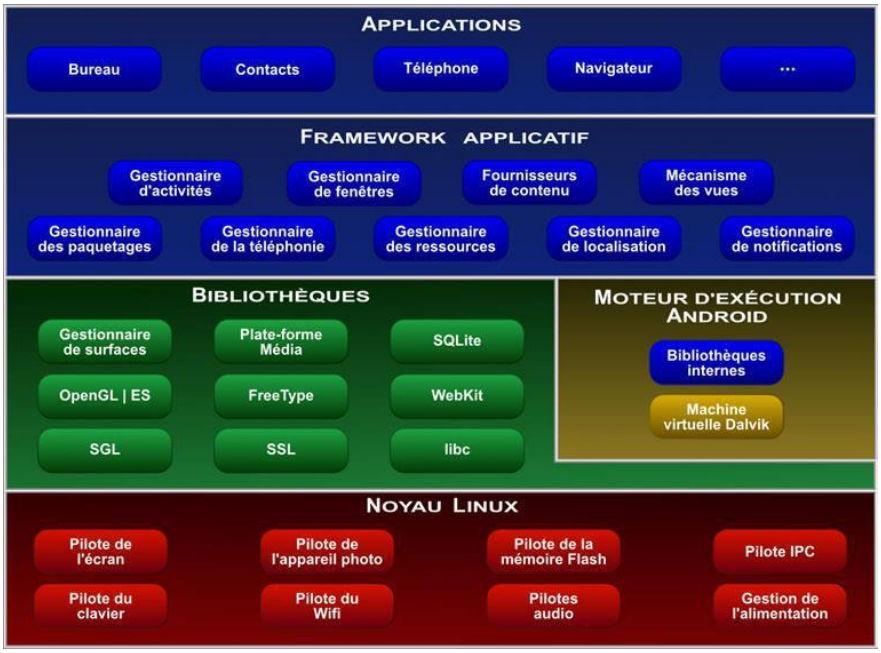
Android

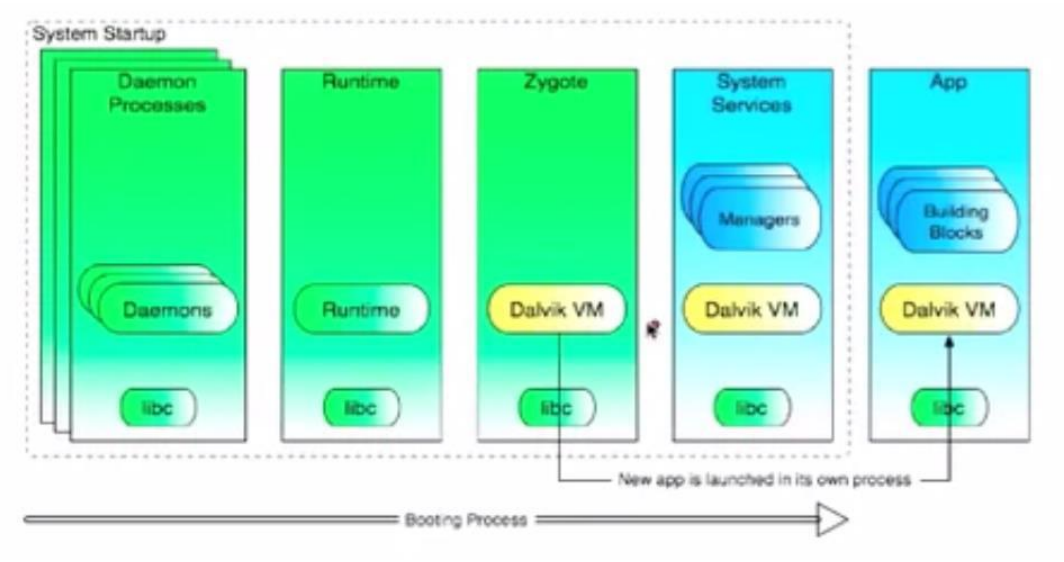
1) Introduction :

Android est une pile de logiciels « open-source » (sous licence apache)incluant un système d'exploitation basé sur le noyau de linux.

Développé par Open Handset Alliance, menée par Google, et d'autres entreprises, vers la fin de l'année 2007

la version 1.0 est crée à la fin de l'année 2008





Platforme Android :

Linux kernel: contient les pilotes du matériel, offre la gestion de la mémoire/processus/l'alimentation/IPC
la première version adoptée est 2.6
Librairies natives: permettant par exemple la gestion des BDD, l'affichage des objets graphiques 2D ou 3D, l'affichage des pages web, le chiffrement, le décodage des formats audio/video …
Machine Virtuelle Dalvik :une JVM optimisée pour les systèmes embarqués
bibliotheques internes (core Lib): les classes java basiques ou specifies à android (java.*, javax.*,android.app.*, android.graphic.* android.view, android.os,..
Applications clés : des classes java (services) réutilisables par la majorité des applications utilisateurs
applications : les programmes manipulés par l'utilisateur final

Middleware :

Moteur WebKit
Libc (Bionic)
SQLite : SGBD
librairies pour lire/ enregistrer les fichiers audio/video
SSL : envoi de messages sécurisés
Open GL :bibliothèque de graphisme 2D et 3D
freeType: gérer l'affichage du texte sur les images BitMap
HAL: des interfaces et des pilotes pour d'autres types de capteurs/matériels

Applications clés :

Activity Manager : Contrôle le cycle de vie des activités et la pile des activités.

Content Providers : permet le partage des données entre activités.

Resource Manager : permet l'accès aux ressources (strings, couleurs, disposition d'interfaces ou layouts).

Notifications Manager – permet l'affichage de messages d'alertes /notifications aux utilisateurs.

View System – un ensemble de vues (UI) utilisables par les applications finales.

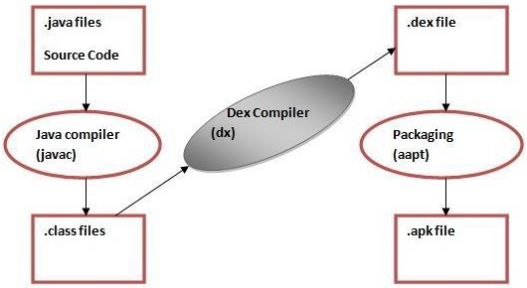
Package Manager – permet aux applications de connaitre des informations sur toutes les applications installées sur l'appareil (favorise la coopération)

Telephony Manager : fournit aux applications des informations sur les services de téléphonie disponibles sur l'appareil (ex: état)

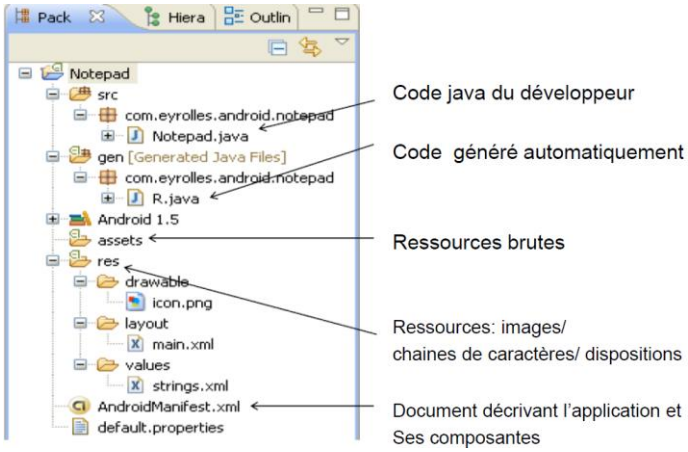
Eléments du SDK :

Plateformes android : plusieurs versions ou api level sous forme de "android.jar"
Doc et exemples / plateforme
Sdk Tools: contient un debogueur DDMS, Virtual device manager, emulateur, mksdcard, traceview, hierarchyview, logcat, sqlite3
Sdk platform-tools: Android Debug Bridge, aapt
Build tools: dx, apksigner, zipalign, jobb qui crypte les fichiers d'extension d'apk.
Image système : pour processeur x86, X86_64,ARM, MIPS
Google API: manipulation des MAP
Code source pour SDK

Processus de génération du APK :



Structure d'un projet android :



Ressources :

Values/

- Chaines de caractère
- Tableaux
- Dimension : la taille des marges/espacement
- Couleurs
- ID…

Drawable/ : images de type gif,png,jpg, ou fichier xml
Layout/ : description de l'interface graphique
menu/ : description des menus
Raw/:fichiers brutes accessible grace Resources.openRawResource()+ ID de la ressource, elles sont empaquetées sans aucun traitement

Xml/ : contient les fichiers XML supplémentaires
anim/ : décrit les propriétés des animations

String :

Mis dans res/values/strings.xml

<resources>
<string name= "b1_text"> ok </string>
<string name="app_name"> essai </string>
<string name="hello"> bonjour </string>
</resources>

Tableau de chaines de caractères :

<resources>
<string-array name="test">
<item>it1</item>
<item>it2</item>
</string-array>
</resources>

Dimensions :

Mis dans res/values/dimens.xml

<resources >
< dimen name = " activity _ horizontal_margin " > 16dp < /dimen >
<dimen name ="activity _ vertical _margin ">16dp < /dimen >
<dimen name="button_height">48dp</dimen>
<dimen name="title_size">32sp</dimen>
< /resources >

Acces en XML : android:layout_width="@dimen/button_height"

Constantes de type couleurs :

Mis dans : res/values/colors.xml
<resources> <color name="rouge_opaque">#f00</color> <color name="rouge_transparent">#80ff0000</color> </resources>
Utilisation : Format #RGB ou #AARRGGBB
Accés en XML : android:textColor="@color/ rouge_transparent
Accés en Java : Resources res = getResources(); int color = res.getColor(R.color.rouge_opaque);

Exemple:drawable :

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent" android:layout_height="match_parent" android:orientation="vertical" >
<ImageView android:layout_height="wrap_content" android:layout_width="wrap_content" android:src="@drawable/image1" />…….
</LinearLayout>

Identifiants :

Mis dans : res/values/ids.xml
<resources> <item type="id" name="button_ok" /> <item type="id" name="textview1" /> </resources>
Acces en XML : <Button android:id="@id/button_ok" android:layout_width="wrap_content" />

Classe R :

Classe auto-génerée
Contient les IDs des ressources du projet
On utilise findViewById ou getResources pour accéder à ces ressources

Exemple:

```
Button b = (Button)findViewById(R.id.b1)
String s= getResources().getString(R.string.hello));
```

```
package com.example.tp2;
public final class R {
    public static final class dimen {
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }

    public static final class id {
        public static final int menu_settings=0x7f080000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050001;
        public static final int menu_settings=0x7f050002;
        public static final int title_activity_main=0x7f050003;
    }
}
```

Principaux composants d'une application mobile :

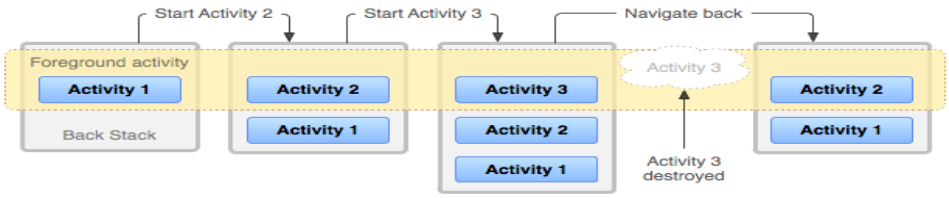
Composant	Description
Activities	Affiche l'IU et gère l'interaction avec l'utilisateur
Services	Exécute un processus en arrière plan associé avec l'application
Broadcast Receivers	Gere la communication entre Android et les applications
Content Providers	Gere les bases de données

Classe Activity :

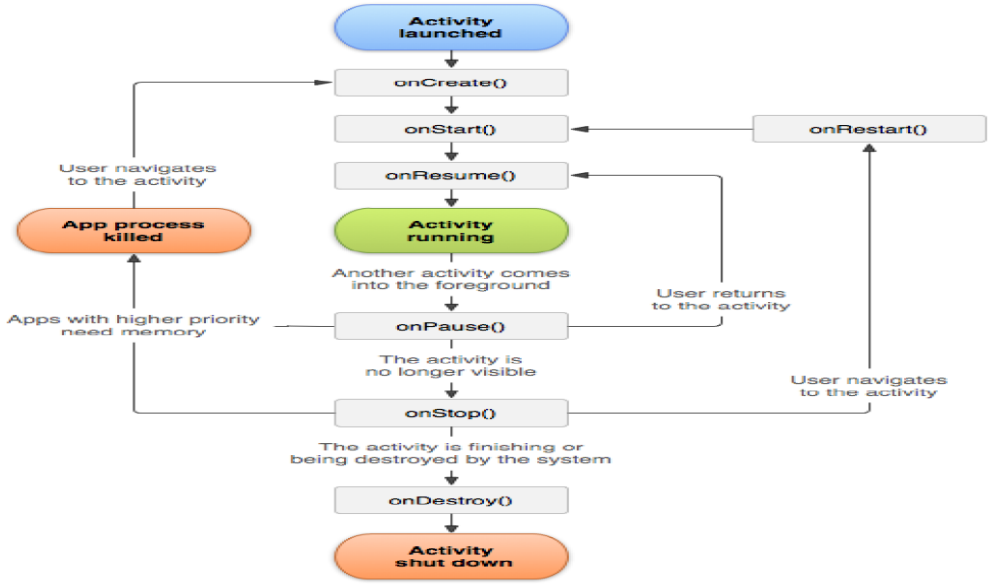
Une activité est une classe affichant un rectangle occupant tout l'écran, et contenant un ensemble de vues (UI Controls) organisées selon une disposition prédéfinie (layout).
Elle est aussi responsable sur la définition des écouteurs.

Notion de tâche :

Collection d'activités appartenant éventuellement à des applications différentes



Classe Activity :



Actions typiques :

OnCreate() :

S'éxecute une seule fois dans la durée de vie de l'activité

Parmi ses responsabilités:

- Initialiser les données des listes (partie modèle)
- Lancer les threads (de type Arrière-plan)
- Instancier quelques variables
- Récupérer l'état précédent de l'activité (bundle)

onStart() : autoriser la visibilité

elle rend l'activité visible, en initialisant l'UI

Eventuellement elle active un broadcastreceiver

onResume() : autoriser l'interaction

Une activité entre en état « resumed» et invoque onResume(), après un évènement lançant l'application en question.

Parmi ses responsabilités:

- initialiser les ressources utilisées (telles que les capteurs GPS, Camera, Broadcastreceivers..)
- Lancer les opérations d'avant plan (animation, lecture de vidéo/audio)
- L'activité sort de l'état « Resumed » après des évènements tels que: la navigation vers une autre activité, mise en veille, lancement d'un processus prioritaire (appel téléphonique)…

OnStop() : (blocage de l'interaction + invisibilité totale)

une activité devient invisible, et entre en état « stopped» suite à des évènements tels que (l'affichage d'une autre activité/ fin ordinaire)

Libération de (presque) toutes les ressources

On doit sauvegarder l'état des données

L'etat des vues est stocké dans un objet bundle grâce à onSaveInstanceState() (après OnResume, et durant ou avant OnStop)

Exécution d'actions garantissant la cohérence de l'app

L' objet « activity » reste en mémoire (pour une éventuelle reprise), sauf en cas de pénurie de mémoire

OnDestroy() : (le processus est tué par android)
Executée Lorsque:

l'activité exécute finish(),
android a besoin de mémoire
Appui du bouton retour
Apres changement d'orientation appel immédiat de oncreate()
La libération de toutes les ressources qui n'étaient pas libérées au niveau de onstop()
En cas de besoin de mémoire, android tue le processus (avec toutes ses activités)

Destruction d'un processus :

% de tuer un processus	Etat du Processus	Etat de l'activité
petit	Foreground (ayant, ou voulant avoir le focus)	Crée Démarrée Poursuivie (Resumed)
moyen	Background (focus perdu)	suspendue (Paused)
élevé	Background (invisible)	arrêtée (Stopped)
	vide	Détruite (Destroyed)

« Manifest » d'une application :

le fichier de configuration de l'application. C'est un fichier indispensable à chaque application qui décrit entre autres :

le point d'entrée de votre application (quel code doit être exécuté au démarrage de l'application) ;
quels composants constituent ce programme ;
les compétences de chaque activité (affichage de page web, Main, envoi d'SMS,….)
les permissions nécessaires à l'exécution du programme (accès à Internet, accès à l'appareil photo...)

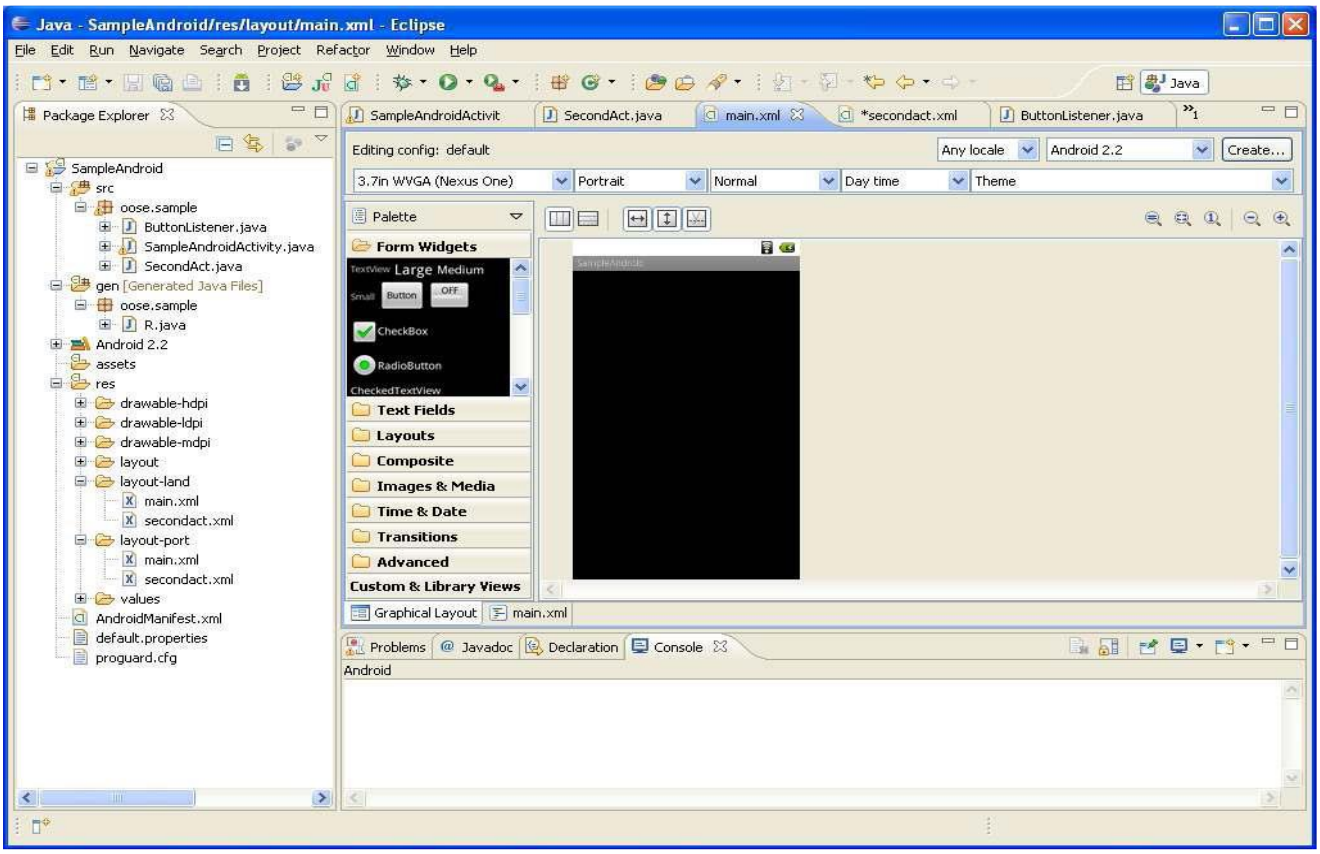
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package=" com. example.tp1"
android:versionCode="1"
android:versionName="1.0">
<uses-sdk
android:minSdkVersion="8"
android:targetSdkVersion="15" />
< uses-permission android:name= "android.permission. CALL_PHONE"/>
< uses-permission android:name= "android.permission.SEND_SMS"/>
< application android:icon = " @ drawable/icon1 "
android :label = " @string/app_name " >
< activity android:name = ".ActivitePrincipale" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<service>...</service>
<receiver>...</receiver>
<provider>...</provider>
</application>
</manifest>

Vues :

Les vues sont les éléments de l'interface graphique que l'utilisateur voit et sur lesquels il pourra agir. Les vues peuvent être des :

vues simples(UI Control)
Botton, TextView, EditText, RatingBar,…
Vues-groupes (viewGroup): vues invisibles qui rassemblent/organisent d'autres vues
Layout (gabarit) :lineaire, relative, grid (grille)
ViewAdapter / Adapter : Gallery, Spinner, ListView
RadioGroup, TimePicker, DatePicker, WebView,…

Layout :



<RelativeLayout
xmlns:android=http://schemas.android.com/apk/res/android
xmlns:tools=http://schemas.android.com/tools
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizotal_margin"
android:paddingRight="@dimen/activity_horizotal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_centerVertical="true"
android:padding="@dimen/padding_medium"
android:text="@string/hello_world"
android : text Size = "100sp"/>

<Button android:text="Go !"
android:id="@+id/b1"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
</Button>

</RelativeLayout>

2) Les intents (intentions) :

C'est une classe qui représente soit:

Une description abstraite d'une opération
une spécification d'une activité cible à lancer
Un évènement qui occure dans android

Exemples:

description d'une opération, ex: ACTION_SENDTO
evenement, ex : Intent.ACTION_BOOT_COMPLETED, Intent.ACTION_BATTERY_LOW

Les Intents permettent d'interagir avec des composants de la même application ou des composants appartenant à d'autres applications.
Ils peuvent démarrer un service ou un broadcast receiver

Types d'intents :

Les Intents explicites définissent explicitement le composant qui doit être appelé par Android

Exemple (ActivityOne.java):

```
Intent i = new Intent(ActivityOne.this, ActivityTwo.class);
i.putExtra("id1", "val1 "); i.putExtra("id2", "val2");
```

id1/id2: representent les clés (de type chaines de caractères, mais les valeurs peuvent être string ou des types primitifs)

Traitement effectué dans l'activité cible

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
    String value1 = extras.getString("id1");
    if (value1 != null) { // faire qcq chose }
}
```

Les Intents implicites précisent l'action qui devrait être exécutée avec éventuellement des arguments.
Ex: pour afficher une page Web on met: Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse("http://....."));startActivity(i);
Tous les navigateurs Web installés doivent être enregistrés avec l'Intent correspondant dans le fichier manifest.xml.

Attributs d'un intent :

Une intention est définie par :

son action,par ex : ACTION_SENDTO, ACTION_DIAL, ACTION_VIEW
ses données (data) :
Uri.parse("geo:33.7749,-1.4192");
Uri.parse("tel:+213550102030");
sa catégorie par ex:
CATEGORY_LAUNCHER, CATEGORY_BROWSABLE
Composant (Component) : nom de l'activité cible
Son Type, ex : "text/plain","text/html","image/png"," image/jpg"
Extra : paires clés/valeurs
it.putExtra(Intent.EXTRA_EMAIL, new String[] {" et1@gmail.com ", "et2@yahoo.com"});
Flags ex :
FLAG_ACTIVITY_CLEAR_TASK
FLAG_ACTIVITY_NO_HISTORY

Recuperation du résultat d'une activité cible :

code de l'activité source (premiere partie) :

```
public void onClick(View view) {
    Intent i = new Intent(ActivityOne.this, ActivityTwo.class); i.putExtra("id1", "val1"); i.putExtra("id2", "val2");
    // initialiser request_code à une valeur entiere arbitraire
    startActivityForResult(i, REQUEST_CODE);
}
```

Code de l'activité cible:

```
public void finish() {
    // Preparer data intent
    Intent data = new Intent(); data.putExtra("id1", "L3");
    data.putExtra("id2", "android"); // RESULT_OK veut dire une fin normale de l'activité cible
    setResult(RESULT_OK, data); super.finish();
}
```

dès que la l'activité cible est terminée, la méthode onActivityResult() de l'activité source est exécutée.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {
        if (data.hasExtra( "id1")) {
            Toast.makeText(this, data.getExtras().getString(" id1"), Toast.LENGTH_SHORT).show();
        }
    }
}
```

Broadcast receivers :

C'est une classe sans UI qui réagit aux evenements qui se produisent dans android
ex:

- fin de démarrage d'android
- Lancement d'un SMS/MMS
- La batterie atteint un certain niveau d'energie
- ….

Elle fonctionne selon le modele publier souscrire

Workflow typique :

Enregistrement du brodacast receiver

- Statique (AndroidManifest.xml) . Ceci est fait après demarrage d'android ou après installation du apk
- ou dynamique (Context.registerReceiver())

Une autre classe crée et diffuse un intent
Android délivre le message (intent) aux composants à son ecoute et lance leurs respectives onreceive() methode + l'intent comme argument
L'évenement est géré dans le code de onReceive()
Apres l'éxecution de onReceive(), Android a le droit d'éliminer le broadcastreceiver.

broadcast receiver (BR) :

Le BR ne contient que la méthode onReceive
Un composant peut définir des permissions sur les BR appelés, de même un BR peut définir des permissions sur les activités appelantes.
il n'a pas le droit de lancer une fenêtre de dialogue/ activité (sa durée de vie est limitée)
Puisque un BR s'éxecute dans le main thread, il est préférable de lancer un service dans onReceive pour les longues tâches
Les BR enregistrés dynamiquement ne réagissent pas aux intents diffusés après l'arrêt de l'application
Un BR peut arrêter la diffusion de l'intent au reste des BR concernés

Exemple complet :

Etape1 : enregistrement statique

```
<application android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name="Receiver1">
        <intent-filter android:priority="100" >
            <action android:name=" CUSTOM_INTENT "> </action>
        </intent-filter>
    </receiver>
</application>
```

Etape2 : création et diffusion de l'intent dans l'activité source (dans la fonction oncreate)

```
……
Intent intent = new Intent();
intent.setAction("CUSTOM_INTENT");
sendBroadcast(intent);
……
```

Etape 3:Reaction du BroadcastRceiver

```
public class Receiver1 extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Intent intercepté.", Toast.LENGTH_LONG).show();
    }
}
```

Caracteristique des intents diffusés :

Portée Locale/globale

Normal/ordered

Sticky/non Sticky

Sans / avec permission

3) Les Threads :

Tache :

- Ensemble d'activités qui appartiennent à la meme application/processus ou non
- Secnario d'execution géré par l'application "activity manager"

Processus :

- Le lancement d'un composant d'une application cree un nouveau processus (si cette app n'a pas déjà de composant(s) en cours d'execution)
- Par defaut une app = un seul processus avec un seul thread (UI thread ou Main Thread ou looper thread)
- Le developpeur peut assoicier des processus separés à des composants de la meme app (android:process)
- La verification de l'appartenance des composants par rapport aux processus est assurée par :
  (int pid = android.os.Process.myPid();)

Thread (d'unprocessus) :

- Flux d'instructions qui s'execute en parallèle avec d'autres flux et qui partage avec eux les variables statiques et le tas (heap) mais qui a sa propre pile et son propre compteur ordinal.

Ui thread vs background thread :

- l'Ui-thread reagit aux evenements de l'utilisateur (interaction avec les widgets)
- contient les methodes du cycle de vie des activités
- Si un widget ne reagit pas au bout de 5 sec, alors android affiche un Boite de Dialogue proposant l'arret de cette app

- Il est toujours conseillé de ne pas bloquer le UI thread avec une longue tâche qui s'éxecutant dans le même thread (ex dans onCreate)

Exemple de tâches consommatrices de temps : téléchargement, requête BDD, chargement d'images,….

"UI toolkit " n'est pas " thread-safe ".

Obtention de l'ID d'un thread est faite par :

- android.os.Process.myTid(); ou Thread.currentThread().getId();

Priorité des processus :

1.Processus en avant plan (activité en interaction utilisateur, service attaché à cette activité, **BroadCastReceiver** exécutant **onReceive()**)

2. Processus visible: il n'interagit pas avec l'utilisateur mais peut influer sur ce que l'on voit à l'écran (activité ayant affiché une boite de dialogue (**onPause()** a été appelée), service lié à ces activités "visibles").

3. Processus de service

4. Processus tâche de fond (activité non visible (**onStop()** a été appelée))

5. Processus vide (ne comporte plus de composants actifs, gardé pour des raisons de *cache*)

Durée de vie d'un thread :

Si la tâche concurrente (thread) est lancée :

- par l'activité principale: sa vie durera le temps de l'activité
- par un service: la tâche survivra à l'activité principale

Classes /fonctions impliquées :

Thread (implementant l'interface Runnable)

- **New thread (new runnable() {…})**
- **Start()**
- **Sleep(…)**
- **Notify()**

Runnable

- **Run ()**

Activity.runOnUiThread(new runnable() {…}))
View.post(new runnable() {…})
AsychTask

- **onPreExecute() UI thread**
- **Results doInBackground(param):BG thread**
- **Publishprogress(progress…) B.G thread → onProgressUpdate(progress…) UI thread**
- **onPostExecute(result) UI thread**

Classes impliquées dans les threads :

Handler :

- sendMessage(msg)
- sendMessageAtFrontOfQueue(msg)
- sendMessageAtTime(msg)
- sendMessageDelayed(msg)
- handleMessage(Message msg)
- Post(new runnable() {….})
- PostAtTime(runnable r, long timeMillis)
- PostDelayed(runnable r, long timeMillis)
- ObtainMessage() // plusieurs signatures

1ere approche du multithreading :

```
public void onClick(View v) {
    Thread th = new Thread(new Runnable() {
        public void run() {
            …….
            runOnUiThread(new Runnable() {public void run() {
                    ImageView image = (ImageView) findViewById(R.id.imageView1);
                    image.setImageResource(R.drawable.ic_action_call);
                }
                ……..
            });
            th.start();
        }
    }
}
```

2eme approche du multithreading :

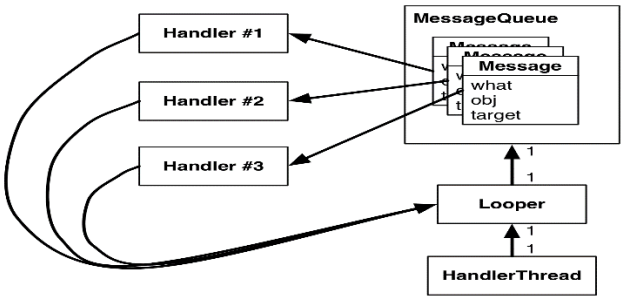
la classe Handler permet la communication (ou délégation) des messages/taches à exécuter(runnable) entre les threads (pas forcement le UI thread et BG thread)

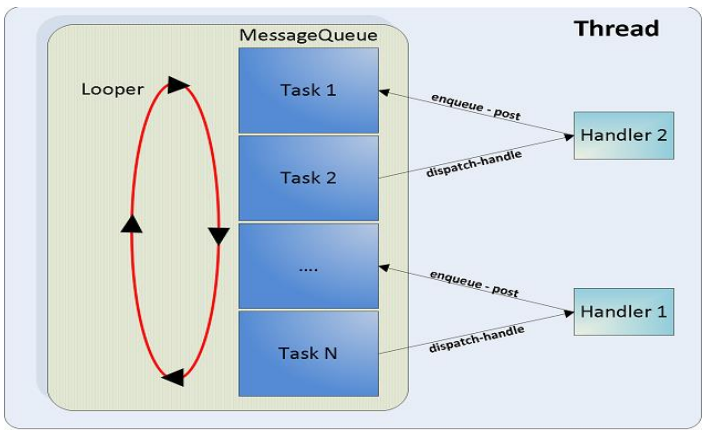
un handler est associé à un seul thread

Runnable: c'est une interface contenant le code à exécuter par le thread traiteur(le thread émetteur connait les étapes de traitement)

Message :classe contenant plusieurs attributs: code-message, donnée de type object, 02 arguments entiers (le thread émetteur ne connait pas l'implementation de l'operation faite par le thread traiteur)

Chaque thread contient un file d'attente « message queue » contenant des messages ou des runnables et un objet looper qui permet de gérer cette file





Tâches du looper :

Le looper route les elements de la file d'attente aux classes concernées
Le traitement se fait selon la politique fifo
Dans le cas d'un message il invoque la fonction du handlemessage() du handler
Dans le cas d'un runnable il invoque simplement la fonction run() de cette classe

Voir le TP4

3eme approche du multi-threading :

AsynTask attend trois paramètres :

Le type de l'information qui est nécessaire au traitement (ex:URL,String ,…)

Le type de l'information qui est passé à sa tâche pour indiquer sa progression (ex:Integer,Void,…)

Le type de l'information passé au code lorsque la tâche est finie (ex: Long,Integer,…).

Exemple :

```
public class MyAsyncTask extends AsyncTask<String, Void, Integer> {…. }
…….
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    MyAsyncTask myTask = new MyAsyncTask(this);
    String param = ″ ‵;
    myTask.execute(Param);
}
```

AsynchTask vs Handler :

L'utilisation de AsynTask est plus facile que celle des Handlers

AsynchTask est utile pour faire une communication entre un thread worker et un UI thread

Avec AsynchTask plusieurs tâches s'executent dans le même thread sauf si on utilise l' option ThtreadPoolExecutor

Avec AsynchTask on doit respecter le workflow par défaut (cad la tâche de fond est executée une seule fois et les points de communication avec le UI thread sont limités)

Avec Handler, on peut assurer la communication entre 02 threads workers ou un thread worker et un autre Main.

Les scenarios de communication avec les threads workers sont plus flexibles (échanges illimités)utile pour les tâches répétitives

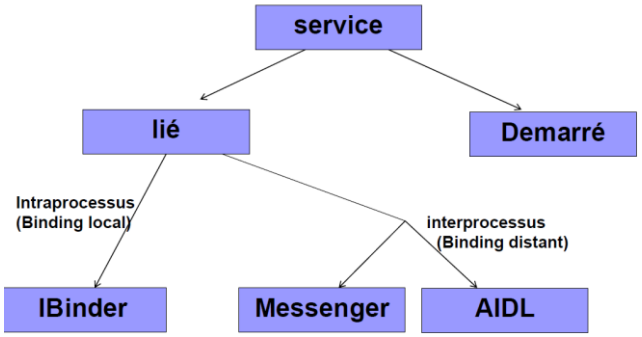
4) Les Services :

Composant d'une application exécutant du code sans UI (généralement les longues tâches)

- Exemples(telechargement, lecture de fichiers audio, requêtes BDD…)

Le service continue l'execution même après suspension/destruction du composant appelant.
Peut s'exécuter durant une période de temps indéfinie
Par defaut un service s'execute dans le main thread du processus hote
Un service est tué par le système si :

- pénurie de ressources (i.e pas assez de mémoire)
- La susceptibilité d'être tué dépend de sa priorité
- La priorité d'un service dépend (par défaut) de l'application qui l'utilise



Implementation d'un service :

Hériter de la classe Service / IntentService

Ajouter le composant «service» dans le fichier manifest :

```
<manifest ... >
    ...
    <application ... >
        <service android:name=".ExampleService" />
        ...
    </application>
</manifest>
```

Types de services :

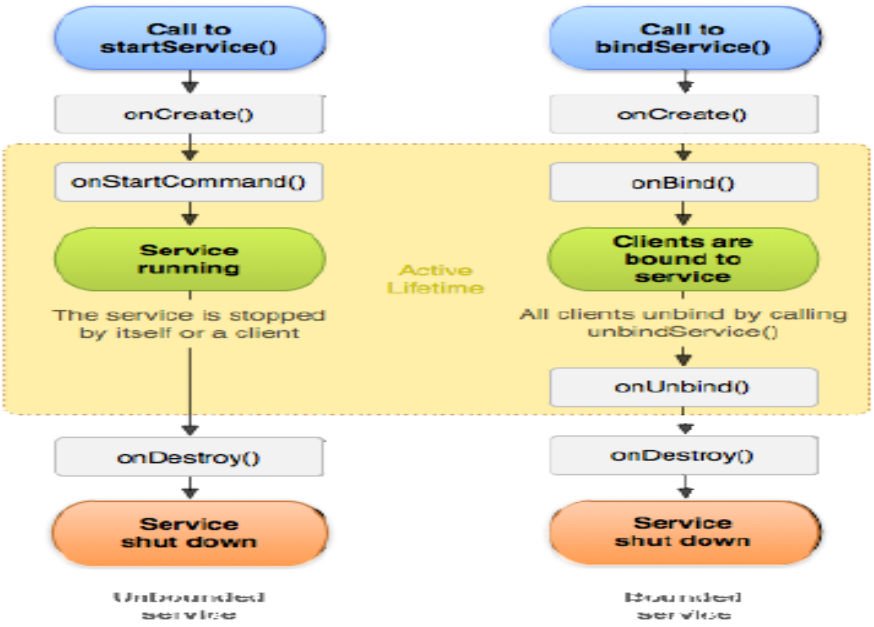
Service démarré (started service)

- Services simples (sans interaction avec le composant appelant), et ne retournant pas de résultât (en general).
- S'executant de manière indéfinie
- Lancés par un composant (ex: une activité) grâce à startService()
- Appel de onStartCommand(Intent, flags, startId) pour l'execution de la tache

Service lié (bound service)

- Le composant client appelle bindService() qui lancera onBind()/onRebind()
- Il permet l'échange de requêtes/réponses avec une application cliente (interaction complexe)
- Il permet de réaliser les RPC (au sein du même système android)
- Lorsque tous les clients deconnectent (unbind), le systeme detruit le service. Pas besoin d'arret explicite du service.
- Deux classes candidates: messanger/ Interface AIDL

Cycle de vie d'un service :



Le lancement d'un service se fait avec un intent explicite.

Service demarré :

- On demande une réaction à un Intent en appelant Context.startService(Intent)
- La communication est uni-directionnelle (one way service)
- Le service est créé si nécessaire
- Le service peut être stoppé, par lui-même stopSelf, par celui qui l'a lancé stopService(Intent)
- Deux classes possibles: Service/ IntentService
- la methode onBind() retournera null.

Exemple :

```
Intent intent = new Intent (this, hreadedDownloadService.class));
intent.putExtra("URL", imageUrl); startService(intent);
```

```
public class DownloadService extends Service {
    public int onStartCommand (Intent intent, int flags, int startId) { ... }
}
```

Service demarré avec intentServ :

- IntentService : classe spécialisée pour les traitements long
- Il suffit d'implementer
- un constructeur de la classe IntentService
- pareil pour onHandleIntent().
- Le service est arrété automatiquement par android lorsque onHandleIntent() s'acheve.

IntentService :

```
public class HelloIntentService extends IntentService {
    public HelloIntentService() {
        super("HelloIntentService");
    }
    /* le traitement des requetes est sequentiel dans le thread worker */
    protected void onHandleIntent(Intent intent) { // faire le travail ici (ex: telechargement). }

    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
        return super.onStartCommand(intent,flags,startId);
    }
}
```

Service demarré avec la classe service :

- Possibilité de créer des threads gérant simultanément plusieurs requêtes (intents)
- La fonction onStartCommand() retourne 03 types de constantes
- START_NOT_STICKY
- START_STICKY
- START_REDELIVER_INTENT
- Un service continue l'execution meme apres l'achevement de onStartCommand()
- Pour arreter un service, il faut appeler explicitement stopSelf() ou stopService(),

Voir le TP5

Service lié :

Lorsqu'un composant (client) appelle Context.bindService(intent,serviceConnection,flags) pour se lier (bind) à un service

flags:

- BIND_AUTO_CREATE (démarre le service si nécessaire)
- BIND_ADJUST_WITH_ACTIVITY(monte la priorité au même niveau que l'activité)
- BIND_WAIVE_PRIORITY (pas de changement de priorité) => La méthode onBind()/onRebind() est appelée sur le service

Messenger :

- Gère une queue de Messages inter-processus
- Traite toutes les requêtes dans un thread à part mais de manière séquentielle
- Messenger.getBinder() crée un IBinder
- Message.obtain() crée un Message
- Message.obtain(int what, int arg1, int arg2, Object obj)
- replyTo (optionel) Messenger pour la réponse
- Messenger.send(message) permet d'envoyer un message
- Un Handler permet de recevoir et traiter des Messages
- new Messenger(new Handler() { ... })

Workflow utilisant la classe Messenger (partie client) :

Implementer le listener ServiceConnection.

Surcharger les methods :

- onServiceConnected() : elle est appelée par le systeme pour deliver l'IBinder retourné par la methode onBind().
- onServiceDisconnected() : elle est appelée par le systeme android lorsque la connection avec le service est perdue, (i.e le service est tué ou tombe en panne).

Appeler bindService(), en passant l'implementation de ServiceConnection.

Lorsque le systeme appelle onServiceConnected(), le client peut interagir avec le service en utilisant l'interface Ibinder

Pour se deconnecter on appelle unbindService().

Workflow utilisant la classe Messenger (partie service) :

Le service implemente un Handler (sous forme de classe interne) qui traite les requetes du client.

le service cree un objet Messenger qui possede une reference sur le Handler precedent.

le Messenger cree un IBinder et le retourne au client (à partir de onBind()).

Le client utilise le IBinder pour instancier un Messenger (possedant une reference sur le Handler du service), avec le messenger, le client peut envoyer des messages (requetes) au service.

Le traitement des messages se fait dans la methode handleMessage() du Handler.

Service lié (Bound service) :

Les activitiés, les services, et les fournisseurs de contenu (content providers) peuvent se lier à un service (pas pour les broadcast receivers).

Un service donné peut réaliser les deux types prédéfinis (started, bound).

Voir le TP6 (Service lie)

5) Fournisseurs de contenu :

Fournisseur de contenu (CP) :

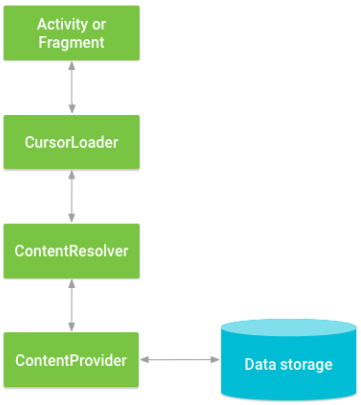
- C'est un composant d'une application qui n'a pas d'interface utilisateur UI
- Il encapsule et partage des données structurées
- Il constitue l'approche officielle d'android pour partager les données entre processus
- Encapsuler les données= offrir une interface standard pour
- inserer, modifier, rechercher, supprimer les données
- La methode de stockage des données est en dehors de la responsabilité du fournisseur de contenu.
- Le concepteur de l'application doit choisir sa propre technologie pour stocker les données de maniere permanente (SGBD SQLITE, fichiers XML,….)

Modèle de données :

Les données sont exposées sous forme de **tables (similaires aux BDD relationnelles)**
Un CP peut gerer plusieurs tables
Chaque table contient une colonne _ID qui assure un identifiant unique de chaque enregistrement
Quelques tables predefinies :

- UserDictionary.Words: ayant comme colonnes: id,word,frequency…
- Contacts: ayant comme colonnes: id,display_name,photo…
- CallLog.Calls

L'accés aux fournisseurs de contenu :



Classes impliquées :

package: android.content
classe abstraite ContentProvider : Encapsule les données

classe abstraite ContentResolver : C'est une classe intermediaire entre le composant client et le CP (elle appartient au processus client)

classe Uri du package: android.net : Fournit un moyen d'identification des tables et des lignes

Classe contrat

URI :

Uniform Resource Identifier : Décrite dans la norme RFC 2396

Syntaxe:

Nom du Schema : vaut "content " pour les CP
"://"
Authorité :dans ce cas c'est l'identifiant du CP
Chemin (Path): chemin ou adresse de la table
ID : c'est un parametre optionnel qui designe le numero de la ligne dans une table

Exemple d'uri (1)

Tous les URIs des CP commencent par content://

URI d'un CP: content:// com.example.cpetudiant

URI d'une table (appelée etudiant) : content:// com.example.cpetudiant/etudiant

URI d'une seule ligne (ligne avec _ID=24) : content:// com.example.cpetudiant/etudiant/24

Exemple d'uri (2)

content://com.android.contacts/contacts	représente une liste d'instances de contacts
content://com.android.contacts/contacts/5	représente le contact dont l'identifiant est 5.

Implementation d'un CP :

Étendre la classe ContentProvider impose l'implémentation des six méthodes suivantes :

- onCreate
- getType
- query
- insert
- update
- delete

le système android s'en charge d'instancier le CP (et pas l'utilisateur)

Hériter de la classe ContentProvider
Ajouter le composant crée dans le fichier manifest :

```
<manifest ... >
    ...
    <application ... >
        <provider android:name="com.example.cpetudiant.etudiantprovider"
        android:authorities="com.example.cpetudiant"
        android:exported="true"/>
        ...
    </application>
</manifest>
```

Classe contentprovider :

boolean onCreate() :

- Initialise la BDD du CP
- Appelée lorsque le premier composant client sollicite le CP

String getType(Uri uri) : Retourne le type MIME d'une table ou d'une ligne (tuple), la table ou le tuple sont identifiés avec l'uri.

int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)

- Met à jours toutes les lignes qui repondent aux criteres de matching (pour une table ou un tuple identifié par l'argument uri)
- Les nouvelles valeurs sont placées dans la structure values.
- La fonction retourne le nombre des lignes modifiées

Uri insert(Uri uri, ContentValues values)

- Elle insere une nouvelle ligne dans la table identifiée par l'argument uri.
- Elle retourne l'URI de la nouvelle ligne inserée

int delete(Uri uri, String selection, String[] selectionArgs)

- Supprime toutes les lignes qui repondent aux criteres de selection (pour une table ou un tuple identifié par l'argument uri).
- Retourne le nombre de lignes supprimées

Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)

- Execute une requete sur les lignes qui repondent aux criteres de selection (la table ou le tuple est deja identifié par l'argument uri)
- La projection contient une liste de colonnes qui composent le resultat.
- le resultat de la requete est retourné sous forme d'un objet "cursor"

ContentResolver :

- Les requetes du composant client ne sont pas executées directement par le CP mais acheminées tout d'abord à un objet ContentResolver
- L'objet ContentResolvers est crée grace à la fonction getContentResolver() (à partir d'une activité ou un autre composant)
- La classe ContentResolver contient les memes methodes que ContentProvider: insert(), update(), delete(), query(), ...
- Elle permet aussi de notifier les observateurs enregistrés lorsque les données du CP changent

Classe statique (ou la classe de contrat) :

Elle définit l'uri de l'autorité

Pour chaque table du CP , elle englobe une sousclasse qui définit :

- les noms des colonnes
- son uri (ie l'uri de la table en cours)
- son type mime
- le type mime de ses tuples

Type mime (suite à classe de contrat) :

La syntaxe générale d'un type mime est:

" type/soustype"

dans le cas d'une table: "vnd.android.cursor.dir/vnd.cpetudiant.contratetudiant_etudiant"

Dan le cas d'une ligne(tuple)

"vnd.android.cursor.item/vnd.cpetudiant.contratetudiant_etudiant"

Remarques :

- La classe CursorLoader permet l'éxecution des requêtes dans un thread worker (séparé du main thread).
- Oncreate est appelée une seule fois lors de l'instanciation du CP, les autres fonctions sont appelées potentiellement plusieurs fois.
- Les fonctions d'un CP doivent être synchronisées pour garder la cohérence de la BDD.
- Ex: public synchronized String getType(Uri uri)

Voir le TP7

6) Introduction à l'informatique embarquée :

Systèmes embarqués :

An embedded system is a combination of computer hardware and software—and perhaps additional parts, either mechanical or electronic—designed to perform a dedicated function.

Un calculateur autonome faisant partie d'un système plus large (électrique, mécanique, aéronautique, hydraulique..), et qui remplit une fonction dédiée (la marche d'un robot, la télécommunication, le contrôle d'un environnement (maison ou machines de production), consoles de jeux ….), ses ressources sont généralement limitées.

Il ne possède généralement pas des entrées/sorties standards et classiques comme un clavier ou un écran d'ordinateur

Le marché des systèmes embarqués (95%) > marché des PC+ serveurs (5%)

Domaines d'applications :

Vehicules (de 30 à 100 calculateurs)
avions (plus de 100 calculateurs)

Informatique embarquée en automobile :

- Réduction des émissions
- Prévention du dérapage (ABS)
- Contrôle des airbags
- Injection électronique
- Régulation de vitesse
- Contrôle de vitres, portes, essuie glasses…
- Multimédia
- ..

Exemples de cartes embarquées :

- Arduino (25$)
- ChipKIT (à base de PIC) (30 $)
- Raspberry Pi (à base d'ARM) (35 $)
- Intel edison (à base d' atome) (70 $)
- BeagleBoard de TI(à base d'ARM) (à partir de 95$)
- Parallela (à base d'ARM) (99 $)

Caractéristiques :

Fiabilité: est l'aptitude d'un dispositif à accomplir une fonction requise dans des conditions données pour une période de temps donnée
Maintenance: capacité d'un système à être rapidement et simplement (à moindre cout) réparé après une erreur
Disponibilité : le rapport entre la durée durant la quelle le système est opérationnel sur la durée totale qu' on aurait souhaité qu'il le soit
Sécurité: communication authentifiée et confidentielle
Faible consommation énergétique : faible capacité de mémoire, faible capacité de calcul
Contraintes sur la **dimension**, le **poids** …
Tolérance aux fautes (Dégradation harmonieuse des performances)

Cout :

- Les processeurs et les microcontrôleurs embarqués ont un faible cout par rapport aux processeurs d'informatique générale
- le prix moyen d'un processeur (tout type confondu), est estimé à 6 USD par unité
- Le prix moyen d'un processeur du PC est de 300 USD par unité

Temps réel (réactivité) :

- Un système temps-réel doit réagir aux stimuli de son environnement dans un délai temporel imposé par l'environnement / l'application.
- On distingue le temps réel dur et souple.

Exemples:

- système de freinage: le délai est de l' ordre 1/16 sec.
- Contrôle des airbags :le délai varie entre de 30 à 150 millisec
- Syteme de traitement d'une video 25 fps: délai est de l' ordre 40 millisec

Dans un système temps-réel, il faut que les résultats soient corrects, en plus il faut les fournir avant la date d'échéance.

- Résultats corrects au sens temporel (si respect du délai)
- Des résultats corrects mais tardifs sont faux.

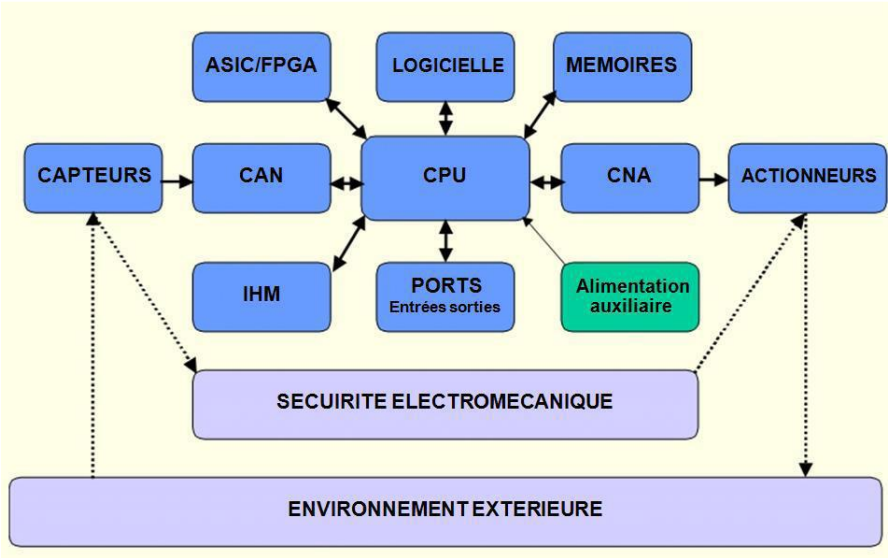
Dures (Hard real-time)

- Une contrainte est dite "dure", si son non respect résulterait en une catastrophe.
- Ex: Freinage d'un véhicule, Contrôle de centrale nucléaire, contrôle des airbags...

Lâches ou souple (Soft real-time)

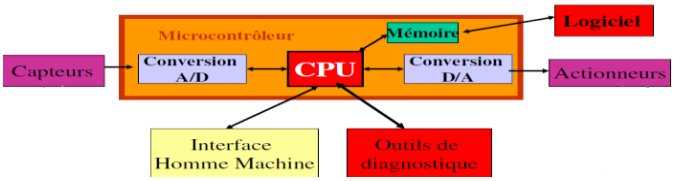
- Le non respect du délai provoque une mauvaise perception chez l'utilisateur
- Ex: systèmes téléphoniques, multimédia...

Architecture générale d'un SE :



Processeur vs microcontrôleur :

Processeur : circuit intégré contenant la CPU
Microcontrôleur : circuit intégré contenant la CPU, la mémoire RAM , ROM, Flash, les interfaces d'entrées/sorties en même temps



Architecture Harvard Vs Von Neumann :

Architecture Von Neumann :

- Les données et le code co-existent dans la mémoire centrale (avec un seul bus d'accés à cette mémoire)
- Utilisée principalement dans l' architecture X86 d'intel, ARM7

Architecture Harvard

- 02 mémoires séparées sont associées aux programmes et aux données des bus différents pour faire l'accés
- Possibilité de faire le parallélisme entre l'éxecution des instructions et le fetch des instructions
- Mmodèle employé dans les microcontrôleurs PIC, AVR, ARM9…

Processeurs CISC Vs RISC :

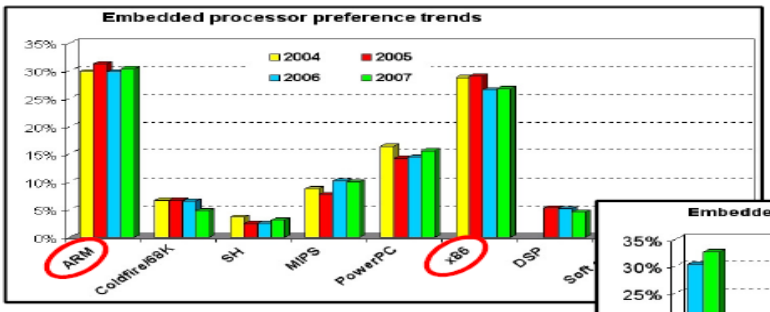
Complex instruction set computer :

- un nombre élevé d'instructions avec des tailles variables et de nombreux modes d'adressages
- L'execution d'une instruction peut durer plusieurs cycles
- La taille du code ASM est généralement petite
- EX: P4, I3, AMD ATHLON,..

Reduced instruction set computer :

- Un nombre réduit d'instructions
- Architecture Load/store
- Durée d'execution est fixe pour toutes les instructions (1 cycle d'horloge)
- La taille du code ASM est généralement longue
- Ex: ARM Cortex, PowerPC, MIPS, ITANIUM,…

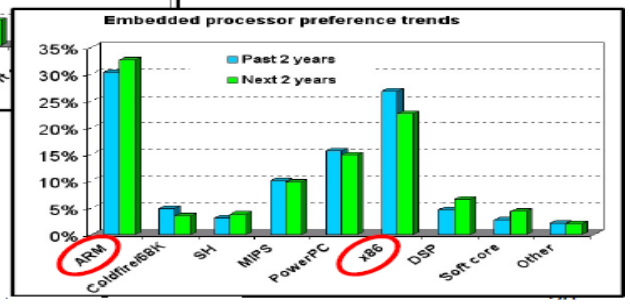
CPU du marché embarqué :

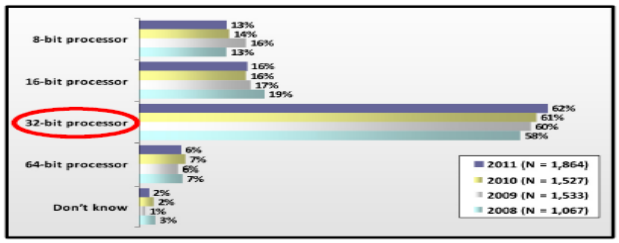


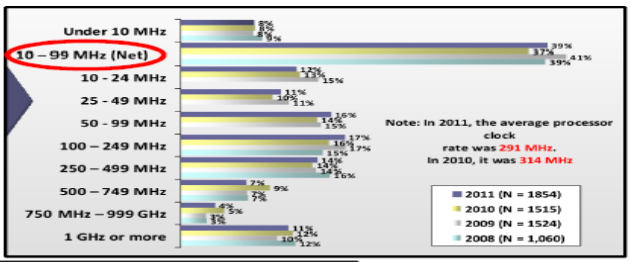
Les processeurs ARM et x86 occupent presque 60 % du marché.

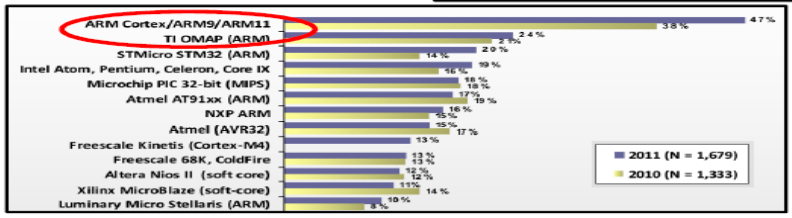
Les processeurs ARM sont très utilisés dans les téléphones « intelligents ».

L'infiltration du marché par les processeurs ARM continue de croître, au détriment des processeurs x86

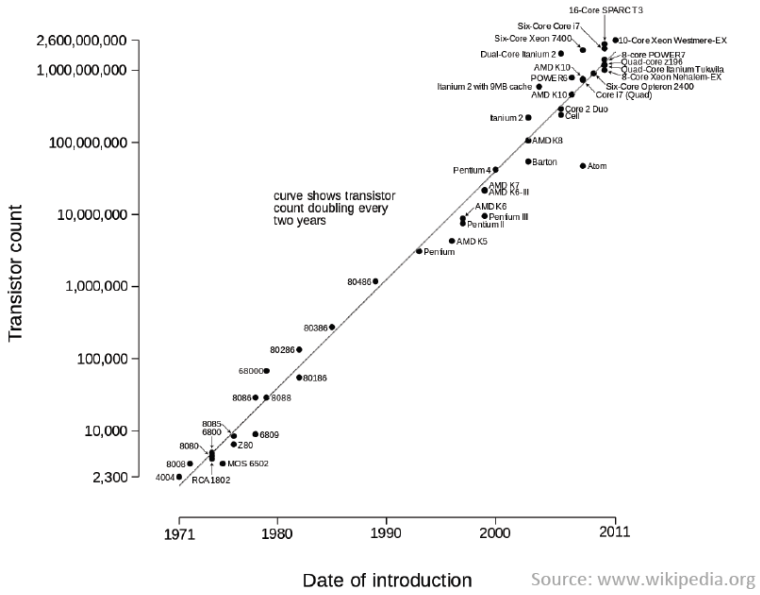








Loi de moore :



Problemes :

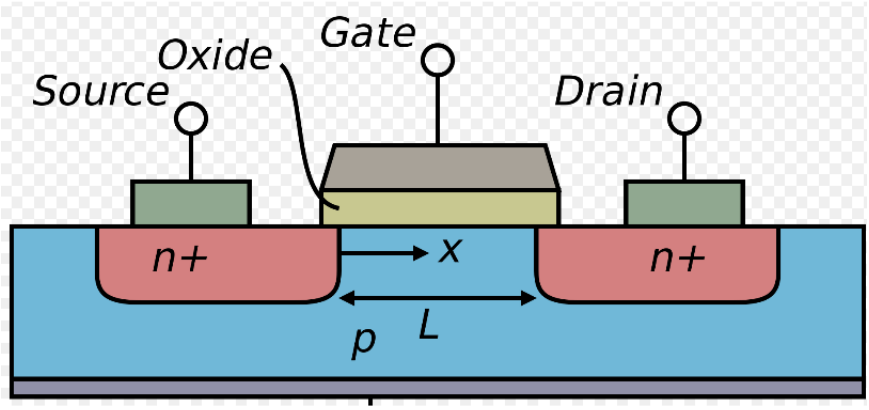
La vitesse d'accés à la mémoire RAM n'augmente qu'avec 1.1 toutes les 02 ans, en contradiction avec *2 pour la vitesse de calcul des CPU (horloge)

Ce mur de mémoire freine le calcul à haute performance

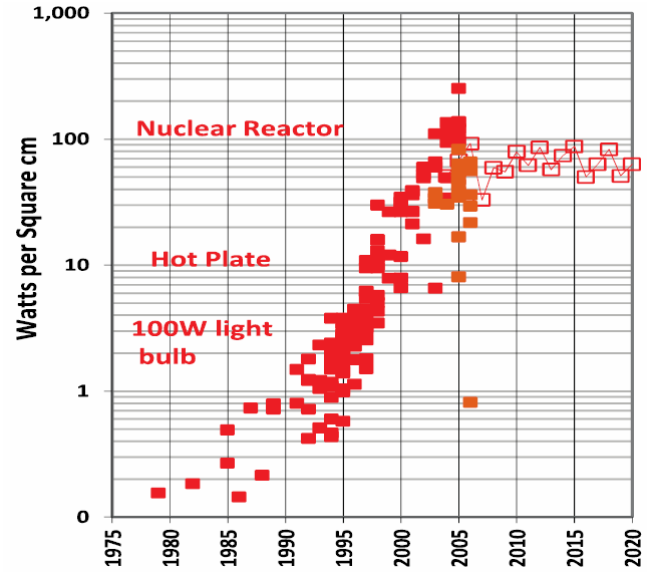
Rupture de la loi de moore :

- la dissipation de la chaleur au niveau des transistors Mosfet arrête l'augmentation de la fréquence blocage momentané de la loi de moore
- Le calcul à haute performance doit être implémenté avec d'autres idées

Transistor Mosfet :



Dissipation de la chaleur :



mono cœur Multi cœur

Causes de l'échauffement des transistors :

Consommation dynamique de l'énergie

- P=1/2 C V2 f .α
- V et f sont mathématiquement reliés (positivement)
- Une réduction de 20% de V autorise un gain de 4 % dans P (f reste fixé)

Consommation statique de l'energie

- La miniaturisation implique un courant de fuite de/vers l'element « gate »
- la réduction exagérée de V implique aussi un courant de fuite

solutions :

Au lieu d'augmenter la fréquence des horloges on augmente le nombre de « coeurs » sur une puce

- Réduction de la vitesse d'horloge
- Réduction du voltage

Fréquence limitée consommation réduite

Processeurs multi-coeurs posent de nouveaux défis

- Problèmes de synchronisation
- Problèmes de communication

ARM vs Atom Vs i7 :

	Cortex A15 (no L2, 32nm)	Cortex A9 (no L2, 40nm)	Atom N270 (45nm)	I7 960 (45nm)
Number of Cores	2 (4 maximum)	2 (4 maximum)	1 Core, 2 HT threads	4 Cores, 8 HT threads
Frequency	1Ghz – 2.5 Ghz	800Mhz (Po) 2Ghz (Per)	1.6 Ghz	3.2 Ghz
Out-of-Order?	Yes	Yes	No	Yes
L1 cache size	32KB I/D	32KB I/D	32KB I/D	32KB I/D
L2 cache size	N/A	N/A	512KB	1MB + 8MB L3
Issue Width	4	4	2	4?
Pipeline Stages	?	8	16	14 ~ 24 (?)
Supply Voltage	?	1.05V (Per)	0.9 – 1.1625 V	0.8-1.375 V
Transistor Count	?	26,00,000?	47,000,000	731,000,000
Die size	?	4.6 mm2 (Po) 6.7 mm2 (Per)	26 mm2	263 mm2
Power Consumption	?	0.5 W (Po) 1.9 W (Per)	2.5W (TDP)	130W (TDP)

Exemples de consommation :

- Pentium M 1.5 Banias (130 nm) 1.5 GHz 24.5 W
- Pentium 4 HT 530 Prescott (90 nm) 3 GHz 84 W
- Core i3-540 Clarkdale (32 nm) 3.06 GHz 73 W
- Core i5-660 Clarkdale (32 nm) 3.33 GHz 73 W
- Core i7-2600 Sandy Bridge (32 nm) 3.4 GHz @4 Cores 95 W
- Dual-core PowerPC MPC8641D 90 nm 2 GHz 1.2 V 15-25 W
- Itanium 2 1500 Madison (130 nm) 1.5 GHz 130 W
- Phenom X3 8750 Toliman (65 nm) 2.4 GHz 95 W(quad core)
- Raspberry Pi 3 1.2 GHZ (ARM Cortex-A53) de 1.5 W - 6.7 W
- Arduino uno (dissipation moyenne) 0.7 W

Domaines d'utilisation :

Processeurs power pc (RISC) de freescale/IBM

- PlayStation III
- Nintendo
- Cameras, routeurs CISCo, véhicules (General Motors)

Processeurs Motorola de freescale (CISC)

- Imprimantes laser
- Stations de travail SUN/HP, macintosh d'apple
- Console de jeux arcade

Processeurs MIPS(RISC) de MIPS Technologies,

- PlayStation II
- Nintendo
- Routeurs
- Tablettes, smartphones

Processeurs / micro-controleur d' ARM(RISC)

- Smartphoes , tablettes, smart watch (montres)
- netbook, ipod media player
- Robots lego Mindstorms de NXT
- Tablettes, smartphones, PDA Cameras (canon) Nintendo_DS, Smart TV
- Arduino, Raspberry Pi

Microcontroleurs Atmel (dont le coeur est nommé AVR)

- Ils suivent l'architecture Harvard 8 bits RISC
- Disponibles dans :
  - dans les cartes arduino
  - Robots lego Mindstorms de NXT

Informatique embarquée vs générale :

Informatique générale :

• Processeur standard
– Multiples unités fonctionnelles
(flottant, exécution désynchronisée, prédiction…)
– Fréquence élevée (> GHz)
– Consommation électrique élevée
– Chaleur
– Taille
• MMU (mémoire virtuelle)
• OS
• Cache
• Grand nombre de périphériques

Informatique Embarquée :

• Processeur dédié (contrôleur)
– Architecture adaptée
– Vitesse faible (~200 MHz)
– 8-32bits : mémoire limitée
– Basse consommation
– Petite taille, faible coût
• Processeur DSP (traitements)
– Très puissants
• Quelques Mo de mémoire
• RTOS
• Pas de cache