Android Intent, IntentFilter Receiver

jean-michel Douin, douin au cnam point fr
version : 14 Mars 2017

Notes de cours

Sommaire

- ***Activity***
  - Rappel du cycle de vie

- **Le patron publish/subscribe**
  - Intent comme notification,
  - intentFilter comme abonnement
  - Receiver comme souscripteur
  - Activity comme publieur

- **MVC**
  - Modèle, Vue, Contrôleur

- **Génie logiciel, conception**
  - Approche par composants Logiciels induite, comme méthode de conception ?

- **Annexes**

Intergiciel : fonctionnalités attendues

- **Devenir une activité**
  - Quel « protocole » doit-on respecter ?
    - Quelles sont les classes à hériter ?
    - Quelles sont méthodes à redéfinir ?, quel cycle de vie ? Est-il est imposé ?

- **Communiquer**
  - Entre deux activités, entre deux applications, entre deux processus,
  - Notification

- **Sélection de la « bonne » application**
  - Une activité se met à la disposition des autres … comment ?
  - Cette activité devient un fournisseur,
  - Maintenance, déploiement,
  - Substitution d'une application par une autre
  - …

Bibliographie utilisée

CCY114/SMB116 : Certificat de compétences : intégrateur d'applications mobiles Cnam

http://developer.android.com/resources/index.html

Le cours de Victor Matos

http://grail.cba.csuohio.edu/~matos/notes/cis-493/Android-Syllabus.pdf

http://www.vogella.com/android.html

http://www.cs.unibo.it/projects/android/slides/android_intents.pdf

http://www.cs.unibo.it/projects/android/index.html

Plusieurs livres

Android A Programmers Guide - McGraw Hill

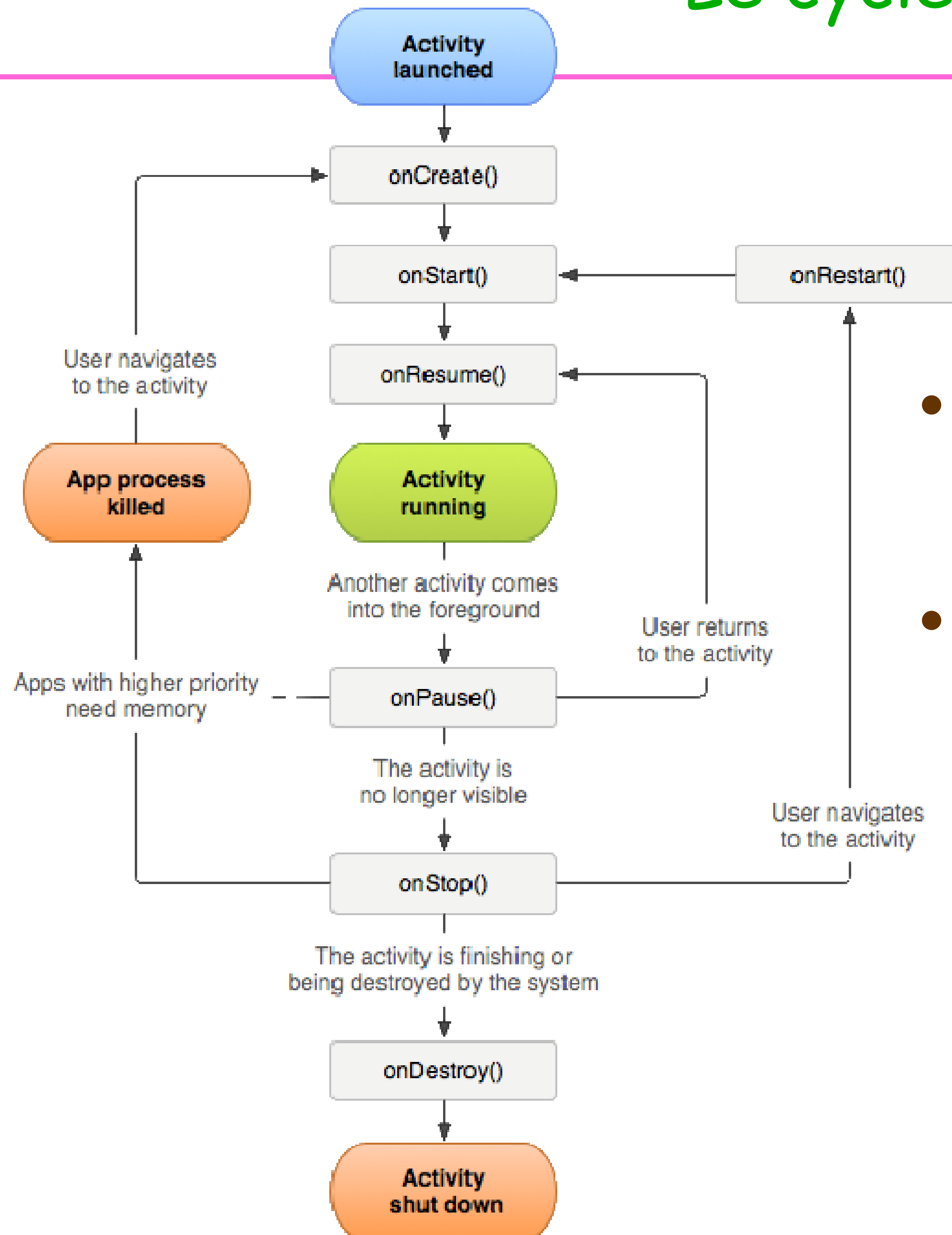
Professional Android Application Development – Wrox

Le livre de Mark Murphy - Pearson

Présentation

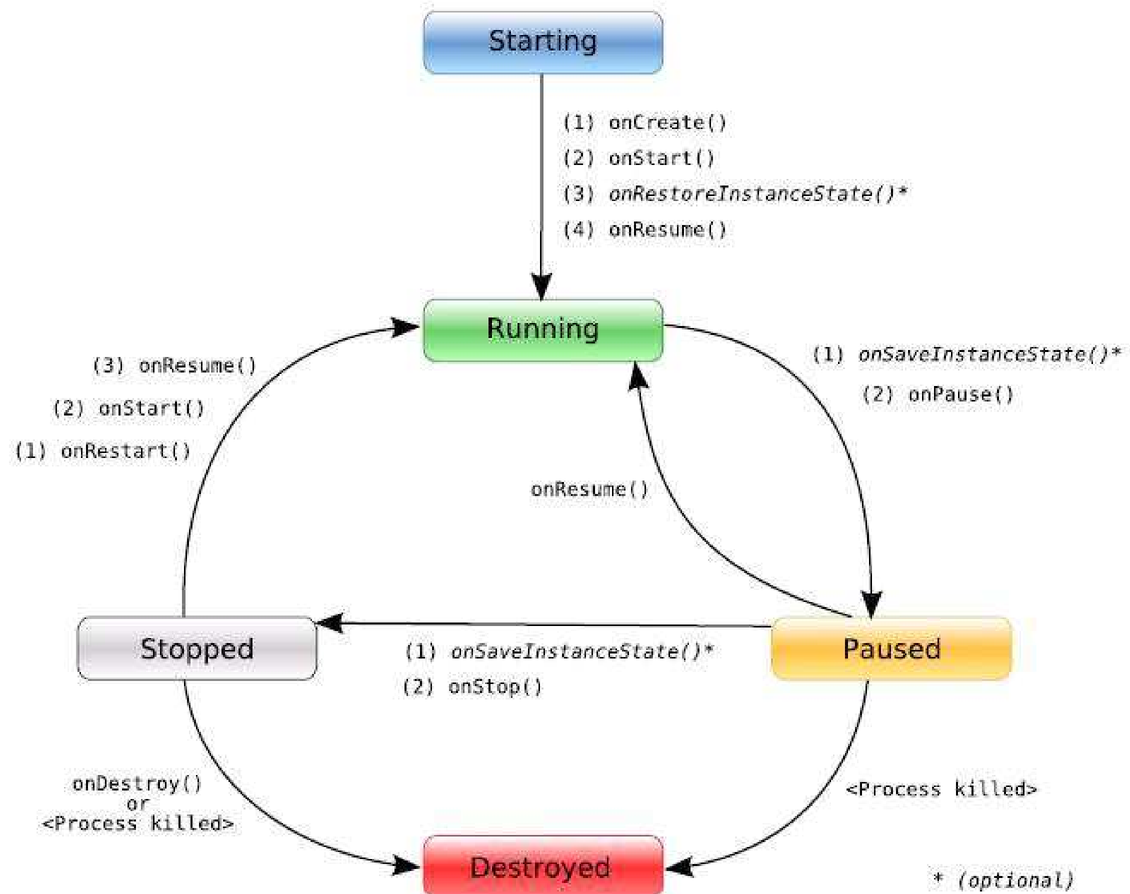
- **Une application peut être constituée de plusieurs écrans,**
- **A chaque écran lui correspond une activité,**
- **Une activité hérite et redéfinit certaines méthodes,**
  - **onCreate, -> …, onPause, -> …, onStop, ->…onDestroy, -> …**
- **Android se charge des appels de ces méthodes,**
  - **Un état de l'activité est donc induit par l'exécution de ces méthodes**
- **Android impose un cycle de vie, un état de l'activité.**
  - **Inversion de contrôle**
    - (cf. M.Fowler) http://martinfowler.com/articles/injection.html

Le cycle de vie d'une activité



- Séquencement imposé
  - Cf. M. Fowler, inversion de contrôle
- Android a le contrôle
  - Impose le séquencement

États d'une Activité



- http://inandroid.in/archives/tag/activity-lifecycle

Activité

- **Créer une activité c'est**
  - **Hériter de la classe *android.app.Activity;***
  - **Redéfinir certaines méthodes**
    - **onCreate**
      - super.onCreate *cumul de comportement (obligatoire)*
      - initialisation *des variables d'instance*
      - setContentView *affectation de la vue*
    - **onStart,**
      - super.onStart *cumul du comportement*
    - **onResume …**
      - super.onResume *cumul du comportement*
    - **onPause …**
      - super.onPause *cumul du comportement*

Activity les bases, *HelloWorldActivity.java*

- **Une simple Activité, HelloWorld** *(encore)*

```
import android.app.Activity;

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

onCreate

- super.onCreate *cumul de comportement (obligatoire)*
- setContentView *affectation de la vue*

Intent : un message

- **Explicite**
  - **Intent intent = new Intent(…..);**
  - **La cible est précisée**

- **Implicite**
  - **La cible ou les cibles dépendent du filtre IntentFilter**

- **Un ou des souscripteurs peuvent être notifiés**

Architecture

- **Nous avons :**
  - **Des souscripteurs de certaines intentions,**
  - **Des publieurs d’intentions,**
  - **Un mécanisme de résolution de la bonne activité,**
  - **Le système Android se charge de tout.**
- **Alors serait-ce le patron Publish-Subscribe ?**

Android Publish Subscribe

- Le patron publish-subscribe
- Publish
  - Intent
  - sendBroadcast
  - sendOrderedBroadcast
- Subscribe
  - IntentFilter
  - BroadcastReceiver
    - onReceive

Patron Publish-Subscribe

- **Ce patron permet aux abonnés ayant souscrit à un type d'événement, d'être prévenus lorsque ceux-ci se produisent.**

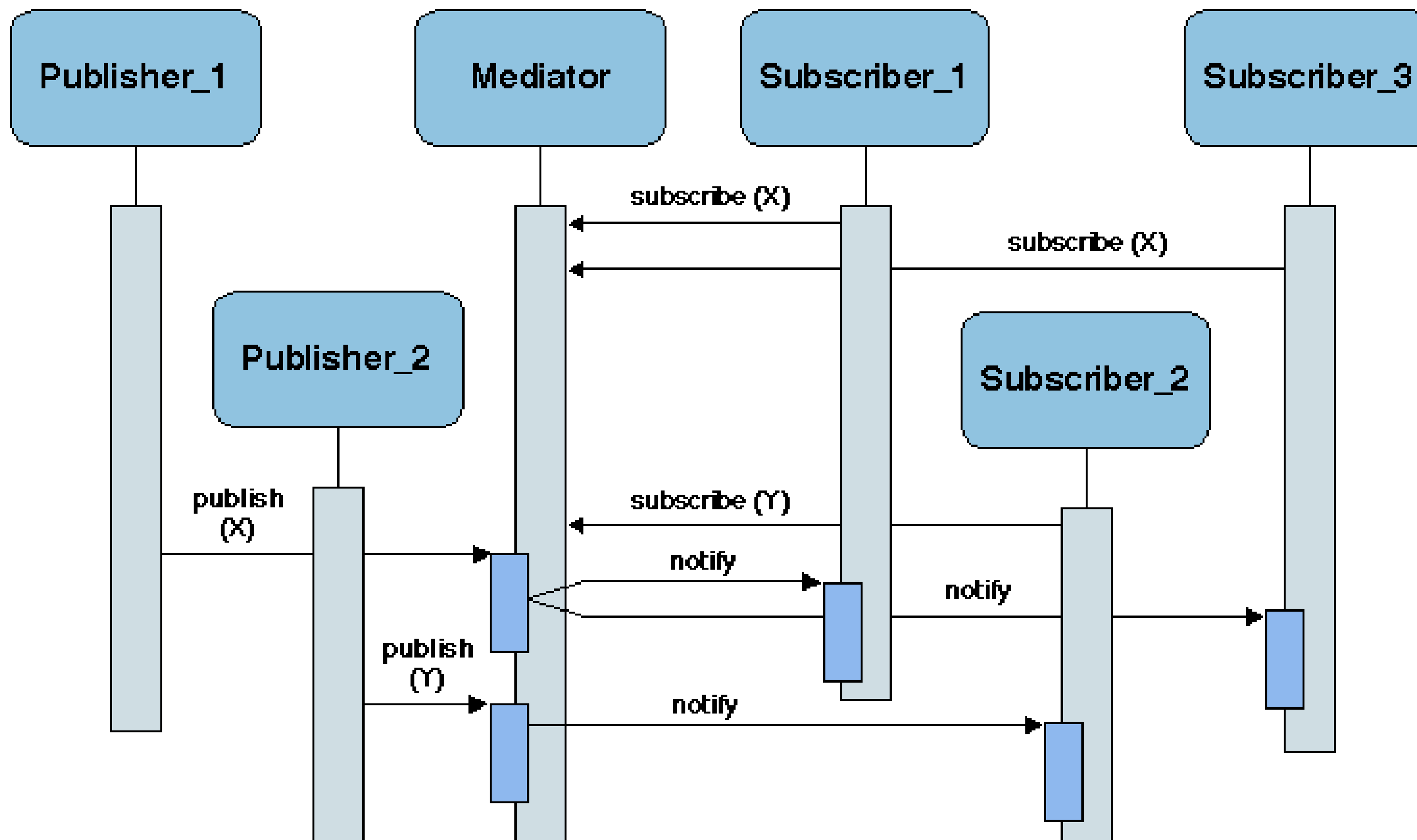
- **Un médiateur se charge de l'inscription des abonnés aux différents thèmes de souscription et assure la notification des évènements aux abonnés concernés.**

- **La notification peut être effectuée selon la priorité des souscripteurs, l'un des souscripteurs peut arrêter la propagation.**

Architecture

- **Nous avons :**
  - **Des souscripteurs de certaines intentions,**
  - **Des publieurs d'intentions,**
  - **Un mécanisme de résolution de la bonne activité,**
  - **Le système Android se charge de tout.**
- **-> patron Publish-Subscribe**

Publish-Subscribe



- source: http://lig-membres.imag.fr/krakowia/Files/MW-Book/Chapters/Events/events-body.html

Publish-subscribe / *pull-push*

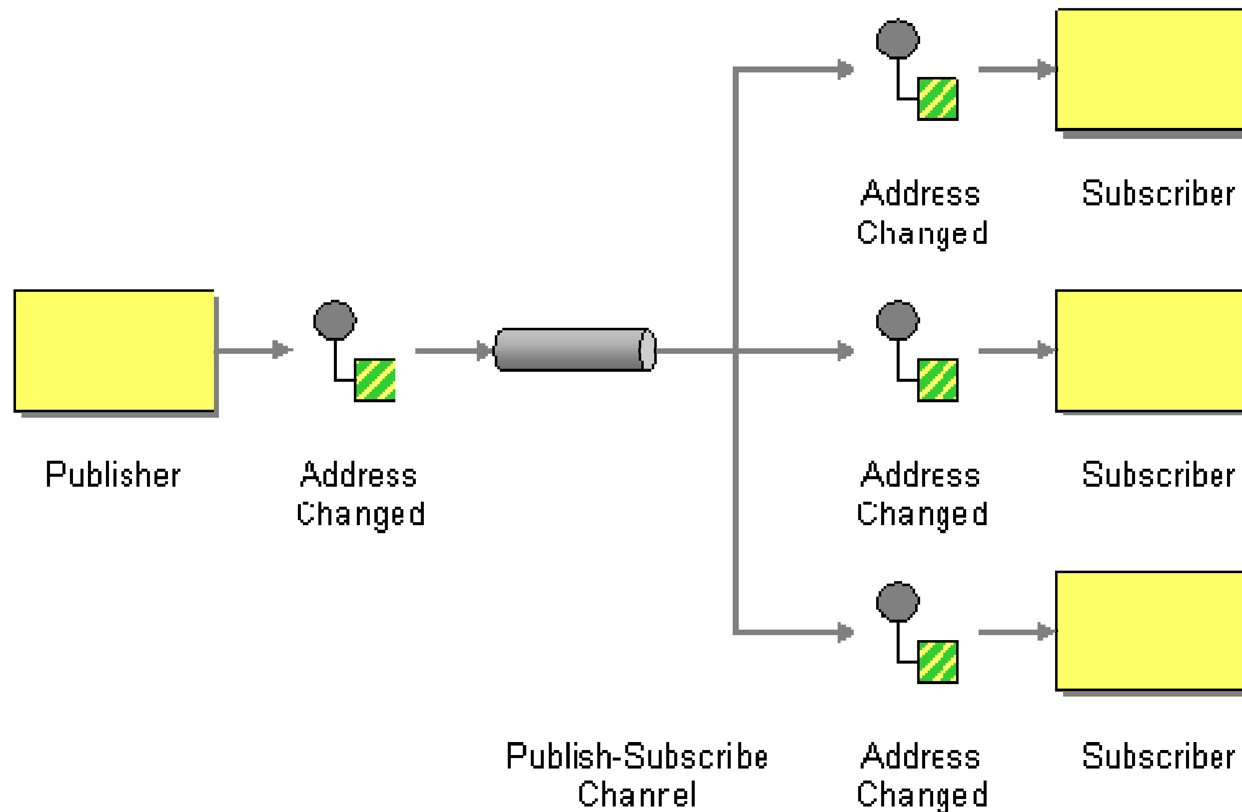
- Un forum de discussion …
  - **Enregistrement** d'un « client » à un sujet de discussion,
  - Un des « clients » décide de poster un message,
  - Les utilisateurs **à leur initiative** vont chercher l'information,
  - ***Publish-subscribe, mode pull***

- Les listes de diffusion, logiciels de causerie,
  - **Abonnement** d'un « client » à une liste de diffusion,
  - Un des « clients » décide de poster un message,
  - Tous les abonnés **reçoivent** ce message,
  - Les abonnés peuvent installer un **filtre** sur les contenus des messages,
  - ***Publish-subscribe, mode push***

Publish Subscribe

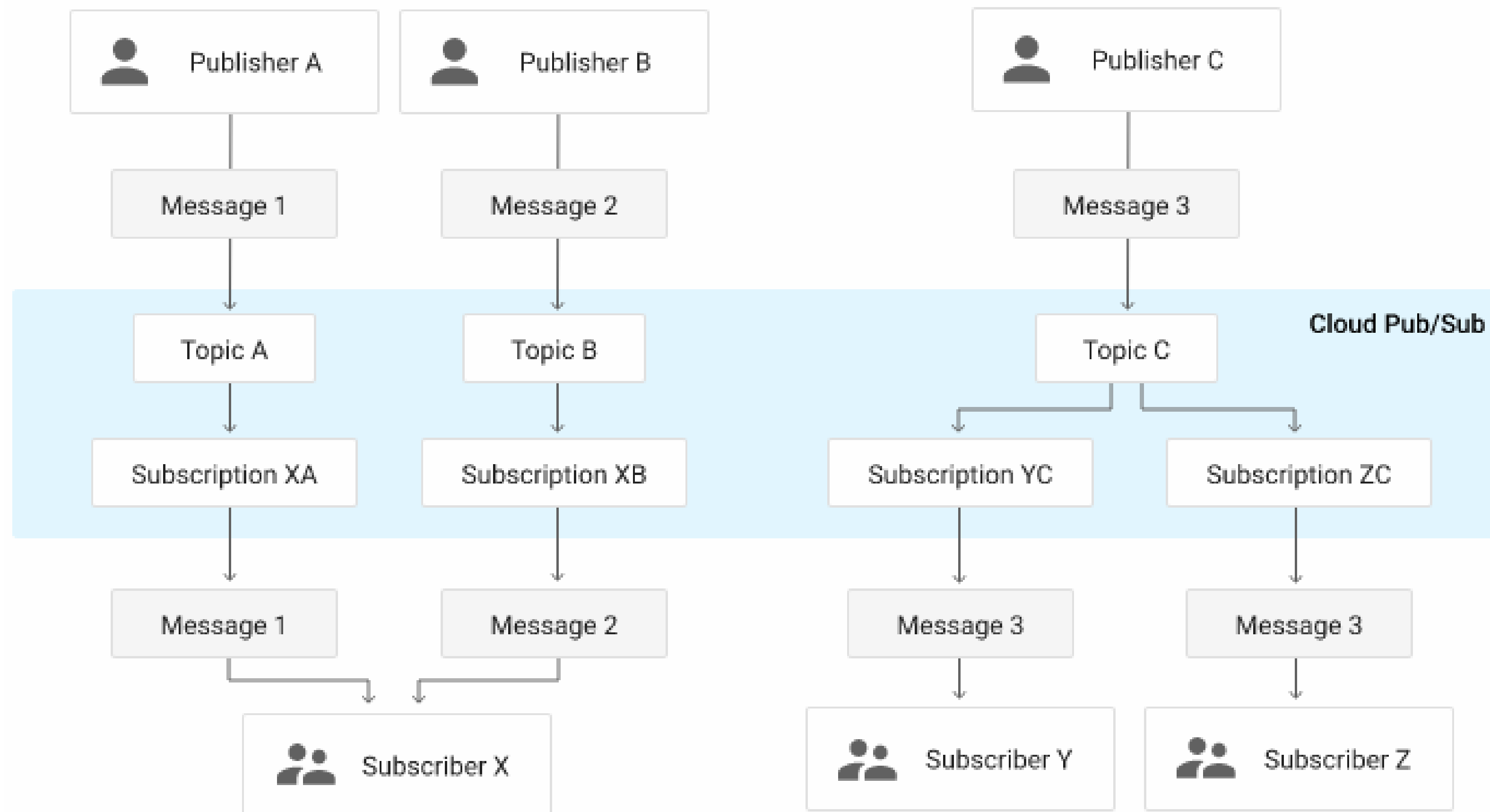
Enterprise
Integration
Patterns

- **Greg Hoppe**
  - http://www.enterpriseintegrationpatterns.com/
  - http://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html

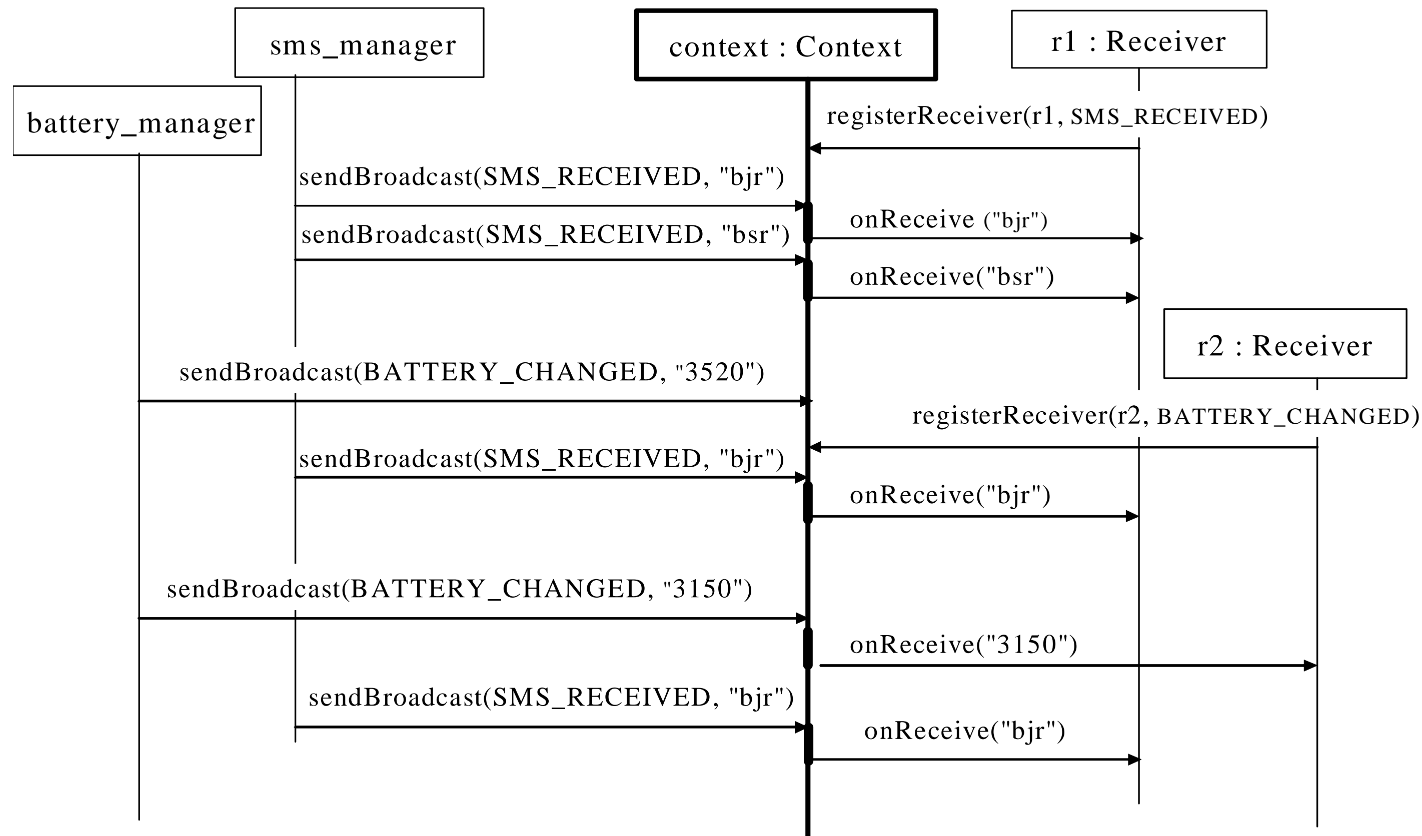


Google Cloud Pub/Sub

- **https://cloud.google.com/pubsub/docs/overview**



Une exemple en une séquence



Une notification d'évènements « SMS_RECEIVED » et « ACTION_BATTERY_CHANGED» ,
Un médiateur de gestion des souscriptions et de notifications (*context : Context*),
Un souscripteur r1 au thème « SMS_RECEIVED »,
Un souscripteur r2 au thème « ACTION_BATTERY_CHANGED».

La classe Context

La classe *Context* contient les opérations :

- d'abonnement d'un souscripteur (registerReceiver) pour un thème de publication,
  - Plusieurs thèmes de souscription par receveur sont possibles,
- de retrait d'un souscripteur (unregisterReceiver) ,
- de publication (sendBroadcast) aux souscripteurs concernés,
- de publication selon la priorité des souscripteurs (sendOrderedBroadcast),
  - Un arrêt de la publication peut être décidé par l'un des souscripteurs.

La classe Context, un extrait

```
public class Context{

  /** Enregistrement d'un souscripteur.
   *  @param receiver le souscripteur
   *  @param filter le ou les thèmes de souscription
   */
  public void registerReceiver(BroadcastReceiver receiver, IntentFilter filter){...}


  public void unregisterReceiver(BroadcastReceiver receiver){...}

  /** Publication sur ce thème.
   *  Les souscripteurs abonnés à ce thème sont tous notifiés.
   *  Note : l'appel de la méthode abortBroadcast par un souscripteur
   *         est sans effet, cf. sendOrderedBroadcast
   *  @param intent le thème de publication
   */
  public void sendBroadcast(Intent intent){...}

  /** Publication ordonnée sur ce thème.
   *  Les souscripteurs sont notifiés selon leur priorité.
   *  Un souscripteur notifié peut interrompre la notification par
   *  l'appel de la méthode abortBroadcast.
   *  @param intent le thème de publication
   */
  public void sendOrderedBroadcast(Intent intent){...}

}
```

Intent, un extrait

- **La classe *Intent* correspond à la notification,**
  - **l'action transmise précise son type.**

```
public class Intent{
  public static final String ACTION_BATTERY_CHANGED     = "ACTION_BATTERY_CHANGED";
  public static final String BATTERY_STATUS_FULL        = "BATTERY_STATUS_FULL";
  public static final String ACTION_BATTERY_LOW         = "ACTION_BATTERY_LOW";
  public static final String ACTION_POWER_CONNECTED     = "ACTION_POWER_CONNECTED";
  public static final String ACTION_POWER_DISCONNECTED = "ACTION_POWER_DISCONNECTED";
  public static final String SMS_RECEIVED               = "SMS_RECEIVED";

 public Intent{...}

 public Intent setAction(String action){...}
```

  - **Exemple :**

```
Intent intent = new Intent();
Intent.setAction(Intent.SMS_RECEIVED);
```

IntentFilter

- **La classe *IntentFilter* précise les thèmes de souscription et éventuellement la priorité associée**

```
public class IntentFilter{

  public IntentFilter(){...}

  public Iterator<String> actionsIterator(){...}
  public void addAction(String action){...}
  public void setPriority(int priority){ {...}
  public int getPriority(){...}
}
```

- **Exemple :**

```
IntentFilter filter1 = new IntentFilter();
filter1.addAction(Intent.SMS_RECEIVED);
filter1.addAction(Intent.ACTION_BATTERY_LOW);
filter1.setPriority(100);
```

BraodcastReceiver

- **La classe abstraite *BroadcastReceiver* implémente les méthodes permettant un abonnement à un ou plusieurs thèmes de publication**
  - **Ces thèmes sont précisés dans une instance de la classe *IntentFilter* .**

- **Cette classe abstraite laisse à ses sous classes la responsabilité d'implémenter la réception d'une notification**
  - **méthode onReceive.**

```
public abstract class BroadcastReceiver{

    public abstract void onReceive(Context context, Intent intent);
```

Receveur, en résumé

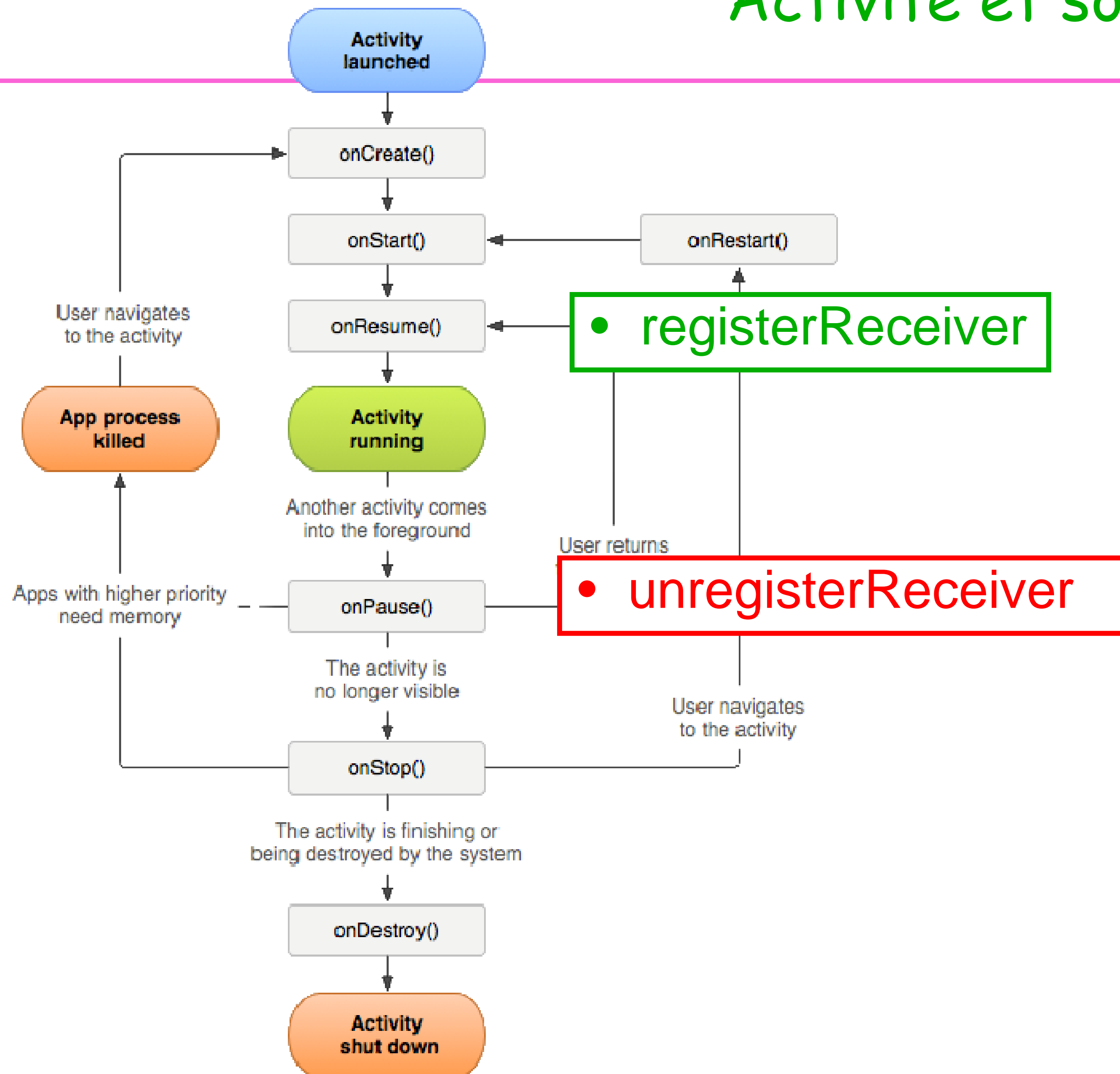
- **Un « Broadcast receiver » est à l'écoute des « Intents »**

- **Le système Android délivre les « Intents » à tous les receveurs candidats, lesquels s'exécutent séquentiellement sans ordre particulier (sendBroadcast)**
  - **Les receveurs candidats avec le même filtre**

- **L'enregistrement du receveur se fait**
  - **par programme**
    - **getApplicationContext().registerReceiver(receiver, filtre)**
  - **Ou bien avec la balise**
    - **<receiver> du fichier AndroidManifest.xml**
      - **<intent-filter>**

- **Par programme sera préféré**
  - **Cycle de vie d'une activité**

Activité et souscription



- registerReceiver

- unregisterReceiver

Envoi de l'intent par l'émetteur

- L'envoi de l'intent est effectué par Android (en asynchrone) à tous les receveurs candidats selon un ordre indéfini :

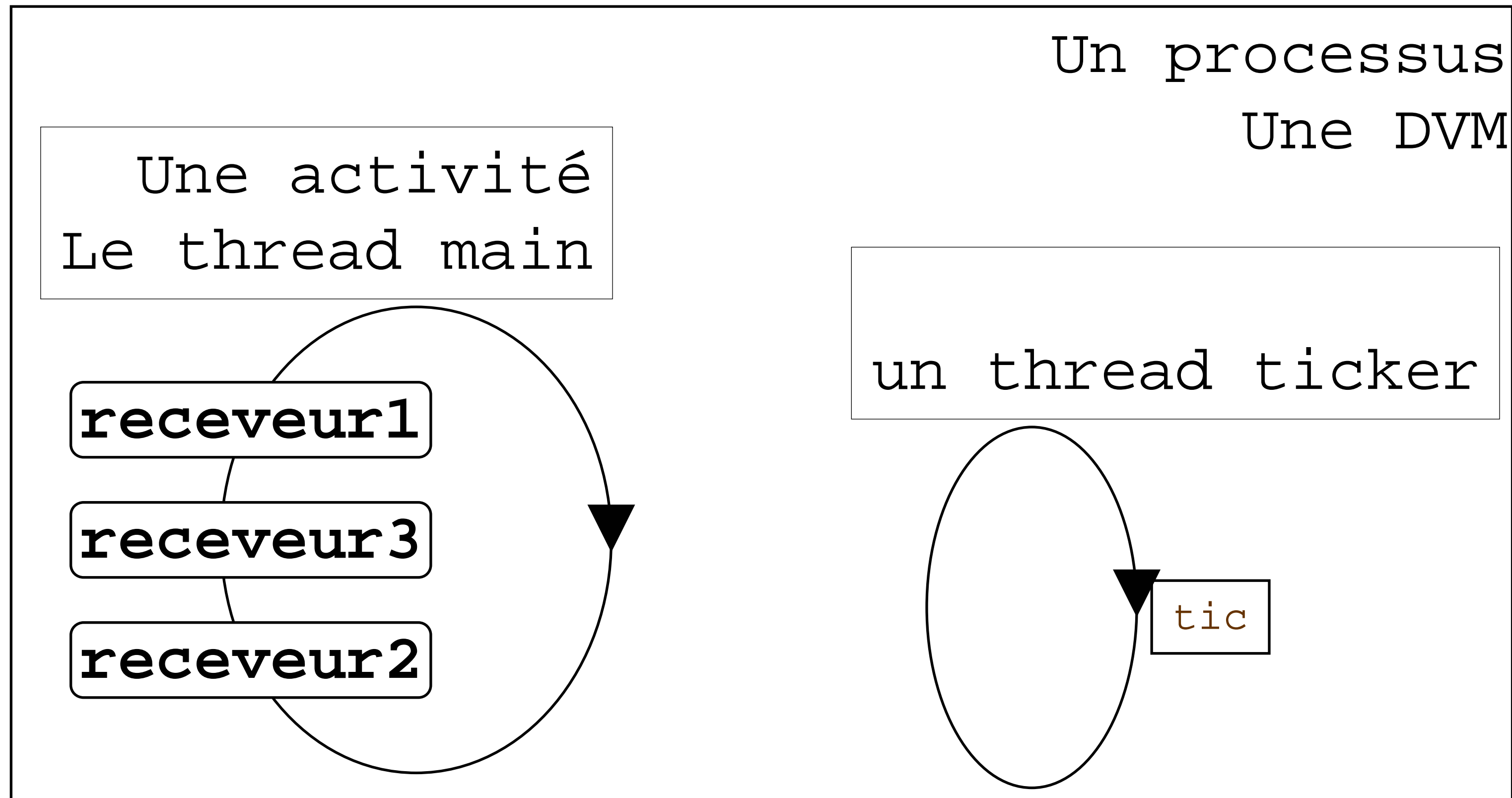
(context.)sendBroadcast

- L'envoi de l'intent est délivré selon une relation d'ordre définie entre les receveurs,
  (*android:priority attribut du filtre, setPriority*)
  - un receveur peut interrompre la propagation (*abortBroadcast*) :

(context.)sendOrderedBroadcast

cf. le Pattern Chaîne de responsabilités dans lequel la chaîne est une liste ordonnée, mise en œuvre uniquement par l'appel de sendOrderedBroadcast

Un exemple



- Au sein de l'activité,
  - 3 receveurs
  - *Un Thread interne à l'activité*

"time.action.TIC"
Une action définie par l'utilisateur

Traitement de l'intent par le receveur

- La méthode onReceive est exécutée, l'intent est transmis

void onReceive(Context ctxt, Intent msg);

```
IntentFilter filter1 = new IntentFilter();
filter1.addAction("time.action.TIC");
filter1.putExtra("count", count);
//filter1.addAction(Intent.ACTION_BATTERY_LOW);
```

Exemple suite, l'activité suivie du receveur

- **Au sein de l'activité**
  - **Un démarrage du Thread depuis l'IHM**

```
public void onClickStart(View v){
    startThread();
}
```

  - **Une association du filtre et du receveur dans le programme**
    - **méthode onResume()**

- registerReceiver

```
IntentFilter filter= new IntentFilter("time.action.TIC");
// filter.setPriority(X); X >= 0 pour un envoi ordonné
receiver  = new TimeReceiver();
getApplicationContext().registerReceiver(receiver1, filter);
getApplicationContext().registerReceiver(receiver2, filter);
getApplicationContext().registerReceiver(receiver3, filter);
```

Exemple suite : le receveur

```
public class TimeReceiver
    extends BroadcastReceiver {


  @Override
  public void onReceive(Context ctxt, Intent intent) {
      …
  }

}
```

Exemple suite : Le thread

- **A chaque seconde écoulée**

  1. **Création de l'intent avec le bon filtre pour les receveurs**
  2. **La valeur du compte est placé dans l'intent**
  3. **Propagation à tous les receveurs candidats**

```
public void run(){
    while(!ticker.isInterrupted()){
       SystemClock.sleep(1000);
       count.set(count.longValue()+1L);

1.          Intent intent= new Intent("time.action.TIC");
2.          intent.putExtra("count", count.get());

3.          sendBroadcast(intent);
    – Ou bien
            sendOrderedBroadcast(intent);

    }
 }  // avec AtomicLong count;
```

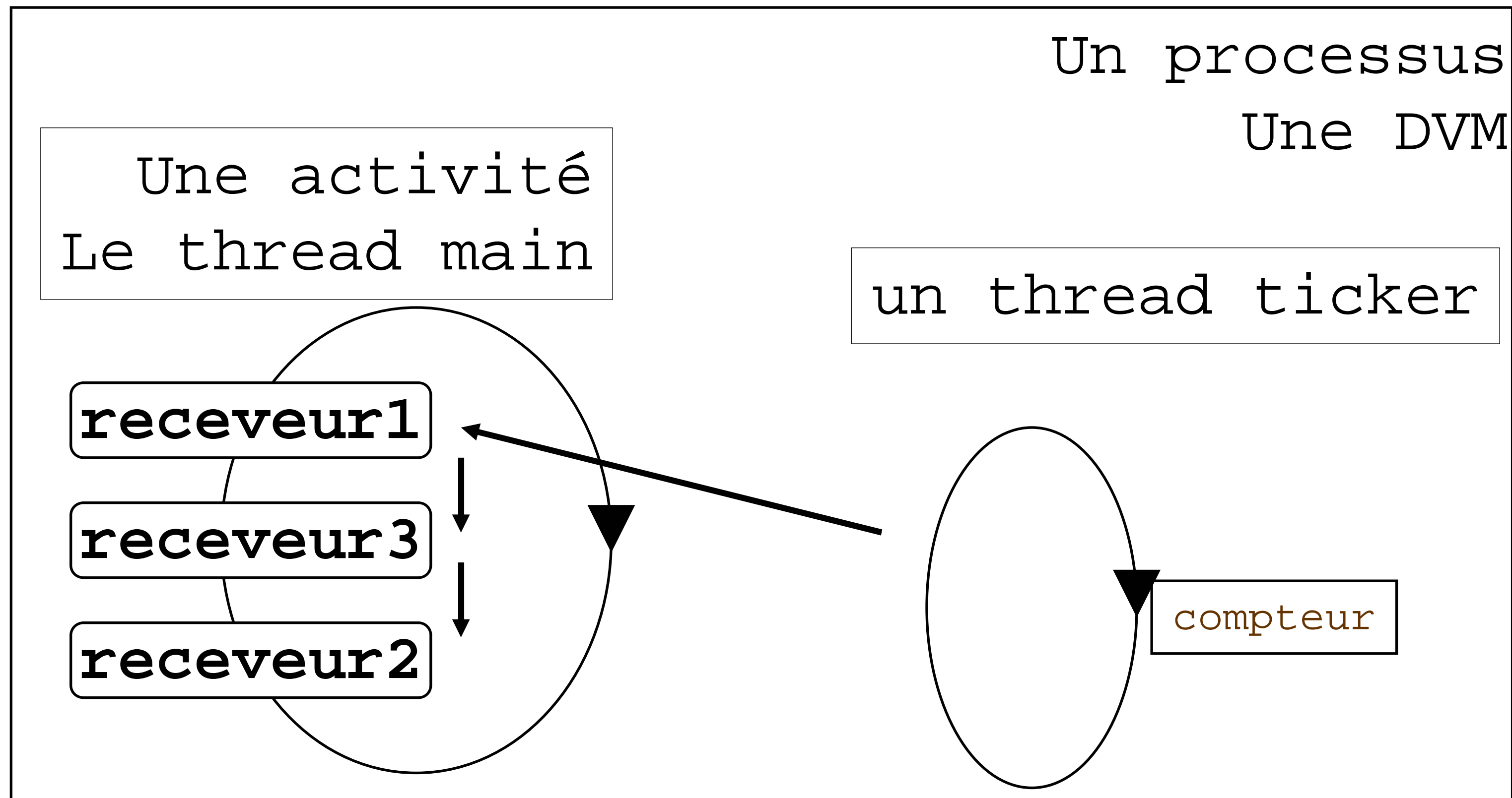
Exemple suite et fin

- **Fin du thread Ticker**
  - **stopThread**
    - **Ticker.interrupt**
  - **unregisterReceiver**

```
public void onPause(){
    super.onPause();
    stopThread();
    unregisterReceiver(receiver1);
    unregisterReceiver(receiver2);
    unregisterReceiver(receiver3);
}
```

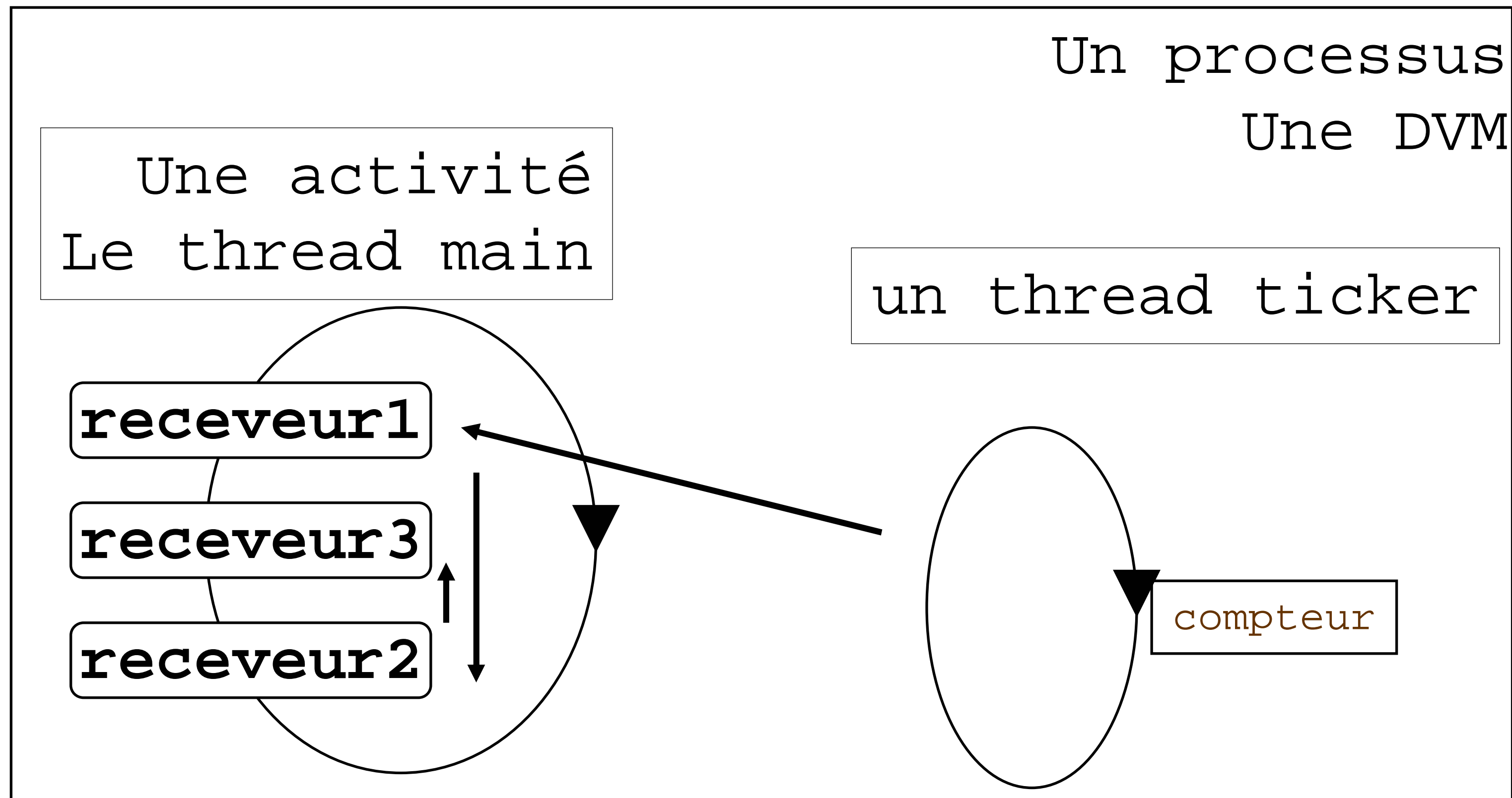
- unregisterReceiver

Ticker: un scénario possible avec **sendBroadcast**



- **sendBroadcast** (asynchrone)

Scénario **sendOrderedBroadcast**



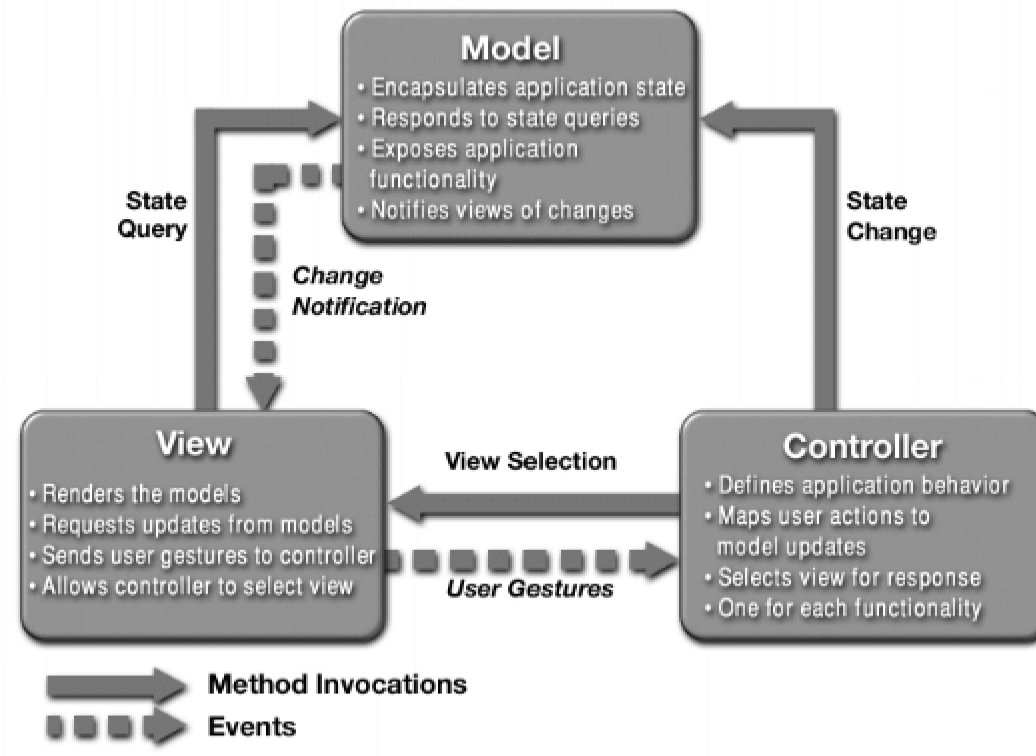
- **sendOrderedBroadcast** (asynchrone)
- *En fonction de la priorité*
  - *priorité(receveur3) > priorité(receveur2) > priorité(receveur1)*

Architecture logicielle

- **MVC**

- **Une proposition pour une discussion**

Architecture MVC, rappel



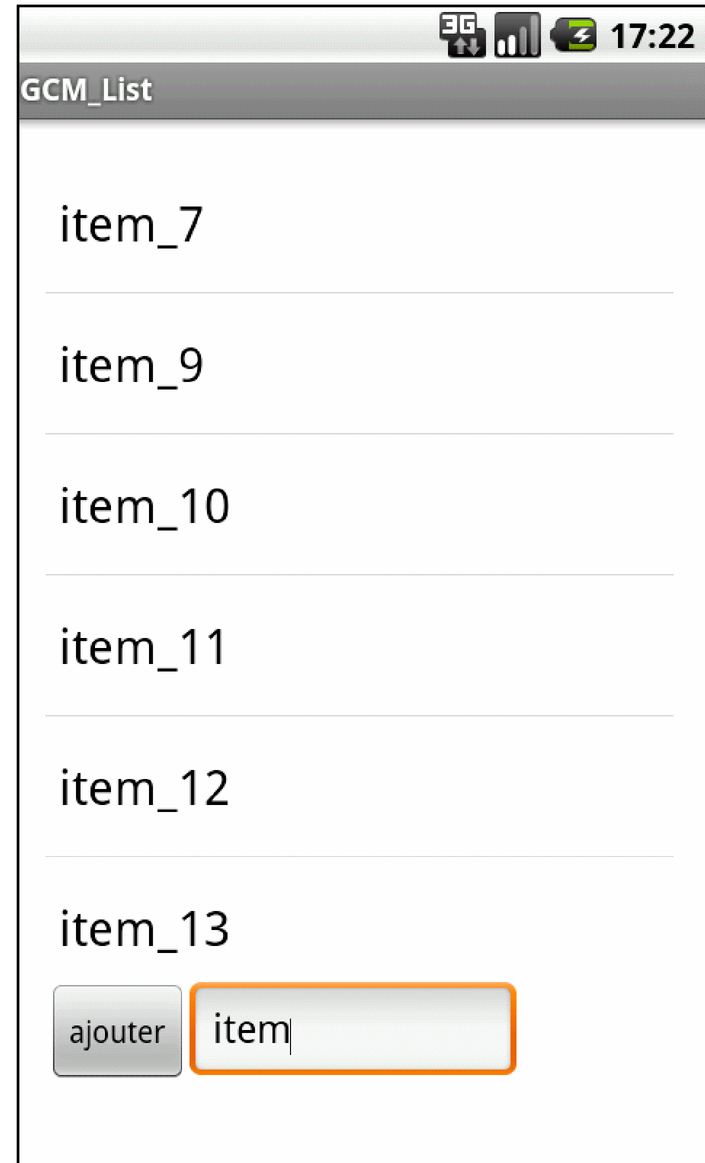
- **Application:**
  - **Une liste d'item : le modèle, du java standard, portable**
  - **ListView + ListActivity : la Vue**
  - **Un Receiver : le Contrôleur**

Android

- **Les outils nécessaires**
- **Intent**
- **IntentFilter**
- **BroadcastReceiver**
  - **registerReceiver**

La vue, une liste d'items

- **Affichage**
- **Opérations**
  - **d'ajout et de suppression**



La liste d'items

•le modèle

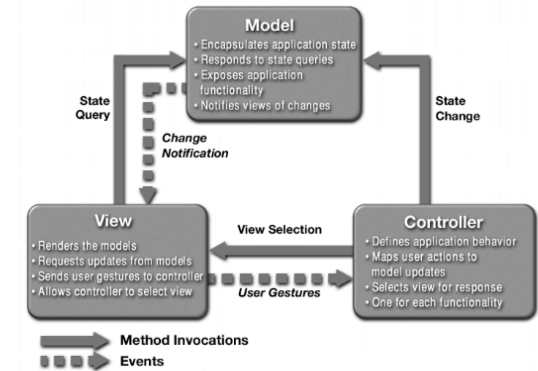
Items
extends Observable

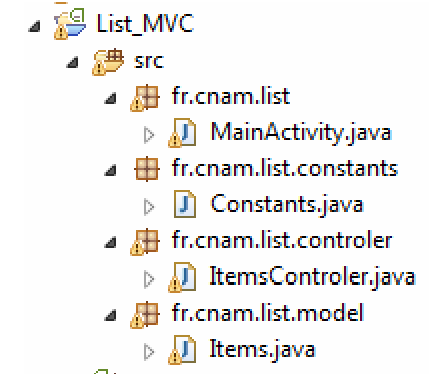
•la vue

MainActivity
extends ListActivity
implements Observer

le contrôleur

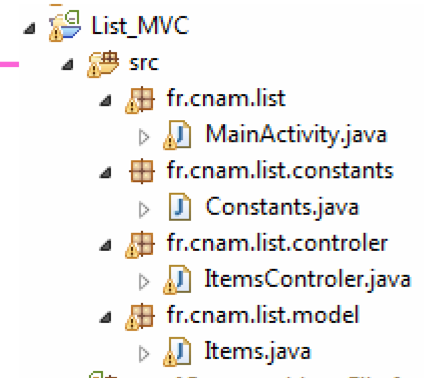
ItemsController
extends BroadcastReceiver





- **Items : le modèle**
- **MainActivity : la vue**
- **ItemsController : le contrôleur**

La classe Items : le modèle

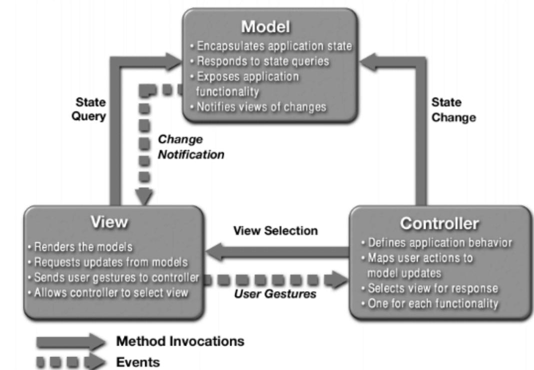



```java
public class Items extends Observable {

    private String name;
    private LinkedList<String> items;


    public Items(String name, Observer obs){
        this.name = name;
        this.items = new LinkedList<String>();
        this.addObserver(obs);
    }


    public void retirer(int position){
        synchronized(this){
            String str = this.items.remove(position);
            setChanged();
            notifyObservers(str);
        }
    }

    public void ajouterAuDebut(String item){
        synchronized(this){
            this.items.addFirst(item);
            setChanged();
            notifyObservers(item);
        }
    }
```

```java
public List<String> getList(){
    return items;
}

public String getName(){
    return name;
}
```

- **Java J2SE portable**
  - synchronized(this) par précaution (plusieurs contrôleurs)

Architecture suite

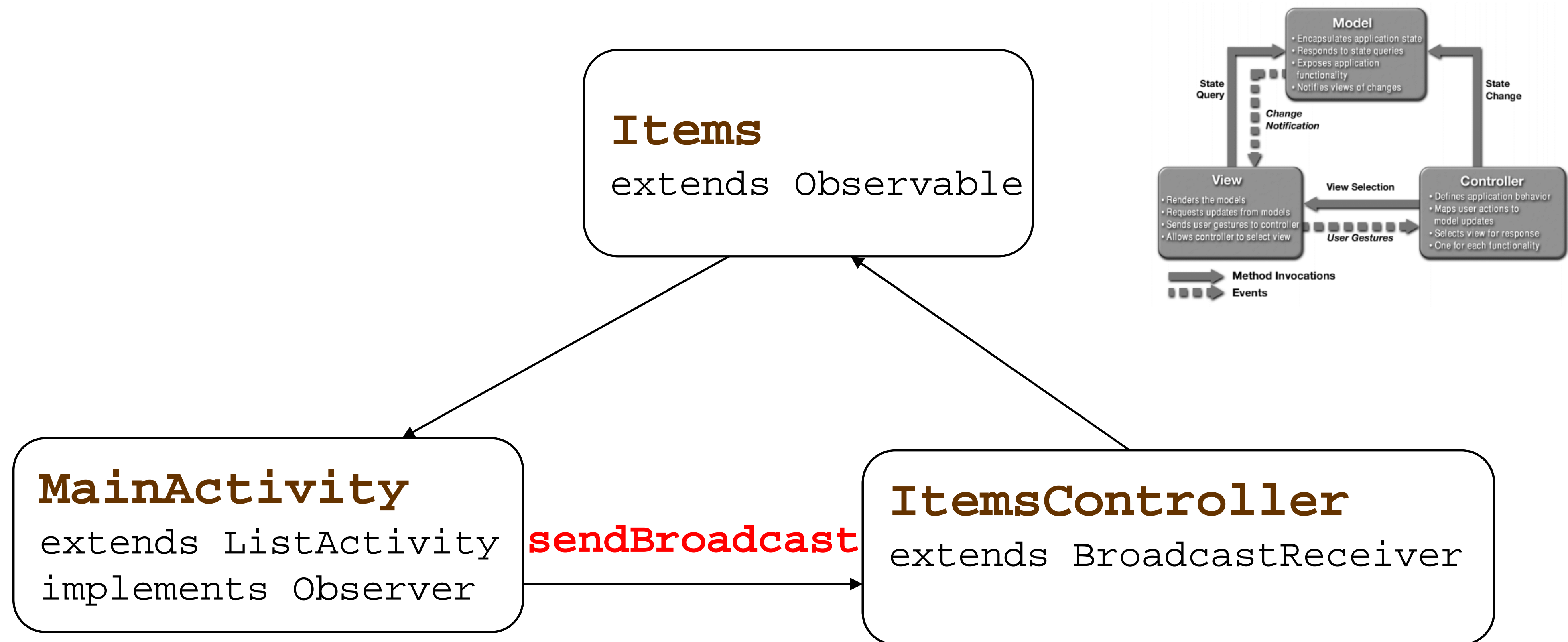


- **Items : le modèle**
  - **extends java.util.Observable**

- **MainActivity : la vue**
  - **extends android.app.ListActivity implements java.util.Observer**

- **ItemsController : le contrôleur**
  - **extends android.content.BroadcastReceiver**

Le contrôleur est un BroadcastReceiver



- **A chaque Clic → sendBroadcast**
  - Intent intent = new Intent();
  - intent.setAction(ItemsController.ACTION);
  - …
  - sendBroadcast(..

Action de l'utilisateur, gérée par le contrôleur

MainActivity
extends ListActivity
implements Observer

sendBroadcast →

ItemsController
extends BroadcastReceiver

```
// Operation d'ajout d'un item,
public void onClickAjouter(View v){
    EditText et = (EditText)findViewById(R.id.itemTexteId);
    String item = et.getText().toString();
    Intent intentToItemsController = new Intent();
    intentToItemsController.setAction(ItemsController.ACTION);
    intentToItemsController.putExtra(OPERATION_KEY,ItemsController.ADD_TOP);
    intentToItemsController.putExtra(DATA_KEY,item);
    sendBroadcast(intentToItemsController);
}
```

- **sendBroadcast**

ItemsController

```java
import fr.cnam.list.model.Items;

public class ItemsController extends BroadcastReceiver {

    public static final String ACTION = "fr.cnam.gcm.list.model.ITEMS";

    public static final String ADD        = "add";        // valeurs
    public static final String ADD_TOP    = "addtop";
    public static final String REMOVE     = "remove";

    private Items items;                  // le modèle
    private Context context;              // Android framework

    public ItemsController(Context context, final Items items){
        this.context = context;
        this.items = items;  // le modèle
    }

    @Override
    public void onReceive(final Context context, final Intent intent) {

        String operation = intent.getStringExtra(OPERATION_KEY);
        if(operation.equals(ADD_TOP))
            items.ajouterAuDebut(intent.getStringExtra(DATA_KEY));
        else if(operation.equals(REMOVE))
            items.retirer(Integer.parseInt(intent.getStringExtra(DATA_KEY)));
        else if(operation.equals(ADD))
            items.ajouter(intent.getStringExtra(DATA_KEY));

    }
```

- **A chaque « clic » la méthode onReceive est exécutée**
- *abortBroadcast(); si non cumul du comportement*

Contrôleur -> Modèle

Items
extends Observable

```
@Override
public void onReceive(final Context context, final Intent intent) {

    String operation = intent.getStringExtra(OPERATION_KEY);
    if(operation.equals(ADD_TOP))
        items.ajouterAuDebut(intent.getStringExtra(DATA_KEY));
```

ItemsController
extends BroadcastReceiver

```
intentToItemsController.setAction(ItemsController.ACTION);
intentToItemsController.putExtra(OPERATION_KEY,ItemsController.ADD_TOP);
intentToItemsController.putExtra(DATA_KEY,item);
sendBroadcast(intentToItemsController);
```

- **Appel de la méthode ajouter du modèle**

Modèle -> Vue

Items
extends Observable

```
@Override
public void onReceive(final Context context, final Intent intent) {

    String operation = intent.getStringExtra(OPERATION_KEY);
    if(operation.equals(ADD_TOP))
        items.ajouterAuDebut(intent.getStringExtra(DATA_KEY));
```

```
@Override
public void update(Observable arg0, Object arg1) {
```

MainActivity
extends ListActivity
implements Observer

- **La méthode update est déclenchée au sein de l'activité**
  - **Attention au contexte si actualisation de l'IHM, (runOnUiThread…)**

La Vue 1/4 initialisation

```java
public class MainActivity extends ListActivity implements Observer{
    // pour l'appel de Log.i
    private final String TAG = getClass().getSimpleName();

    // la Vue : ListView, cf. le fichier XML

    private Items items; // le Modèle

    private ItemsController itemsController; // le Controleur


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.activity_main);

        this.items = new Items("items", this); // le modèle avec this son observateur
        this.itemsController = new ItemsController(this, items); // le contrôleur du modèle items

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, android.R.id.text1,items.getList());
        setListAdapter(adapter);

        for(int i=0;i<20;i++) items.ajouter("item_" + i);  // une liste avec quelques items

    }
```

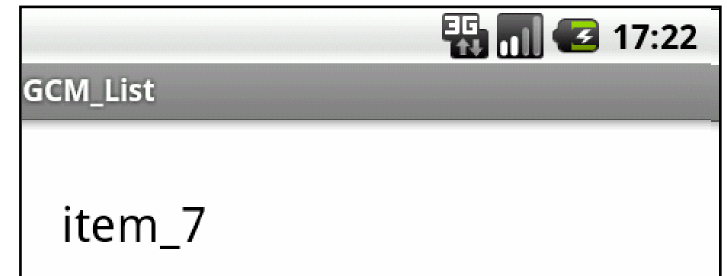
- **onCreate de l'activité**
  - **Création du modèle et du contrôleur**

La Vue 2/4 enregistrement du contrôleur

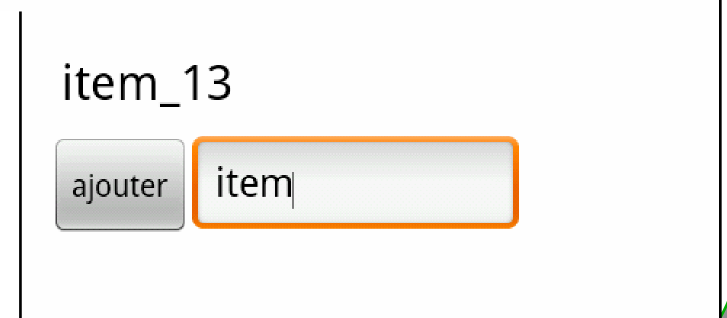
```
@Override
public void onResume(){
    super.onResume();
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(ItemsController.ACTION);
    registerReceiver(itemsController, intentFilter);
}


@Override
public void onPause(){
    super.onPause();
    unregisterReceiver(itemsController);
}
```

- **onResume de l'activité** *(ou onCreate, dépend de l'application)*
  - **enregistrement du contrôleur**
- **onPause** *(ou onDestroy)*

La Vue 3/4 A chaque clic !




```
// Operation d'ajout d'un item,
public void onClickAjouter(View v){
    EditText et = (EditText)findViewById(R.id.itemTexteId);
    String item = et.getText().toString();
    Intent intentToItemsController = new Intent();
    intentToItemsController.setAction(ItemsController.ACTION);
    intentToItemsController.putExtra(OPERATION_KEY,ItemsController.ADD_TOP);
    intentToItemsController.putExtra(DATA_KEY,item);
    sendBroadcast(intentToItemsController);
}
```



- **onClickAjouter**

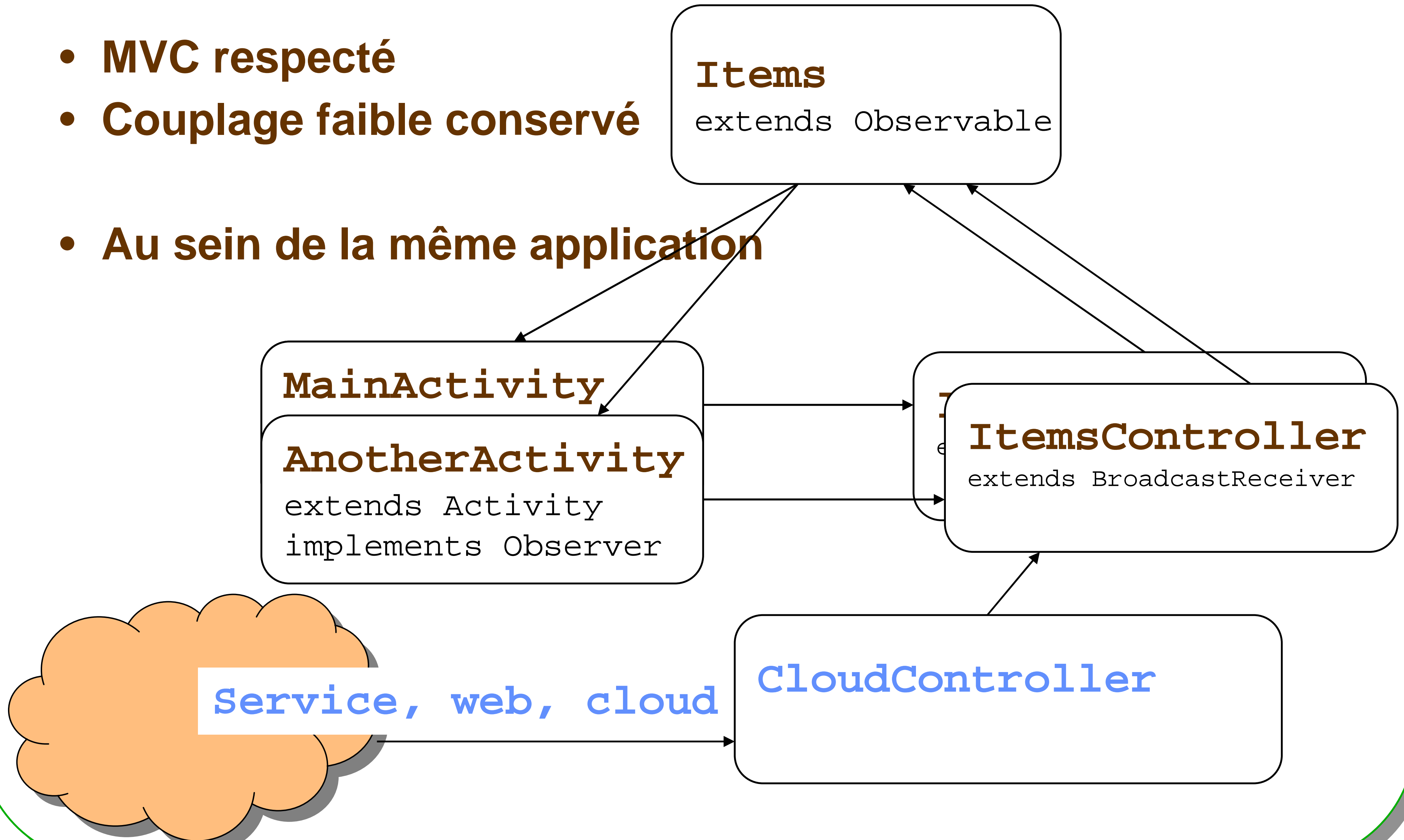
La Vue 4/4 update

```java
@Override
public void update(Observable arg0, Object arg1) {
//((ArrayAdapter<String>)this.getListAdapter()).notifyDataSetChanged(); // mise à jour de la vue
  runOnUiThread(new Runnable(){
      public void run(){
      ((ArrayAdapter<String>)getListAdapter()).notifyDataSetChanged();
      }
  });
}
```

- **update appelée par le modèle**
  - **(extends Observable)**
  - **La vue est un observateur(implements java.util.Observer)**
  - **runOnUiThread au cas où**

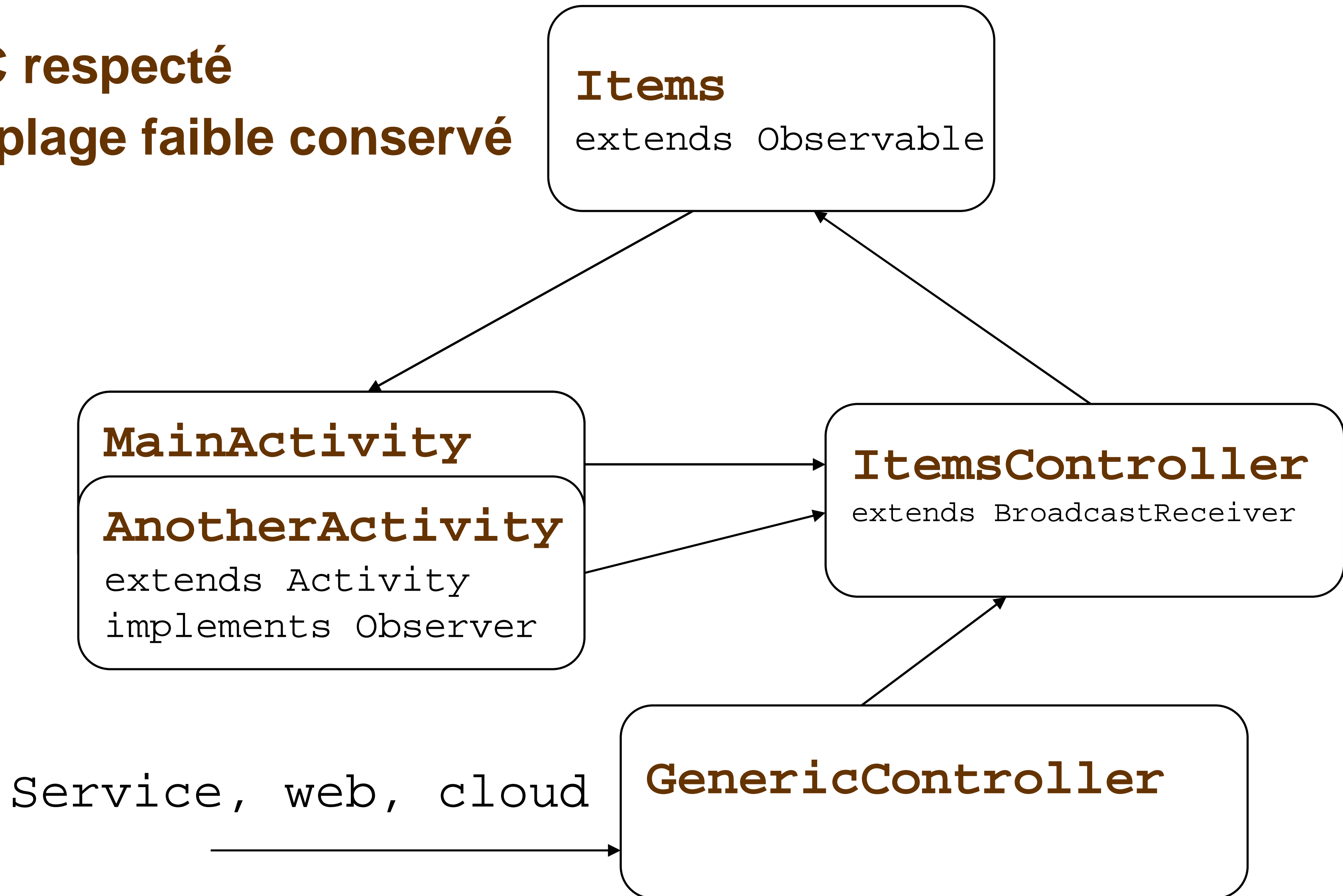
Discussion

MVC



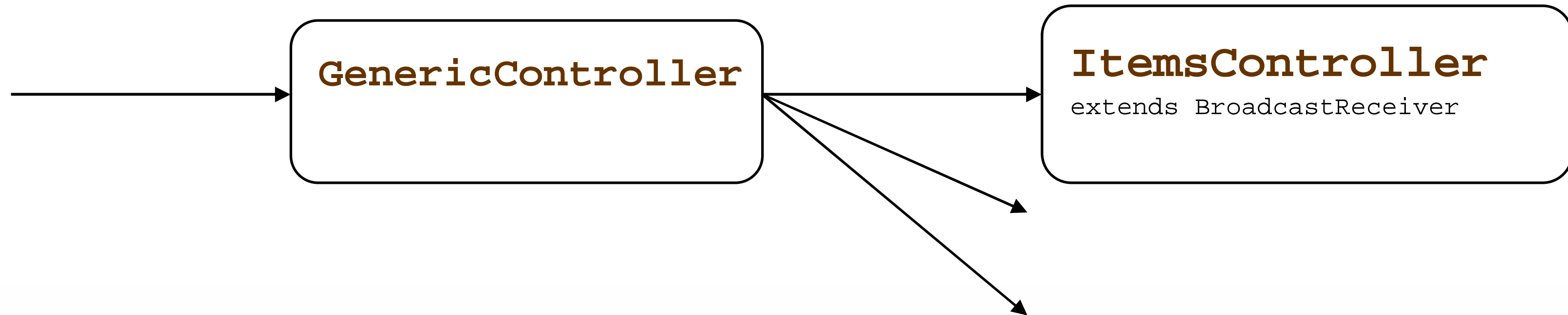
Généralisation

- **MVC respecté**
- **Couplage faible conservé**



- **Le champ Action sélectionne le contrôleur ad'hoc**

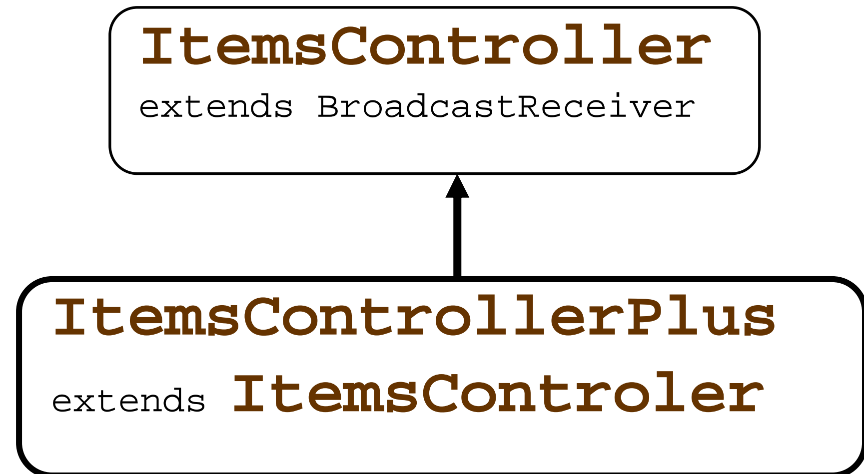
GenericController




```
@Override
public void onReceive(final Context context, final Intent intent) {

    Intent intentToController = new Intent();
    intentToController.setAction(intent.getStringExtra(ACTION_KEY));
    intentToController.putExtra(OPERATION_KEY,intent.getStringExtra(OPERATION_KEY));
    intentToController.putExtra(DATA_KEY,intent.getStringExtra(DATA_KEY));
    context.sendBroadcast(intentToController);

}
```

- **L'ACTION_KEY est redirigée vers le « bon » contrôleur**
- **Discussion**

Cumul du comportement



- **Avec une sous classe de ItemsController: ItemsControllerPlus**

```
@Override
onReceive(Context context, Intent intent){
  super.onReceive(context, intent);
  …
  …
```

Annexes

L'intent peut contenir des paramètres

- **Les extras, un *Bundle*, une *Map* !**
  - **Une table de couples <clé, valeur>, la clé est de type String**

```
Intent i = new Intent();
// i.setAction…
i.putExtra("fichier","hello.mp3");
i.putExtra("compteur",2);
```

Intent	putExtra (String name, double[] value) Add extended data to the intent.
Intent	putExtra (String name, int value) Add extended data to the intent.
Intent	putExtra (String name, CharSequence value) Add extended data to the intent.
Intent	putExtra (String name, char value) Add extended data to the intent.
Intent	putExtra (String name, Bundle value) Add extended data to the intent.
Intent	putExtra (String name, Parcelable[] value) Add extended data to the intent.
Intent	putExtra (String name, Serializable value) Add extended data to the intent.

Des paramètres à l'intention de

L'activité lit les paramètres transmis

- **Les extras, un *Bundle*, une *Map* !**

    Une table de couples <clé, valeur>, la clé est de type String

```
Intent i = getIntent();
String f = i.getStringExtra("fichier");
```

double	getDoubleExtra(String name, double defaultValue) Retrieve extended data from the intent.
Bundle	getExtras() Retrieves a map of extended data from the intent.
int	getFlags() Retrieve any special flags associated with this intent.
float[]	getFloatArrayExtra(String name) Retrieve extended data from the intent.
float	getFloatExtra(String name, float defaultValue) Retrieve extended data from the intent.
int[]	getIntArrayExtra(String name) Retrieve extended data from the intent.
int	getIntExtra(String name, int defaultValue) Retrieve extended data from the intent.

Des paramètres reçus par l'activité sélectionnée

putExtra, getExtras

Intent	putExtra (String name, Bundle value) Add extended data to the intent.
Intent	putExtra (String name, Parcelable[] value) Add extended data to the intent.
Intent	putExtra (String name, Serializable value) Add extended data to the intent.
Intent	putExtra (String name, int[] value) Add extended data to the intent.
Intent	putExtra (String name, float value) Add extended data to the intent.
Intent	putExtra (String name, byte[] value) Add extended data to the intent.
Intent	putExtra (String name, long[] value) Add extended data to the intent.
Intent	putExtra (String name, Parcelable value) Add extended data to the intent.

Bundle	getExtras () Retrieves a map of extended data from the intent.
int	getFlags () Retrieve any special flags associated with this intent.
float[]	getFloatArrayExtra (String name) Retrieve extended data from the intent.
float	getFloatExtra (String name, float defaultValue) Retrieve extended data from the intent.
int[]	getIntArrayExtra (String name) Retrieve extended data from the intent.
int	getIntExtra (String name, int defaultValue) Retrieve extended data from the intent.
ArrayList<Integer>	getIntegerArrayListExtra (String name) Retrieve extended data from the intent.

http://developer.android.com/reference/android/content/Intent.html

Types simples : c'est prévu

Objets métiers : ils ***implement***eront Parcelable

L'activité transmet une instance complète : Parcelable

- **La classe des instances transmises implémente Parcelable**
- **Parcelable comme Serializable**
  - **Sauf que tout est à la charge du programmeur**
  - **Chaque champ doit être écrit et restitué**
- **Exemple :**
  - **Auditeur est une classe qui implémente Parcelable**
    - **Les noms de méthodes sont imposées via l'interface**
- **Envoi depuis l'activité a1**

```
Auditeur unAuditeur = new Auditeur("alfred");
intent.putExtra("auditeur", unAuditeur);
startActivity(intent);
```

- **Appel**
  - **du constructeur**
  - **Lors de la transmission**
    - **writeToParcel**

```
public class Auditeur implements Parcelable {
   private String nom;
   private long   id;

   public Auditeur(String nom){
       this.nom = nom;
       this.id = globalId++;
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(this.nom);
        dest.writeLong(this.id);
    }
```

L'activité reçoit une instance Parcelable

- **Restitution :**
  - Classe Auditeur suite

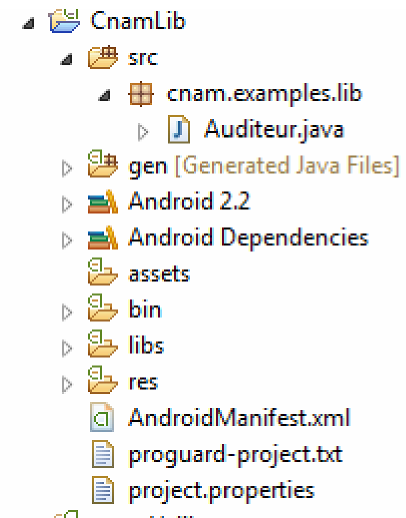
```
public static final Parcelable.Creator<Auditeur> CREATOR
        = new Parcelable.Creator<Auditeur>() {
    public Auditeur createFromParcel(Parcel in) {
        return new Auditeur(in);
    }

    public Auditeur[] newArray(int size) {
        return new Auditeur[size];
    }
};

private Auditeur(Parcel in) {
    this.nom = in.readString();
    this.id = in.readLong();
}
```

- Les noms de méthodes sont imposées via l'interface
- Un constructeur avec en paramètre l'instance Parcel reçue
  - Déclenché par le mécanisme de restitution

- **La classe *Auditeur* se doit d'être dans une librairie**
  - (afin d'éviter la recopie multiple des sources)

Une librairie, un .jar

- **Connaissance des classes par les deux activités ?**
  - **Toutes ces classes sont dans un projet eclipse : case isLibrary à cocher**
  - **Une librairie, un .jar à installer dans chaque application cliente**
  - **CnamLib contient la classe Auditeur ci-dessous**



Serializable / Parcelable, discussions

- **Serializable**
  - **Puissant mais lent, dû à l'usage de l'introspection**
  - ***Comically slow …***
- **Parcelable**
  - **Pas d'introspection, tout est défini par l'utilisateur**
  - ***Fanstastically quick …***
- **http://stackoverflow.com/questions/5550670/benefit-of-using-parcelable-instead-of-serializing-object**

`Serializable` is comically slow on Android. Borderline useless in many cases in fact.

`Parcel` and `Parcelable` are fantastically quick, but its documentation says you must not use it for general-purpose serialization to storage, since the implementation varies with different versions of Android (i.e. an OS update could break an app which relied on it).

- **Benchmark intéressant …**
  - **http://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking**
    - **Java Serializable, Externalizable, JSON …**