existera dans plusieurs états au cours de sa vie, par exemple un état actif pendant lequel l'utilisateur l'exploite, et un état de pause quand il reçoit un appel.

En fait, quand une application se lance, elle se met tout en haut de ce qu'on appelle la pile d'activité.

Une pile est une structure de données de type « FIFO » (First In, First Out), c'est-à-dire qu'on a accès qu'à un seul élément de la pile : le tout premier élément, appelé aussi sommet. Quand on ajoute un élément à cette pile, il prendra la première place et deviendra le nouveau sommet. Quand on veut supprimer un élément, ce sera le sommet qui sera supprimé et l'objet en seconde place deviendra le nouveau sommet.

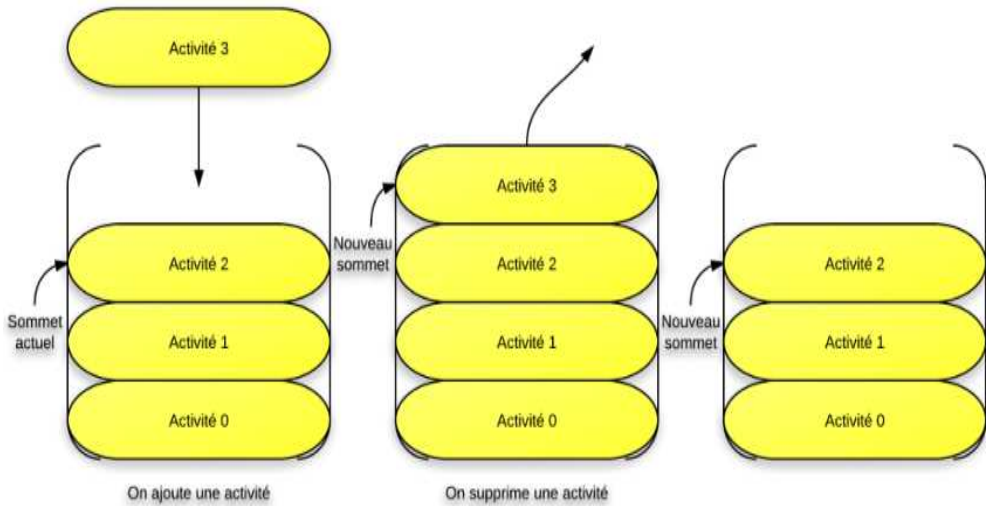


Figure 3.7 : la pile des activités.

L'activité que voit l'utilisateur est celle qui se trouve au-dessus de la pile. C'est pourquoi quand un appel arrive, il se place au sommet de la pile et c'est lui qui s'affiche à la place de notre application, qui n'est plus qu'à la seconde place. Notre activité reviendra uniquement à partir du moment où toutes les activités qui se trouvent au-dessus d'elle s'arrêtent et sortent de la pile.

Une activité peut se trouver dans 3 états qui se différencient surtout par sa visibilité :

État	Visibilité	Description
Active (« active » ou « running »)	L'activité est visible en totalité.	Elle est sur le dessus de la pile, c'est ce que l'utilisateur consulte en ce moment même et il peut l'utiliser dans son intégralité. C'est cette application qui a le *focus*, c'est-à-dire que l'utilisateur agit directement sur l'application.
Suspendue (« paused »)	L'activité est partiellement visible à l'écran. C'est le cas quand vous recevez un SMS et qu'une fenêtre semi-transparente se pose devant votre activité pour afficher le contenu du message et vous permettre d'y répondre par exemple.	Ce n'est pas sur cette activité qu'agit l'utilisateur. L'application n'a plus le focus, c'est l'application sus-jacente qui l'a. Pour que notre application récupère le focus, l'utilisateur devra se débarrasser de l'application qui l'obstrue, puis l'utilisateur pourra à nouveau interagir avec. Si le système a besoin de mémoire, il peut très bien tuer l'application (cette affirmation n'est plus vraie si vous utilisez un SDK avec l'API 11 minimum).
Arrêtée (« stopped »)	L'activité est tout simplement oblitérée par une autre activité, on ne peut plus la voir du tout.	L'application n'a évidemment plus le focus, puisque l'utilisateur ne peut pas la voir, il ne peut pas agir dessus. Le système retient son état pour pouvoir reprendre mais il peut arriver que le système tue votre application pour libérer de la mémoire système.

Figure 3.8 : différents états d'une activité.

5.3 Cycle de vie d'une activité :

Une activité n'a pas de contrôle direct sur son propre état (et par conséquent le programmeur non plus), il s'agit plutôt d'un cycle rythmé par les interactions avec le système et d'autres applications.

Voici un schéma qui présente ce que l'on appelle le cycle de vie d'une activité, c'est-à-dire qu'il indique les étapes que va traverser une activité pendant sa vie, de sa naissance à sa mort.

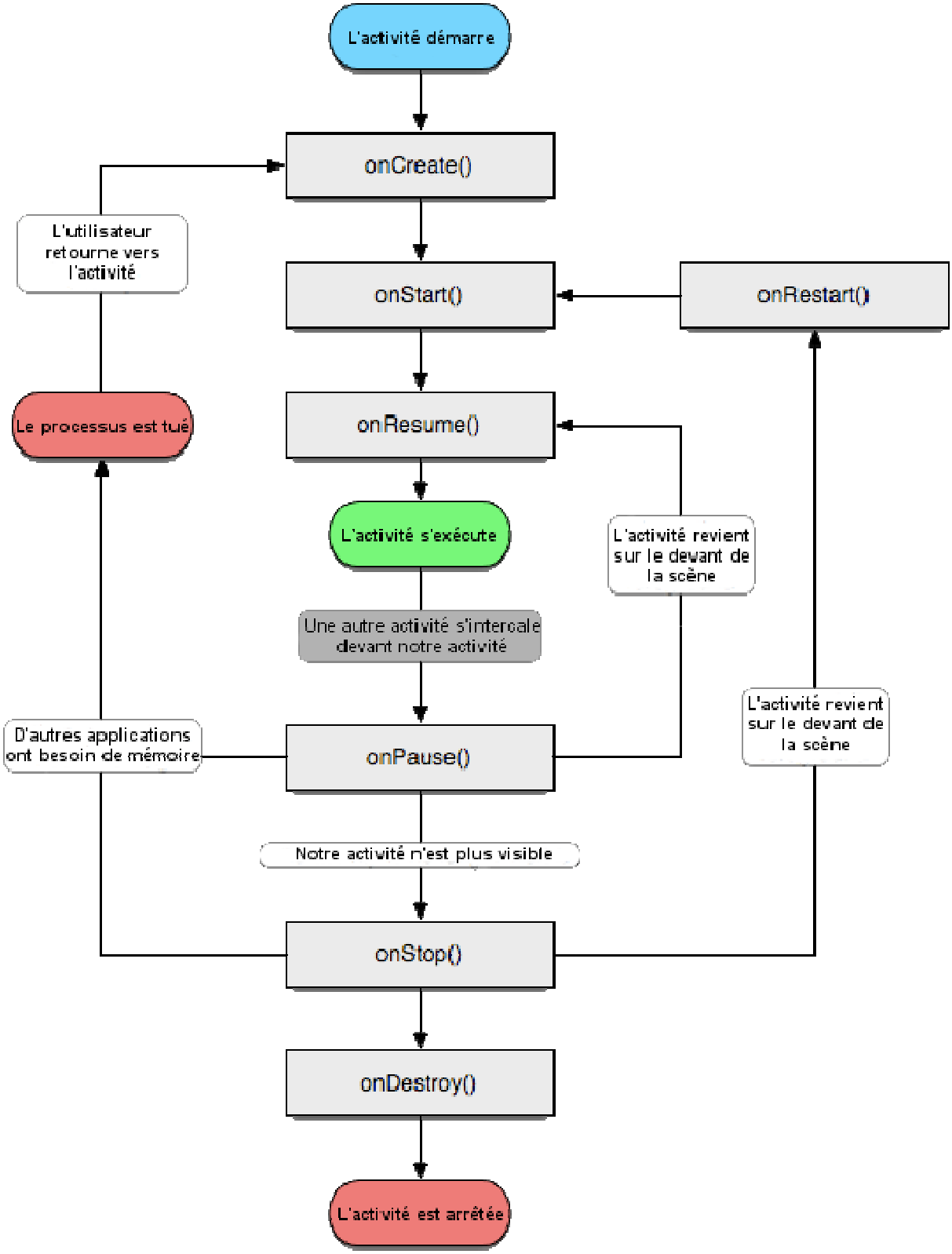


Figure3 .9 : cycle de vie d'une activité.

Le cycle de vie d'une activité est parsemé d'appels aux méthodes relatives à chaque étape. Il informe ainsi le développeur sur la suite des événements et le travail qu'il doit accomplir.

6. Partie graphique :

«Il n'y a pas de bonne application sans bonne interface utilisateur : c'est la condition de son succès auprès des utilisateurs. »[4]

Les interfaces graphiques prennent une place de plus en plus importante dans le choix des applications par les utilisateurs, tant dans l'implémentation de concepts innovants qu'au niveau de l'ergonomie.

Comme, sur bien des plates-formes, les interfaces d'applications Android sont organisées en vues et layouts, avec néanmoins quelques spécificités.

6.1 Le concept d'interface :

Une interface n'est pas une image statique mais un ensemble de composants graphiques, qui peuvent être des boutons, du texte, mais aussi des groupements d'autres composants graphiques, pour lesquels nous pouvons définir des attributs communs (taille, couleur, positionnement, etc.).

Sous Android, on peut décrire une interface utilisateur de deux façons différentes : avec une description déclarative XML ou directement dans le code d'une activité en utilisant les classes adéquates.

La façon la plus simple de réaliser une interface est d'utiliser la méthode déclarative XML via la création d'un fichier XML qui se placera dans le dossier /res/layout du projet.

6.2 Les vues :

Le composant graphique élémentaire de la plate-forme Android est la vue : tous les composants graphiques (boutons, images, cases à cocher, etc.) d'Android héritent de la classe View. Android offre la possibilité de regrouper plusieurs vues dans une structure arborescente à l'aide de la classe ViewGroup. Cette structure peut à son tour regrouper d'autres éléments de la classe ViewGroup et être ainsi constituée de plusieurs niveaux d'arborescence.

L'utilisation et le positionnement des vues dans une activité se feront la plupart du temps en utilisant une mise en page qui sera composée par un ou plusieurs layouts .

6.3 Positionner les vues avec les layouts :

Un layout, ou encore mise en page, est une extension de la classe ViewGroup. Il s'agit en fait d'un conteneur qui aide à positionner les objets, qu'il s'agisse de vues ou d'autres layouts au sein d'une interface.

On peut imbriquer des layouts les uns dans les autres, ce qui nous permettra de créer des mises en forme évoluées. Dans la suite de ce chapitre, nous parlerons ainsi de layout parent et de layout enfant (ou plus généralement d'éléments enfants voire simplement d'enfants), le layout enfant étant inclus dans le layout parent.

On peut utiliser différents types de layout. En fonction du type choisi, les vues et les layouts seront disposés différemment :

- LinearLayout : permet d'aligner de gauche à droite ou de haut en bas les éléments qui y seront incorporés. En modifiant la propriété orientation on peut signaler au layout dans quel sens afficher ses enfants : avec la valeur horizontale, l'affichage sera de gauche à droite alors que la valeur verticale affichera de haut en bas ;
- RelativeLayout : ses enfants sont positionnés les uns par rapport aux autres, le premier enfant servant de référence aux autres ;
- FrameLayout : c'est le plus basique des layout. Chaque enfant est positionné dans le coin en haut à gauche de l'écran et affiché par-dessus les enfants précédents, les cachant en partie ou complètement. Ce layout est principalement utilisé pour l'affichage d'un élément (par exemple, un cadre dans lequel on veut charger des images) ;
- TableLayout : permet de positionner nos vues en lignes et colonnes à l'instar d'un tableau.

6.4 Les composants graphiques [11]:

La brique élémentaire de construction des interfaces graphiques est le ***widget***.

Un layout est une vue spéciale qui peut contenir d'autres vues et qui n'est pas destinée à fournir du contenu à l'utilisateur. Le layout se contente de disposer les vues en un certain motif. Les vues contenues sont les enfants, la vue englobante est le parent, comme en XML. Une vue qui ne peut pas en englober d'autres est appelée un widget (« composant » en français).

Voici un récapitulatif des widgets les plus simples :

a) TextView :

Le TextView est le widget le plus basique qu'il soit : une simple zone de texte. Évidemment la classe définit de nombreux attributs Java et ses équivalents XML (dans sa version déclarative tag XML) pour gouverner finement sa représentation (couleur, police de caractères, dimensions...) et son comportement (conversion automatique des adresses mail, numéros de téléphone et liens web en éléments cliquables...).

L'exemple ci-dessous montre quatre TextView positionnés verticalement :

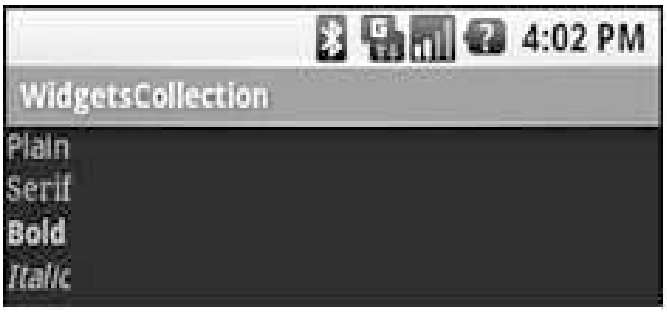


Figure 3.10 : Plusieurs TextView

b) L'EditText :

C'est une extension du TextView. Ce widget est un champ de texte éditable.

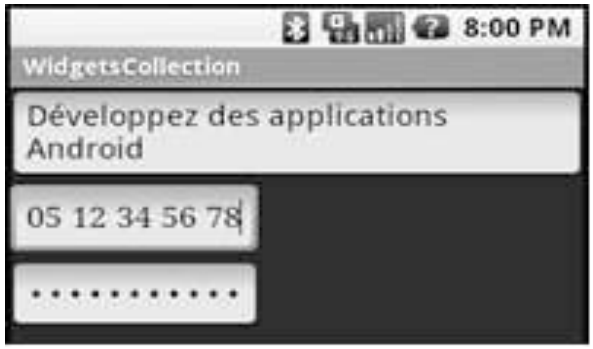


Figure 3.11 : Quelques EditText.

c) La CheckBox :

La classe CheckBox est une case à cocher identique au tag ☐ des formulaires HTML.

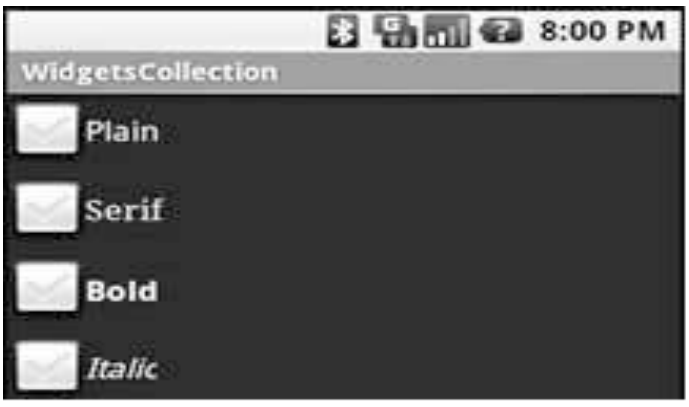


Figure 3.12 : Des CheckBox.

d) RadioGroup :

La classe RadioGroup, accompagnée de la classe RadioButton, sert à afficher des boutons radio. Les boutons radio définissent un ensemble d'options parmi lesquelles l'utilisateur ne pourra en choisir qu'une.

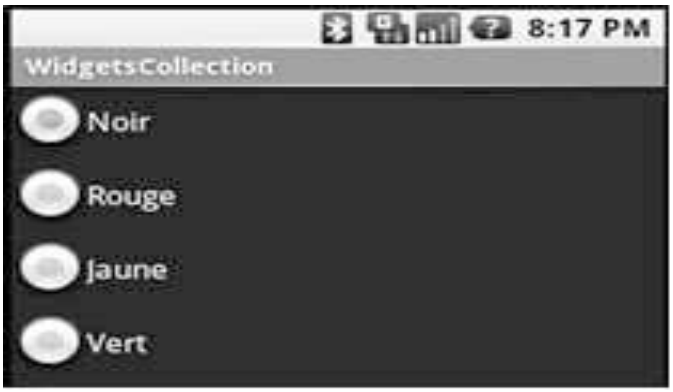


Figure 3.13 : Quatre RadioButton.

Résumé :

pour résumer toutes ces informations , appliquons sur notre programme :

Tout d'abord, commençons par faire une IHM. Il y a deux méthodes :

- En passant par un fichier XML
- Ou en le codant à la main

Nous allons faire cela en passant par le fichier XML car c'est la méthode la plus simple pour modéliser le graphisme. Regardons l'arborescence du projet :

On peut y voir dans le dossier « /res/layout » le fichier main.xml. C'est ce fichier qui va être modifié pour construire notre interface.

Nous allons utiliser :

- des LinearLayout,
- des EditText
- des TextView,

7. Partie fonctionnelle [9]:

Après tout ceci, il faut ajouter des actions aux « EditText » qu'on a mis précédemment. Pour cela, nous allons rajouter du code à la classe qui se trouve dans le dossier « /src/ ».

Revoyons ce code :

```
1) package mon.MemoirPack.demo;


2)  import android.app.Activity;
3)  import android.os.Bundle;


4)   public class DemoAndroidActivity extends Activity {
5)       /** Called when the activity is first created. */
6)        @Override
7)       public void onCreate(Bundle savedInstanceState) {
8)            super.onCreate(savedInstanceState);
9)           setContentView(R.layout.main);
        }
  }
```

Figure 3.14 : Squelette minimal pour créer une première activité.

1 : Là on déclare que notre programme se situe dans le package « mon.MemoirPack.demo »,

2 et 3 : On importe deux classes qui se trouvent dans deux packages différents : les classes Activity et Bundle,

4 : Comme indiqué dans la section précédente, une activité dérive de la classe Activity,

6 : Le petit @Override permet d'indiquer que l'on va redéfinir une méthode qui existait auparavant,

7 : Cette méthode est la première qui est lancée au démarrage d'une application, mais elle est aussi appelée après qu'une application se soit faite tuée par le système en manque de mémoire. C'est à cela que sert l'attribut de type Bundle :

- S'il s'agit du premier lancement de l'application ou d'un démarrage alors qu'elle avait été quittée normalement, il vaut nul.

- Mais s'il s'agit d'un retour à l'application après qu'elle a perdu le focus et a redémarré, alors il pourra contenir un état sauvegardé de l'application qu'on aura pris soin de constituer. Par exemple,

Dans cette méthode, on devra définir ce qui doit être recréé à chaque fois que l'application est lancée après que l'utilisateur en soit sorti (volontairement ou non), donc l'interface graphique et toutes les variables à initialiser par exemple.

8 : L'instruction super signifie qu'on fait appel à une méthode ou un attribut qui appartient à la superclasse de la méthode actuelle, autrement dit la classe juste au-dessus dans la hiérarchie de l'héritage, c'est-à-dire la classe Activity. Ainsi, super.onCreate fait appel au onCreate de la classe Activity, mais pas au onCreate de DemoAndroidActivity. En fait, le onCreate de la classe Activity va procéder à des initialisations à partir de ce fichier. Il gère bien entendu le cas où le Bundle est nul. Cette instruction est obligatoire.

la methode « onCreate » de cette classe, pour le moment ne peut que faire appel au fichier main.xml que l'on a modifié pour y faire l'IHM,avec le setContentView() .

Nous allons donc rajouter le code java qui permettra d'exécuter les commandes shell tapées par l'utilisateur via l' IHM.
Pour reprendre le schéma de fonctionnement des interprètes de commandes, ce code devra lancer les exécutables correspondants.

Une application Java s'exécute dans la machine virtuelle java qui est par défaut un environnement propre indépendant de l'environnement du système hôte.
L'exécution d'une application dans un système hôte ou l'interprétation d'une commande nécessite l'interaction de l'application Java avec ce système .Pour cela, java fournit la classe Runtime().

7.1. La classe Runtime

Cette classe fait partie du package java.lang et permet l'interaction entre un programme Java et l'environnent où il est exécuté (et donc les exécutables présents dans cet environnement).

L'exécution d'une application externe se fait grâce aux méthodes *exec*() de la classe Runtime.

Chaque application Java possède une instance unique de la classe *Runtime* qui lui permet de s'interfacer avec son environnement. L'instance se récupère avec la méthode statique *getRuntime*().

1)

```
Runtime runtime = Runtime.getRuntime();
```

Pour lancer une application externe il nous suffit maintenant d'appeler l'une des six méthodes *exec*() de la classe *Runtime* et dont voici les déclarations :

2)

```
public Process exec(String command);
```

Permet d'exécuter une ligne de commande dans un processus séparé.

3)

```
public Process exec(String[] cmdarray);
```

Permet d'exécuter une commande avec ses arguments dans un processus séparé.

4)

```
public Process exec(String[] cmdarray, String[] envp);
```

Permet d'exécuter une commande avec ses arguments dans un processus séparé en spécifiant des variables d'environnement.

5)

```
public Process exec(String[] cmdarray, String[] envp, File dir);
```

Permet d'exécuter une commande avec ses arguments dans un processus séparé en spécifiant des variables d'environnement et le répertoire de travail.

6)

```
public Process exec(String command, String[] envp);
```

Permet d'exécuter une ligne de commande dans un processus séparé en spécifiant des variables d'environnement.

7)

```
public Process exec(String command, String[] envp, File dir);
```

Permet d'exécuter une ligne de commande dans un processus séparé en spécifiant des variables d'environnement et le répertoire de travail.

7.2. La classe Process

Les différentes méthodes *exec*() renvoie un objet de type Process. Cette classe représente le processus de l'application externe et va nous permettre d'interagir avec lui. La classe *Process*, qui est abstraite, définit les 6 méthodes suivantes :

- la méthode **destroy**() qui permet de tuer le processus de l'application externe,
- la méthode **exitValue**() qui permet de récupérer la valeur de retour du processus de l'application externe,
- la méthode **getErrorStream**() qui permet de récupérer le flux d'erreur du processus de l'application externe,
- la méthode **getInputStream**() qui permet de récupérer le flux de sortie du processus de l'application externe,
- la méthode **getOutputStream**() qui permet de récupérer le flux d'entrée du processus de l'application externe,
- la méthode **waitFor**() qui met le thread courant en attente que le processus de l'application externe se termine.

7.3. Communiquer avec l'application

a) Notion de Flux

Un flux désigne un "canal" qui peut être connecté à différentes sources ou à différentes cibles susceptibles d'engendrer ou de recevoir des données: fichier, périphérique de communication,…etc. En java, il existe deux ensembles de classes permettant de gérer les flux

➢ **Les flux d'entrée**

- InputStream : flux binaires d'entrée;
- InputStreamReader: Transforme un flux binaire d'entrée en un *flux texte*
- BufferedReader : filtre pour ajouter un tampon à un flux texte d'entrée (pouvant contenir une ligne de texte par exemple)

➢ **Les flux de sortie**

- OutputStream : flux binaire de sortie
- OutputStreamWriter : flux texte de sortie
- BufferedWriter : filtre pour ajouter un tampon à un flux texte

b) Récupération des flux

Une application externe exécutée via la classe Process fournit trois flux récupérables par les méthodes getInputStream(), getOutputStream() et getErrorStream(),

- la méthode getInputStream() permet de récupérer le flux de sortie standard de l'application externe.

- la méthode getOutputStream() permet de récupérer le flux d’entrée standard de l'application externe.
- la méthode getErrorStream() permet de récupérer le flux d'erreur de l'application externe.

c) Implémentation :

Voici un bout de code de son implémentation :

```
try {
        Process process= Runtime.getRuntime().exec(edit.getText().toString());

        String tmp;

        // Read input


BufferedReader br = new BufferedReader(new InputStreamReader(process.getInputStream()));

while((tmp = br.readLine())!=null)
        bff+="\n"+tmp;

br.close();


// Error input

br = new BufferedReader(new InputStreamReader(process.getErrorStream()));

while((tmp = br.readLine())!=null)

bff+="\n"+tmp;

br.close();


} catch(Exception e) {

e.printStackTrace();

Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
```

Le fichier Xml correspondant

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
 <EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Commande"
    android:id="@android:id/edit"/>
<TextView
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:id="@android:id/text1"
   android:background="#FF008000"/>
</LinearLayout>
```

En exécutant ce programme on obtient :

Figure 3.15 : interface de l'interprète.

En tapant une commande du shell, « ls » par exemple, on obtient :

Figure 3.16 : Exécution d'une commande.

A LA FIGURE 3.16, On voit qu'il a bien lister le contenu du répertoire courant.

->*Problèmes rencontrés :*

Mais si on tape une autre commande tel que « cd » ou « pwd », on obtient une belle erreur.

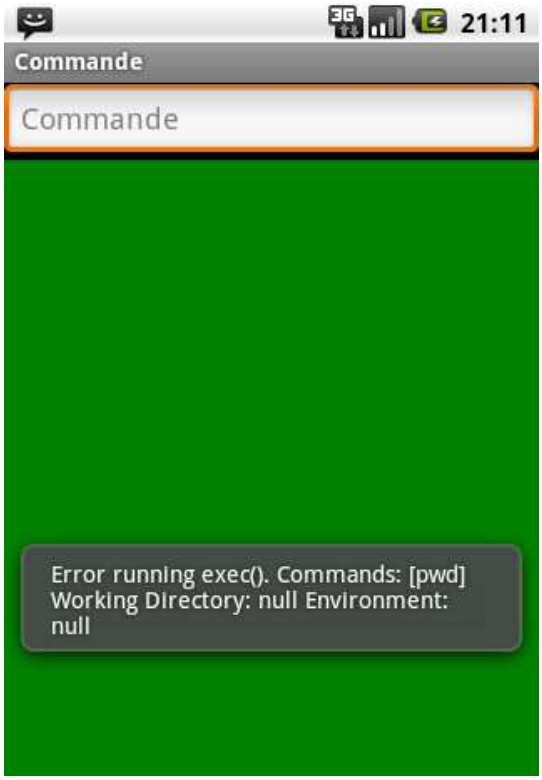


Figure 3.17 : erreur rencontrée.

Cette commande ne marche pas car elle nécessite des permissions particulières. Notre process tourne dans un environnement cloisonnée. Pour que notre application utilise des commandes systèmes, elle doit être logée dans le répertoire système non dans le répertoire application.

Le Terminal emulator quand à lui gère cette commande car il utilise des fonctions qui ne sont pas disponible dans le SDK tel que createSupProcess()(réservé au système).

->*Solution :*

La solution serait de programmer directement les codes des commandes qui seront mis à disposition de l'utilisateur, au lieu de faire appel à la class Runtime()

Exemple :

Pour résoudre le problème du « PWD » on code donc cette commande :

Code :

```
if (commande.startsWith("pwd"))


pwd= (currentDirectory.getAbsolutePath());
```

En ré-éxecutant le code on obtient :

Figure 3.18 : résultat du pwd.

A la figure 3.18 , on voit un « / » qui signifie qu'on est à la racine (/).

8. Conclusion :

Nous avons vu dans ce chapitre comment concevoir une application pour la plate-forme Android .

Nous avons présenté chaque composant de l'application, nous avons vu comment le fichier de configuration de l'application la décrit et comment toutes les pièces interagissent entre elles.

Nous avons présenté l'activité qui est composée de deux volets :

- la logique de l'activité et la gestion du cycle de vie de l'activité qui sont implémentés en Java dans une classe héritant de Activity.
- l'interface utilisateur, qui pourra être définie soit dans le code de l'activité soit de façon plus générale dans un fichier XML placé dans les ressources de l'application.

Donc pour conclure ; une application Android est un assemblage de composants liés grâce à un fichier de configuration.

Dans le prochain et dernier chapitre, nous présenterons le résultat final de notre travail.

1. Introduction :

Après avoir présenté dans le chapitre précédent les différentes étapes de conception d'une application pour android et plus précisément un interprète de commande, nous allons présenter dans cette dernière partie l'environnement de développement, les outils qui ont servi à la réalisation de notre application, et nous terminerons par la présentation de ses fonctionnalités à travers ses différentes interfaces.

« L'implémentation correspond à la mise en œuvre de la solution obtenue après la phase de conception. »[12]

2. Présentation de l'environnent de travail :

Nous, avons à cet effet, travaillé sous le système d'exploitation Ubuntu 12.04 avec l'environnement de développement Eclipse. Et le SDK proposé par Google pour les systèmes Unix/Linux.

a) SDK [9]:

Le kit de développement (Software Development Kit) Android se présente sous la forme d'un fichier zip qu'il suffit de décompresser dans un répertoire quelconque.

Une fois le kit installé, l'utilisateur se retrouve avec l'arborescence suivante :

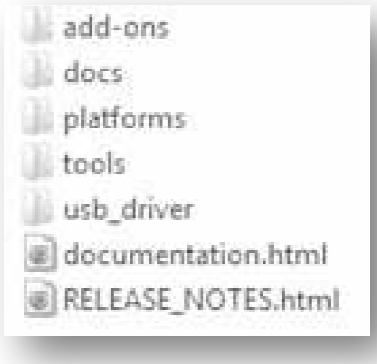


Figure 4.1 : Le contenu du SDK.

- Le répertoire « docs » contient la documentation HTML d'Android. Il s'agit exactement du site http://developer.android.com.
- Le répertoire « samples » (sous-répertoire de docs) regroupe quelques exemples de code dont il est fait référence par endroits dans la documentation.
- Le répertoire « tools » liste les utilitaires Android. La plupart de ces outils sont à utiliser en ligne de commande. Ces commandes sont suffisamment nombreuses pour mériter qu'un paragraphe dans ce même chapitre leur soit réservé.
- « usb_driver » stocke les drivers Windows (pour les systèmes x86 et amd64) nécessaires pour

déboguer les applications directement sur le téléphone et non plus sur l'émulateur. Le chemin de ces drivers est à spécifier à Windows par la boîte de dialogue qui apparaîtra après que l'on ait branché le mobile à l'ordinateur par USB. Sous Linux et Mac, aucun driver n'est à installer.

- Le répertoire « platforms » contient entre autres la bibliothèque Java android.jar.
- Le dossier « add-ons » accueille les API optionnelles de la plateforme comme par exemple Google Maps.
- Enfin, à la racine du SDK, se trouve le fichier documentation.HTML ouvrant sur la documentation

b) Eclipse[11] :

Eclipse est un IDE (Integrated Development Environment) gratuit très prisé des développeurs Java et des entreprises. Eclipse est fourni sous la forme d'une archive au format Zip, laquelle peut être décompressée directement à l'emplacement où on souhaite installer Eclipse. Une fois l'archive décompressée, Eclipse est prêt à être démarré en exécutant eclipse.exe à la racine de son répertoire d'installation.

Utiliser Eclipse nous permettra de gagner en efficacité de développement et de maintenir nos applications beaucoup plus simplement qu'en utilisant un autre IDE non conçu pour les développements Android.

Pour faciliter le développement d'applications Android, Google propose un module, compatible avec cet IDE, nommé ADT

Une fois Eclipse démarré, on doit installer le module ADT. Le téléchargement et l'installation d'ADT sont des processus simples qui s'effectuent directement depuis Eclipse.

L'installation d'ADT devrait avoir enrichi l'interface d'Eclipse avec les éléments suivants :

• le bouton représentant un téléphone permet d'ouvrir le gestionnaire d'AVD (Android Virtual Devices) ;

• celui du milieu avec le signe d'addition permet de créer des projets Android ;

• le bouton orné du signe JU permet de créer un projet de test Android utilisant la bibliothèque de tests Junit ;

• le dernier bouton permet de créer des fichiers de ressources Android (disposition d'interface, valeurs, préférences, animation, menu, etc.).

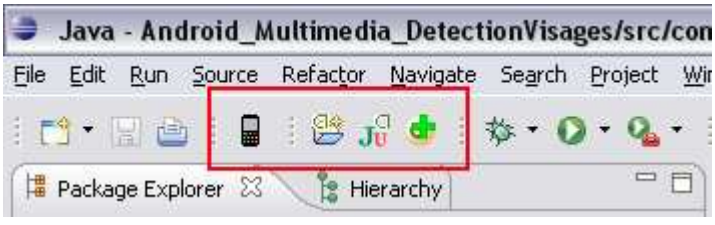


Figure 4.2 : Outils rajoutés par ADT.

c) Configurer un appareil virtuel Android :

Avant de vous lancer dans le développement, il vous faut au moins créer un profil d'appareil afin de pouvoir lancer notre application.

Pour créer un tel profil, on clique sous Eclipse sur le bouton du gestionnaire d'AVD (bouton représenté par une icône en forme de téléphone comme indiqué auparavant) . On Clique ensuite sur le bouton New... :

Figure 4.3 : Le gestionnaire d'appareils virtuels Android.

Une fois dans la fenêtre du gestionnaire d'AVD, On remplit les champs indiqués ci- après dans le tableau :

Name	Nom de l'appareil virtuel.
SDCard	Taille qui sera utilisée pour la simulation d'une carte mémoire de type SD Card. Choisissez une valeur en fonction de la taille des fichiers que vous voudrez stocker sur la carte, sachant que les téléphones actuels ont tendance à être livrés avec plusieurs centaines de méga-octets pour stocker musiques et vidéos. Pour les exemples de ce livre quelques méga-octets suffiront. De façon générale, une taille de « 64MB » (64 Mo) sera amplement suffisante pour la majorité de vos applications.
Target	La plate-forme cible à émuler. Les choix de cette fenêtre dépendront des paquetages que vous aurez sélectionnés. Si vous les avez tous installés, vous aurez la possibilité de choisir des versions de 1.1 à 2.1 et ainsi de construire des images pour émuler différentes versions de la plate-forme et mieux tester vos applications.
Skin	Choisissez le type d'écran simulé par le programme.

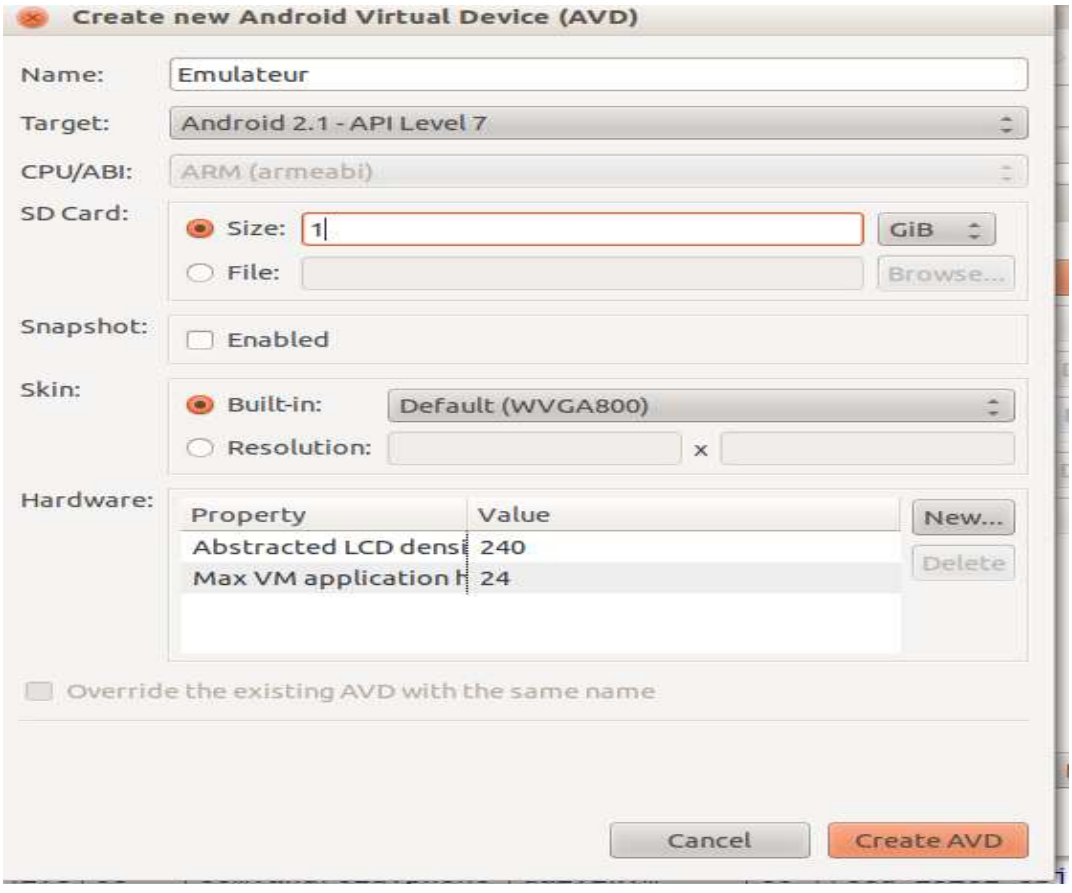


Figure 4.4 : Création de l'AVD.

Pour créer l'AVD, on clique sur le bouton Create AVD.

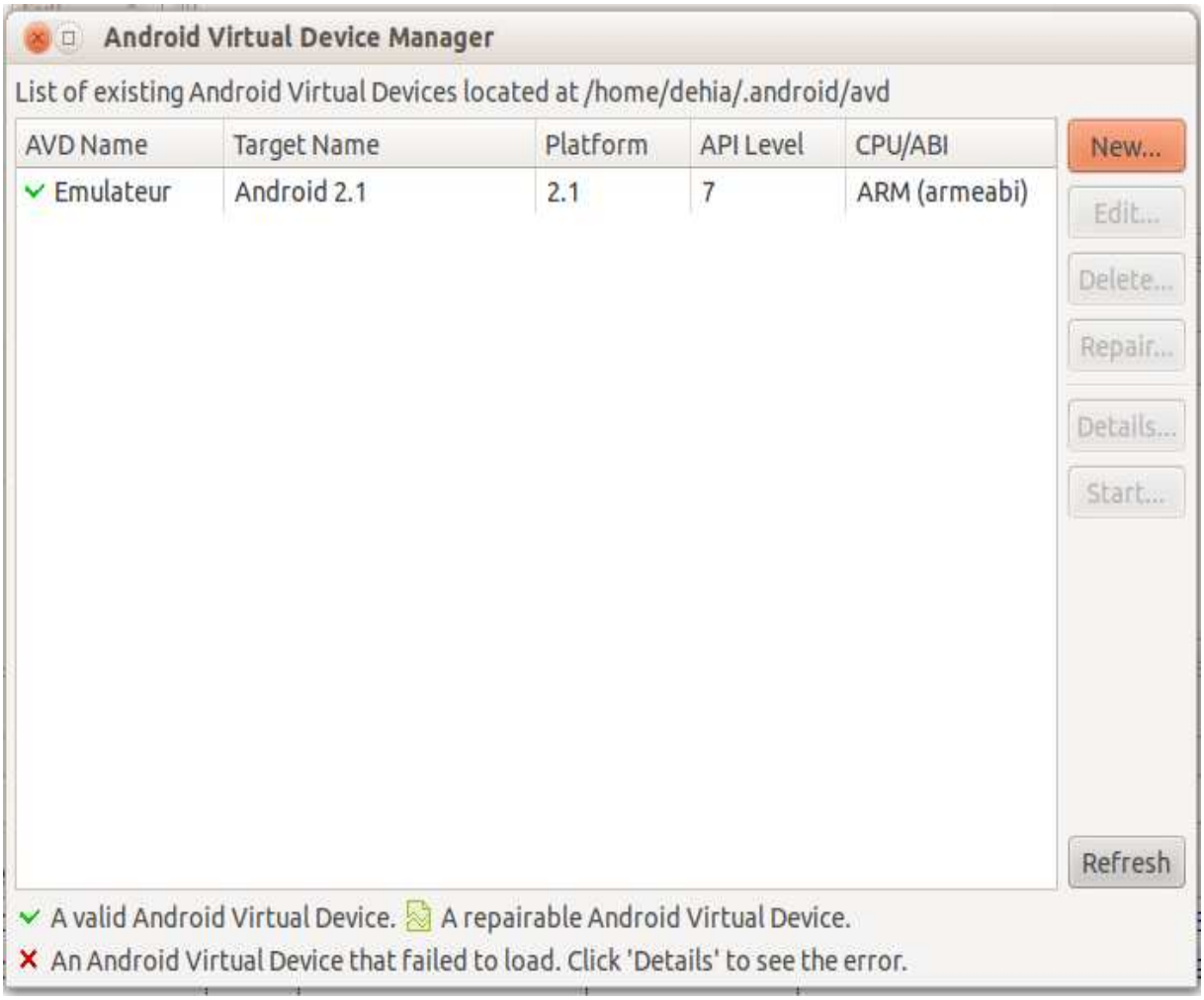


Figure 4.5 : Création d'un appareil virtuel Android 2.1

En appuyant sur Start, on lance l'émulateur :

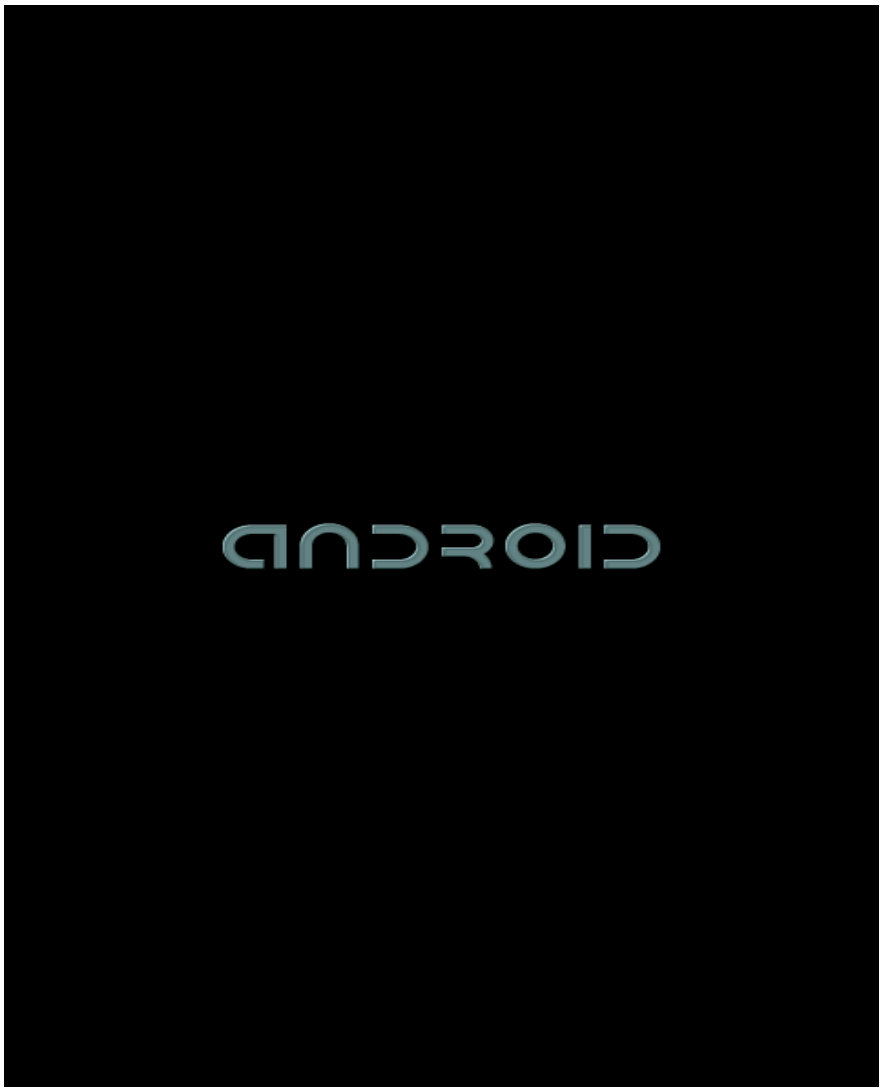


Figure 4.6: lancement de l'émulateur.

Figure 4.7 : l'émulateur 2.1.

3. Vue générale de l'application :

Notre application, comme elle a été présentée précédemment, va exécuter les différentes commandes tapées par l'utilisateur :

Voici à quoi ressemble l'interface de l'application :

Figure 4.8 : l'IHM de l'application.

On tape la commande dans L'EditText, puis on appuie sur le boutant entrée pour avoir le résultat en bas dans le TextView.

L'action appuyer sur entrée efface directement la commande écrite auparavant pour pouvoir en écrire une autre.

On peux quitter l'application soit en utilisant le clavier du téléphone avec la touche retour arrière ou menu ou bien en ligne de commande en tapant « *bye* » .

4. Exemples d'utilisation du shell :

1) la commande « date » : affiche la date et l'heure ;

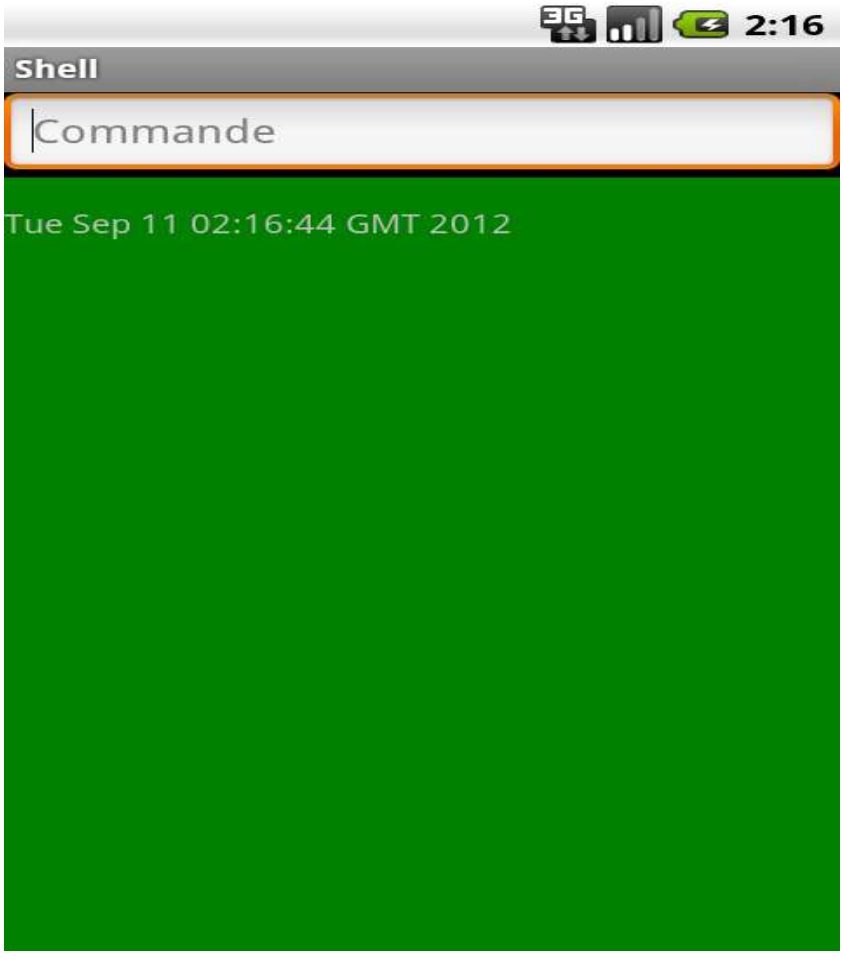


Figure 4.9: résultat obtenu pour « date » .

2) la commande « ls » qui liste le contenu du répertoire courant ;

Figure 4.10 : résultat obtenu pour « ls » .

3) la ligne de commande « cd data »: aller dans le répertoire data ensuite on tape « pwd » pour voir si on est bien dans data ;

Figure 4.11 : résultat obtenu pour « cd data » .

4) Pour revenir en arrière on tape « cd.. » et on fait un « pwd » ;

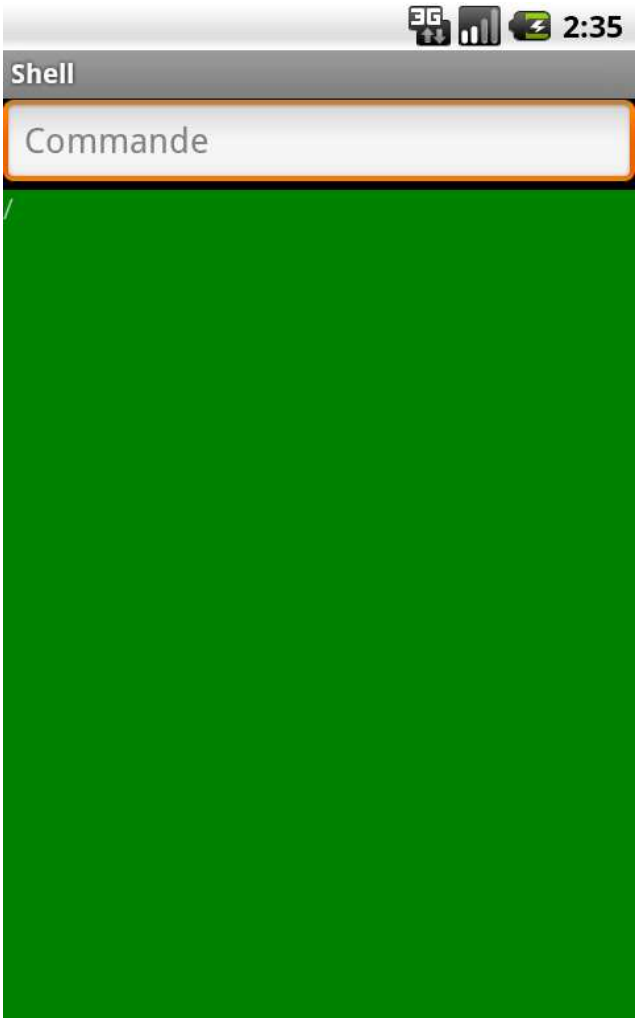


Figure 4.12 : résultat obtenu pour « cd.. ».

5) « mkdir monDossier » : crée un répertoire du nom de « monDossier » dans l'espace mémoire alloué à l'application ;

Figure 4.13 : résultat de obtenu pour « mkdir monDossier » .

5. La liste des commandes supportées par le shell :

Les commandes supportées par l'application:

" bye": quitter l'application

" cat fichier": ouvre un fichier en lecture

" cd rep" :change le reprtoire courant pour un repertoire fils rep

" cd..": revient au repertoire pére

" date": affiche la date et l'heure

" df":indique l'espace occuper par le système de fichier

" help":affiche le menu help

" google":lance le moteur de recherche installer sur le téléphone

" ls": lister le contenu du répertoire courant

" mkdir doss": crée le dossier doss

" mkfl fich": crée un fichier et l'ouvre en écriture

" ps": affiche les processus en exécution

" pwd": affiche le répertoire courant

" rm dossier/fichier": supprime un dossier vide ou un fichier

6. Conclusion :

Nous avons présenté dans ce chapitre les différentes étapes d'exécution de notre application à savoir l'interprète de commande linux.

Après avoir testé l'application avec différentes commandes, et étudier les résultats obtenus, nous sommes arrivés à la conclusion que notre objectif d'implémenter un interprète de commande pour la plateforme Android était atteint.

CONCLUSION GENERALE ET PERSPECTIVES

Nous avons étudié dans ce mémoire la plateforme android, son architecture, ses règles de développement, et aussi les interprètes de commandes et leurs intérêts.

Android est la première plateforme pour appareils mobiles qui soit réellement ouverte et complète avec tous les logiciels nécessaires au fonctionnement d'un téléphone mobile.

Le nom Android est le nom donné à la pile applicative conçue par Google et spécialement optimisée pour les appareils contraints. Les terminaux mobiles étant sévèrement contraints par leur faible puissance de calcul, une autonomie en énergie réduite, des mécanismes de saisie limités (surface d'affichage faible, interfaçage entre l'homme et la machine peu évolué, etc.) ou des capacités mémoires restreintes, Android s'avère parfaitement conçu pour cette utilisation.

Aujourd'hui, Android est principalement utilisé dans les téléphones portables. Ceci dit, la philosophie du système d'exploitation étant de s'abstraire au maximum du matériel sur lequel il tourne (type de processeur, taille d'écran, capteurs, etc.), il se retrouve dans des environnements totalement variés : les téléphones mobiles bien sûr, mais également les tablettes, les netbook, les GPS, les lecteurs multimédias ou mêmes les télévisions, boxes internet et autres appareils mobiles.

Pour programmer des applications Android, on doit au moins connaître les bases de Java. En effet, la programmation Android utilise la syntaxe de ce langage, plus une bibliothèque de classes s'apparentant à un sous-ensemble de la bibliothèque de Java SE (avec des extensions spécifiques).

Nos perspectives pour notre application seraient bien avidement la programmation de plus de commandes, comme les commandes roots, système...etc, en routant le téléphone.

Le développement d’une application est une étape en soi. La phase ultime du développement d’une application Android est sa mise à disposition des

utilisateurs via sa publication sur l'android market. Si le codage reste l’essentiel du travail de la réalisation d’une application Android, cette dernière étape ne doit pas être négligée. La publication d’une application n’est que le commencement du cycle de vie de cette dernière car une fois en ligne, on aura besoin de la mettre à jour régulièrement, soit pour corriger des bugs notifiés par les utilisateurs, soit pour proposer de nouvelles fonctionnalités ou adapter notre création aux différentes versions du SDK.

« L'aventure ne fait que commencer ! »

BIBLIOGRAPHIE

- **[1] :** Éric Jacoboni, L'art du développement Android Titre original : Beginning Android ,publié par Apress, 2009 ,

- **[2] :** florent Garin , Android : développer des application pour le google phone, aux édition Dunod ,2009

- **[3] :** Frédéric Brault , Hackez Google Android : Introduction à la programmation système , aux édition ayrolles, 2009 .

- **[4] :** Damien Guignard ,Julien Chable ,Emmanuel Robles avec la contribution de Nicolas Sorel et Vanessa Conchodon ,Programmation Android :De la conception au déploiement avec le SDK Google Android 2 , aux édition ayrolles ,2009 .

- **[5] :** Développement d'application pour terminaux android :

  www.siteduzero.com

- **[6] :** www.wikipedia.com

- **[7] :** Victor Matos ,Android Environment Emulator, Cleveland State University

- **[8] :** http://developer.android.com/index.html

- **[9] :** http://developer.android.com/guide/developing/tools/emulator.html

- **[10] :** Philippe SOULEMA, Système d'exploitation Unix par Ingénieur Développement société SLIGOS

- **[11] :** www.developpez.com

- [12] : I. Filali & A. Mellouk, « Implémentation d'une méthode de clustering de documents XML », mémoire d'ingénieur, Université Mouloud Mammeri de Tizi-Ouzou, 2008.

ANNEXE

DDMS (Dalvik Debug Monitor Service)

C'est l'un des outils de l'arsenal du développeur Android. C'est une sorte de "couteau suisse" qui nous permet de parcourir les fichiers journaux, de modifier la position GPS fournie par l'émulateur, de simuler la réception d'appels et de SMS et de parcourir le contenu de l'émulateur pour y placer ou en extraire des fichiers.

Pour utiliser DDMS, il faut lancer le programme DDMS qui se trouve dans le répertoire tools/ de notre installation du SDK. Au départ, on ne verra dans la partie gauche qu'une arborescence des émulateurs avec les programmes qu'ils exécutent.

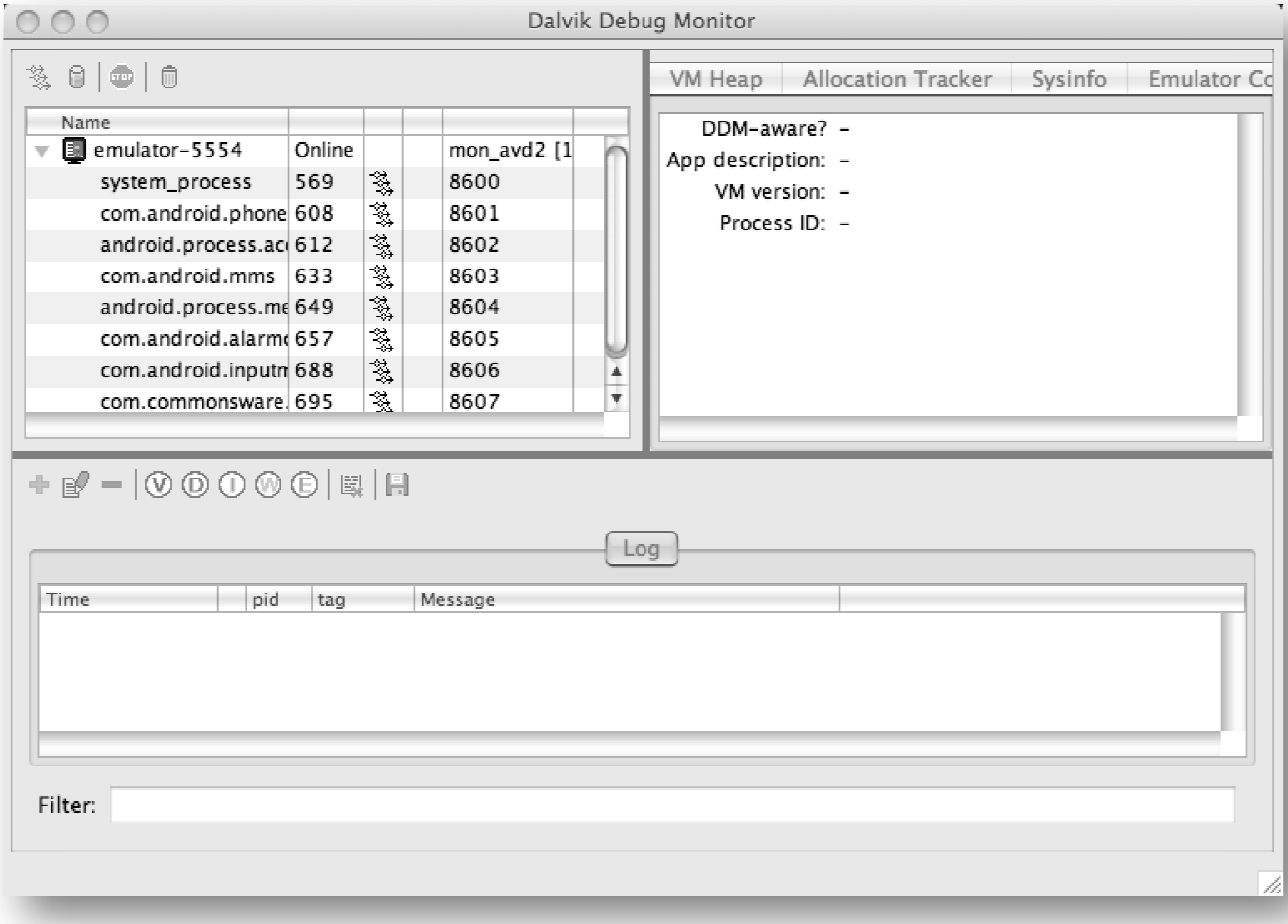


Figure 1 : Vue initiale de DDMS.

Cliquer sur un émulateur permet de parcourir le journal des événements qui apparaît dans la zone du bas et de manipuler l'émulateur via un onglet qui se trouve à droite.

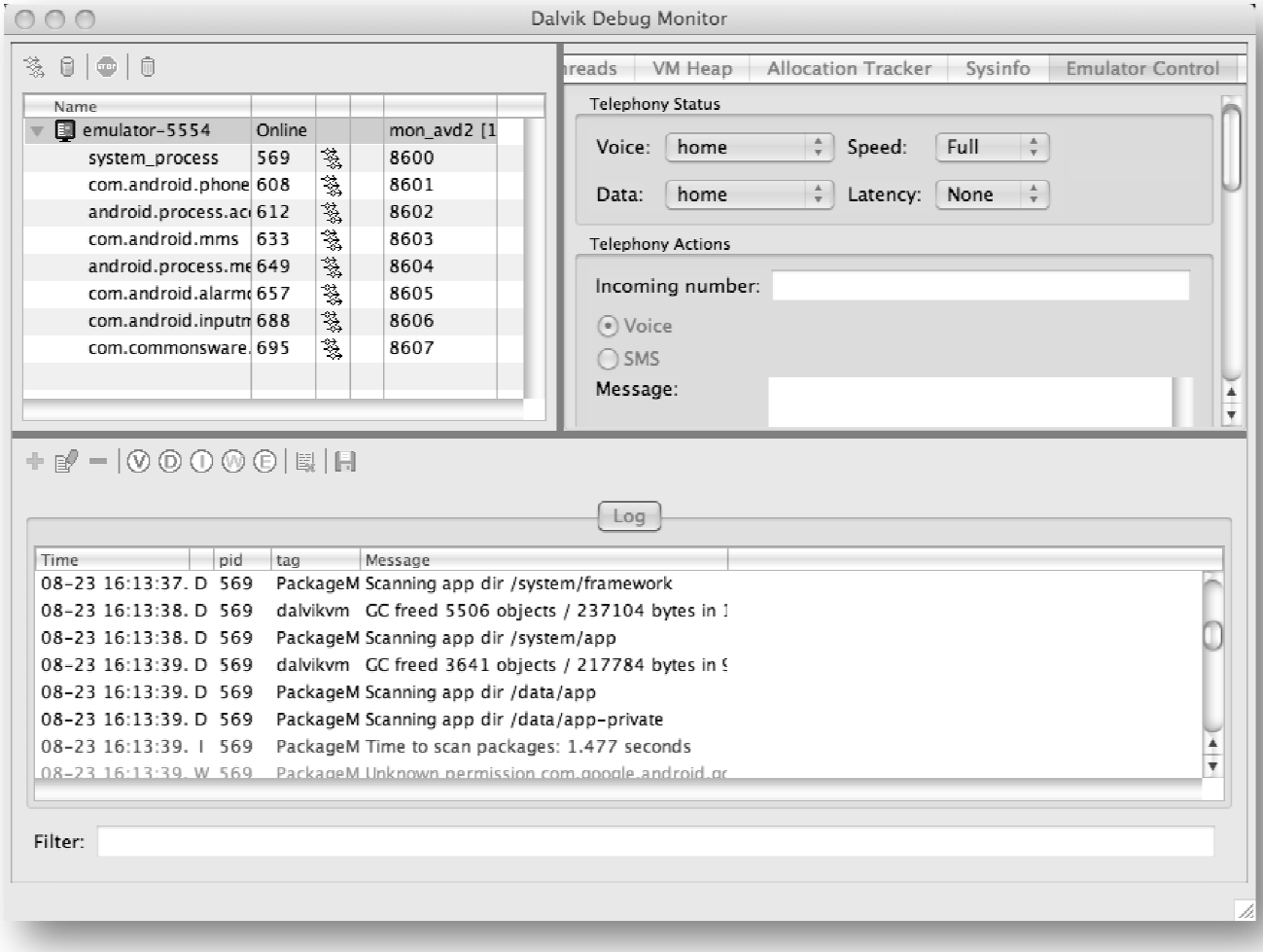


Figure2: Émulateur sélectionné.

➢ *Journaux :*

DDMS permet d'examiner le contenu du journal dans un tableau doté d'une barre de défilement. Il suffit de cliquer sur l'émulateur ou le terminal que l'on veut surveiller pour que le bas de la fenêtre affiche le contenu du journal.

En outre, on peut agir comme suit :

- Filtrer les entrées du journal selon l'un des cinq niveaux représentés par les boutons E à V dans la barre d'outils.

- Créer un filtre personnalisé pour ne voir que les entrées correspondantes. Pour ce faire, on clique sur le bouton + et on remplit le formulaire : le nom qu'on choisira pour ce filtre sera utilisé pour nommer un autre onglet qui apparaîtra à côté du contenu du journal.
- Sauvegarder les entrées du journal dans un fichier texte, afin de pouvoir les réutiliser plus tard.

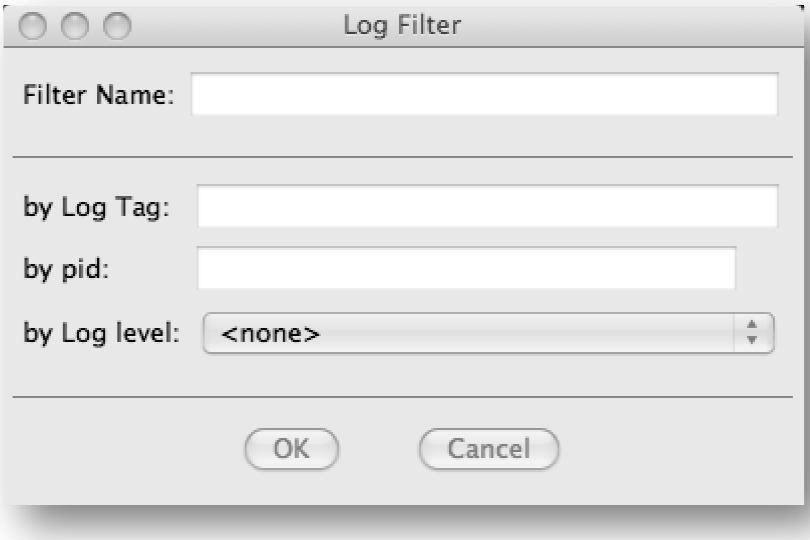


Figure 3 : Filtrage des entrées du journal avec DDMS.

➢ *Stockage et extraction de fichiers :*

DDMS permet l'extraction et le stockage des fichiers sur l'émulateur ou le terminal. Il suffit, pour cela, de sélectionner l'émulateur ou le terminal concerné, puis de choisir l'option Device > File Explorer... à partir du menu principal : une fenêtre de dialogue typique comme celle de la Figure 4 permet alors de parcourir l'arborescence des fichiers.

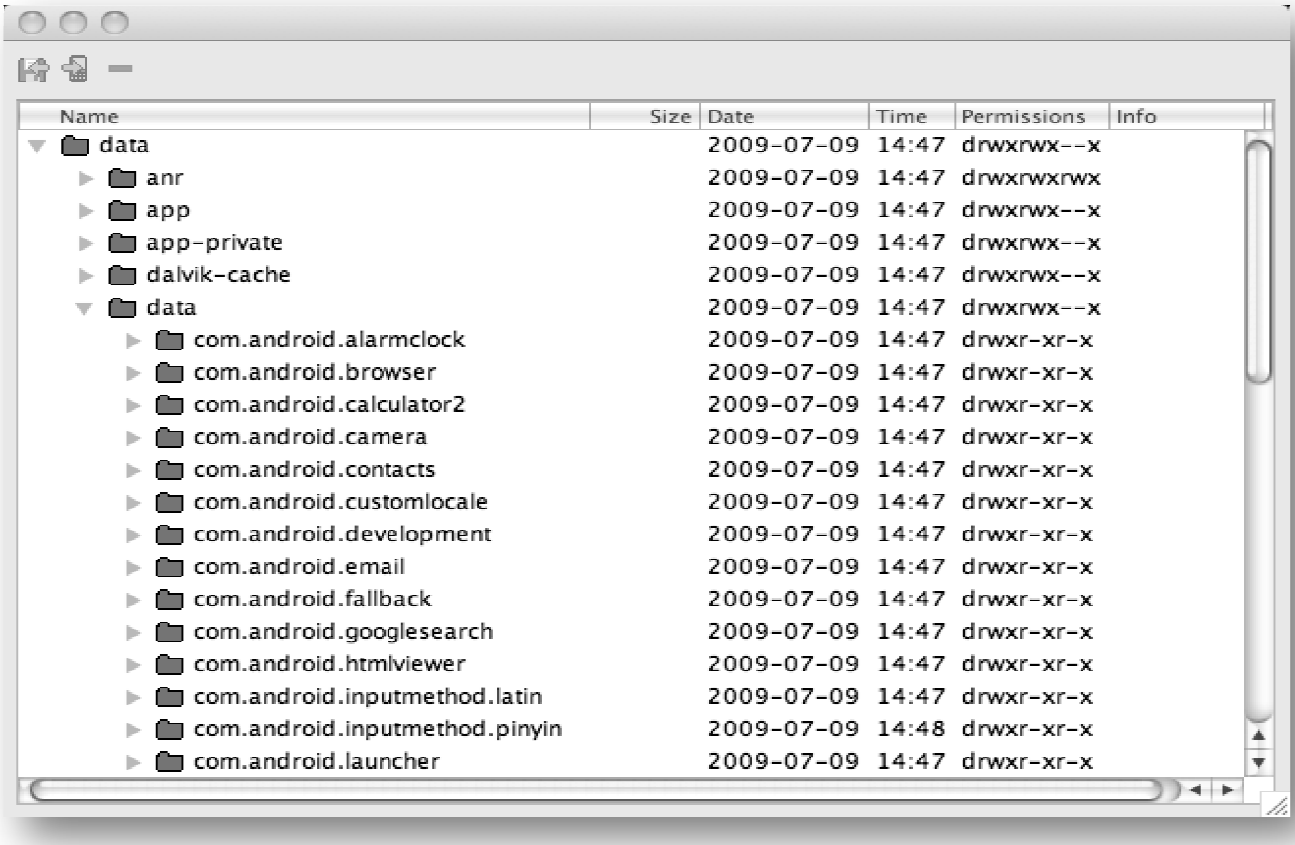


Figure 4 : Explorateur de fichiers de DDMS.

Il faut sélectionnez simplement le fichier concerné et cliquez sur le bouton d'extraction (à gauche) ou de stockage (au milieu) de la barre d'outils pour le transférer vers ou à partir de la machine de développement. Le bouton de suppression (à droite) permet de supprimer le fichier sélectionné.

➢ *Copie d'écran :*

Pour faire une copie d'écran de l'émulateur ou d'un terminal Android, on fait simplement Ctrl+S ou on choisit Device > Screen capture... dans le menu principal. Ceci ouvrira une boîte de dialogue contenant une image de l'écran courant, comme à la Figure 5.

À partir de là, on clique sur "Save" pour sauvegarder l'image au format PNG sur la machine de développement, rafraîchir l'image à partir de l'état courant de l'émulateur ou du terminal, ou cliquer sur "Done" pour fermer la boîte de dialogue.

Figure 5 : une capture d'écran.

➢ *Appels téléphoniques et SMS :*

DDMS sait également simuler la réception d'appels téléphoniques et de SMS via le groupe "Telephony Actions" de l'onglet "Emulator Control" (voir Figure 2).

Figure 6 : Simulation de la réception d'un appel.

Figure 7 : Simulation de la réception d'un SMS dans l'application SMS/MMS.