2010 - 2011

Alexandre BOSLÉ
+00 33 (0)6 82 56 24 96
Erwan JEGOUIC
+00 33 (0)6 76 55 21 64
Abderahman KRIOUILE
+00 33 (0)6 13 15 33 81
Marcel Cyrille LAMENU
+00 33 (0)6 46 71 30 46

Tél. : 01 34 94 83 60

Projet Industriel
E.S.I.A.L. - EBP
Tableau de bord mobile

[ANNEXE TECHNIQUE]

Ce document rassemble l'ensemble des spécifications techniques relatives aux phases de conception et de développement, et dont seuls l'utilisation et les effets résultants sont abordés dans le rapport final.
Ici, des détails de l'architecture des classes et des méthodes mises à contribution sont fournis.

ARCHITECTURE MODÈLE / VUE / CONTRÔLEUR

Nous avions initialement élaboré une architecture de classes selon l'organisation **Modèle / Vue / Contrôleur**, de manière à disposer d'une base formelle nous guidant dans la construction de l'application, et développant davantage les aspects structurels abordés par les diagrammes issus de la phase de conception.

Cette architecture ayant progressivement évolué au cours de la phase de développement, la composition finale du programme diffère de nos considérations d'origine.

Nous avons néanmoins fait le choix d'inclure les diagrammes de classes ci-dessous à titre informatif, dans la mesure où ils fournissent un aperçu schématique du fonctionnement de l'application :

Le modèle M.V.C.

L'architecture *Modèle / Vue / Contrôleur* est une architecture, et une méthode de conception qui organise l'Interface Homme-Machine d'une application logicielle ; une façon d'organiser une interface graphique d'un programme. Elle consiste à distinguer trois entités distinctes qui sont : le *modèle*, la *vue* et le *contrôleur* ayant chacun un rôle précis dans l'interface.

L'organisation globale d'une interface graphique est souvent délicate. Bien que le concept M.V.C. d'organiser une interface ne soit pas la solution miracle, elle fournit souvent une première approche qui peut ensuite être adaptée. Elle offre aussi un cadre pour structurer une application.

Dans l'architecture M.V.C., les rôles des trois entités sont les suivants :

- Modèle : données (accès et mise à jour)
- Vue : interface utilisateur (entrées et sorties)
- Contrôleur : gestion des événements et synchronisation

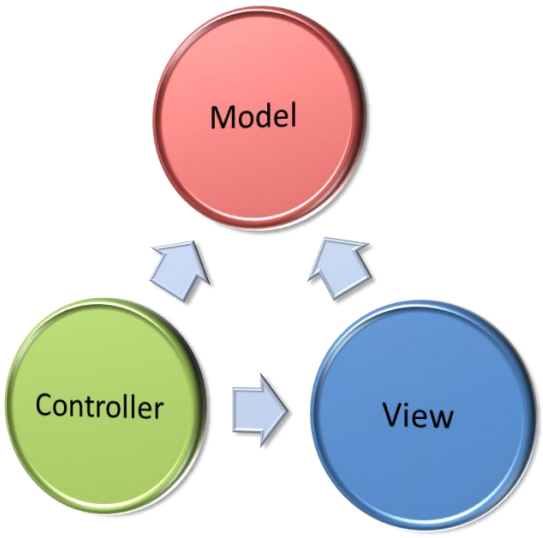

***Figure 1** : Architecture M.V.C.*

On obtient les diagrammes suivants avec le rôle de chacun d'entre eux :

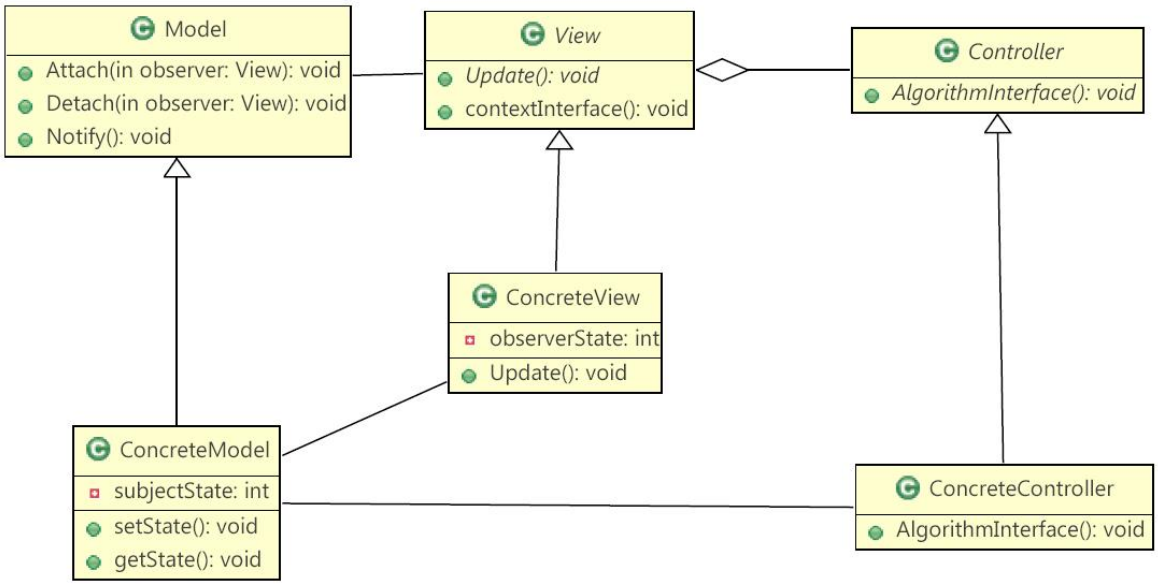

***Figure 2** : Diagramme de classes : Modèle - Vue - Contrôleur*

Rôle du contrôleur

Le contrôleur est chargé de la synchronisation du modèle et de la vue. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer. Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle et avertit ensuite la vue que les données ont changé, pour que celles-ci soient mises à jour. Certains événements de l'utilisateur ne concernent pas les données mais la vue. Dans ce cas, le contrôleur demande à la vue de se modifier.

Dans le cas de notre étude, une action de l'utilisateur peut aussi consister en la sélection d'un service pour y visualiser les informations. Ceci ne modifie pas le contenu du fichier X.M.L. mais nécessite simplement que la vue s'adapte et offre à l'utilisateur une vision des informations de ce service.

Le contrôleur est souvent scindé en plusieurs parties dont chacune reçoit les événements d'une partie des composants. En effet, si un même objet reçoit les événements de tous les composants, il lui faut déterminer l'origine de chaque événement. Ce tri des événements peut s'avérer fastidieux et peut conduire à un code peu élégant, ce pourquoi on évite le problème en répartissant le contrôleur en plusieurs objets.

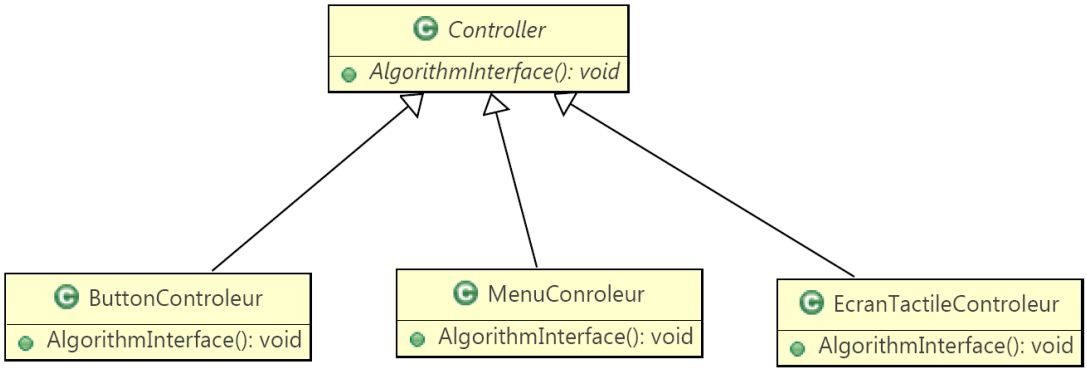

***Figure 3** : Diagramme de classes : Partie Contrôleur*

Rôle de la vue

La vue fait l'interface avec l'utilisateur. Sa première tâche est d'afficher les données qu'elle a récupérées auprès du modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrées, action sur un bouton, …). Ces différents événements sont envoyés au contrôleur.

La vue peut aussi donner plusieurs vues, partielles ou non, des mêmes données. La vue peut aussi offrir la possibilité à l'utilisateur de changer de vue. L'application comporte, comme le montre le diagramme suivant, plusieurs vues différentes :

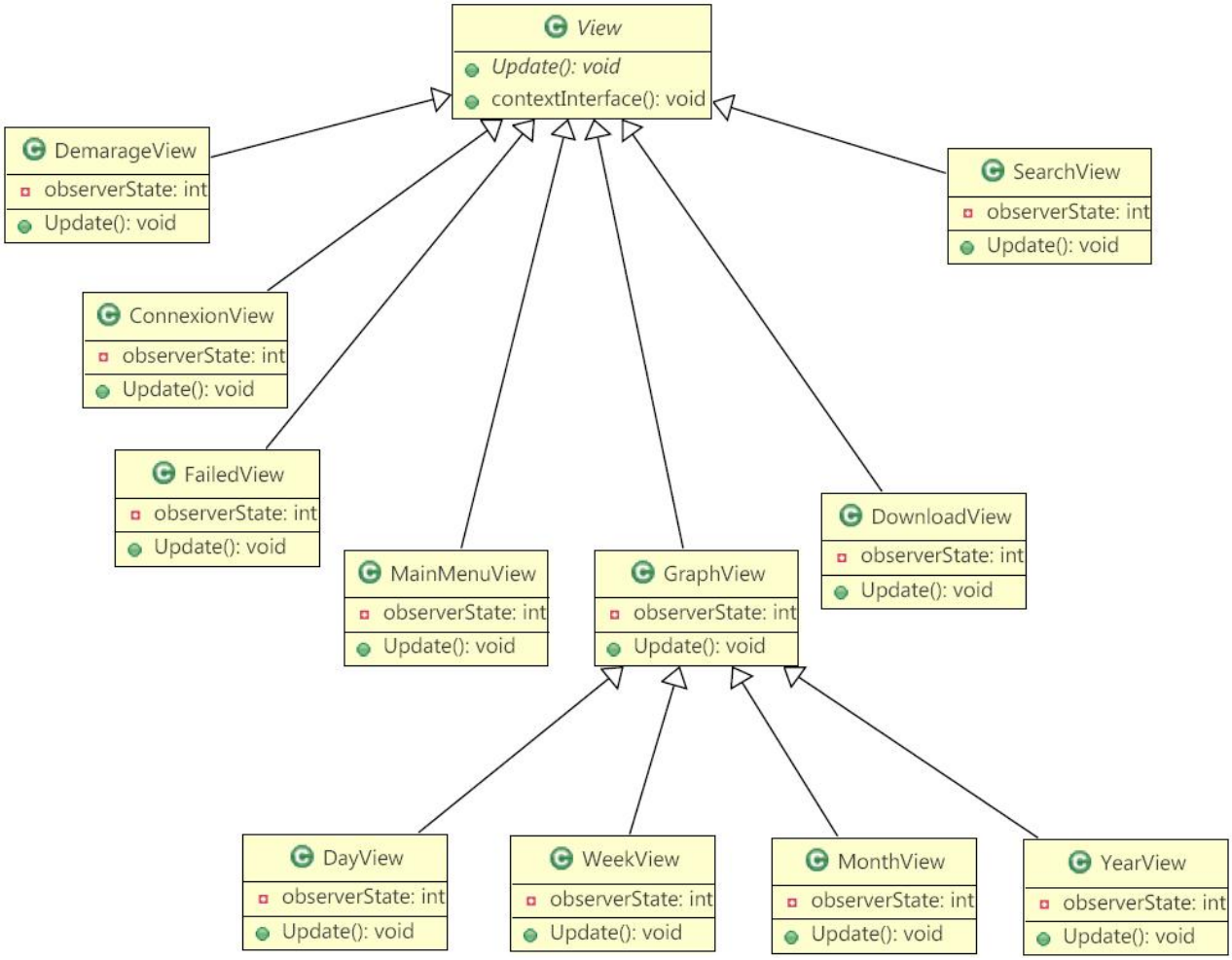

***Figure 4** : Diagramme de classes : Partie Vue*

Rôle du modèle

Le modèle contient les données manipulées par le programme. Il assure la gestion de ces données et garantit leur intégrité. Dans le cas typique d'un fichier X.M.L., c'est le modèle qui la contient.

Le modèle offre des méthodes pour mettre à jour ces données (insertion, suppression, changement de valeur, …). Il offre aussi des méthodes pour récupérer ses données. Dans le cas de données importantes, le modèle peut autoriser plusieurs vues partielles des données. Si par exemple le programme manipule un fichier X.M.L. comme dans notre cas, le modèle peut avoir des méthodes pour accéder à tous les services, toutes les informations d'un service particulier ou toutes les informations sur les utilisateurs du service.

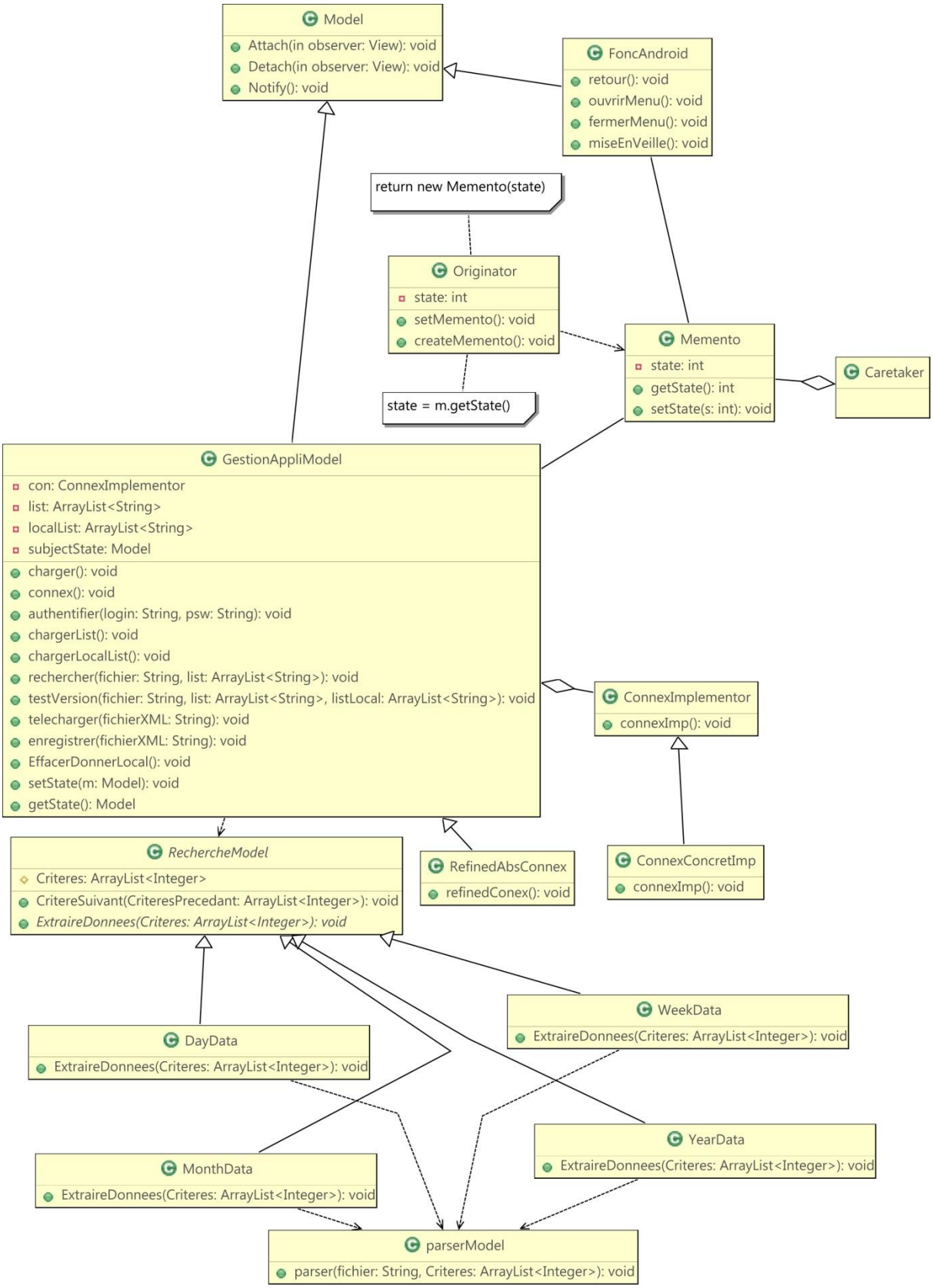

***Figure 5** : Diagramme de classes : Partie Modèle*

La classe GestionAppliModel comporte l'ensemble des fonctionnalités de l'application. Cette classe constitue le modèle de recherche des données (recherche par date, année, mois, jour).

À la lumière des diagrammes précédents et des diagrammes issus de notre phase de conception, nous avons produit la décomposition de l'application selon les classes suivantes :

1) Une classe dédiée au traitement des données obtenues depuis le service web.

2) Une classe chargée de la connexion de l'application au service web.
Elle nécessite en entrée les informations d'authentification (identifiant et mot de passe) qui devront être reconnus et validés par le serveur distant développé par EBP.

3) Une classe consacrée à l'affichage de données, qui, une fois l'utilisateur identifié auprès du service web, figurera la page d'accueil sur le *Smartphone*, afin de présenter les différentes statistiques disponibles pour l'utilisateur. Les classes subordonnées sont alors respectivement associées aux différents types de valeurs que pourra sélectionner l'utilisateur (chiffre d'affaires, quantité de stocks, achats, ventes, etc.).

4) Une classe qui va afficher les données sous formes de graphiques : une fois le choix d'une sous-classe correspondant à un type spécifique de données effectué, l'utilisateur devrait alors avoir des informations sur cette sous-classe. Par exemple, l'utilisateur pourra connaître le niveau d'un produit en stock. À cet effet, ayant choisi le stock, une certaine quantité d'informations lui est fournie par l'application (nom du produit, date de placement en stock, si possible) et il ne reste alors plus à l'utilisateur qu'à valider ses choix. Un graphique s'affiche ensuite, lui présentant la répartition du stock sur la période spécifiée.

5) Une classe chargée du traitement, ou *parsing*, du fichier X.M.L. : cette classe va nous permettre de lire les informations qui se trouvent dans le fichier X.M.L. et les traduire afin qu'ils puissent être utilisées en JAVA.

1) Les composants d'une application

Une application Android est un assemblage de composants liés grâce à un fichier de configuration.
Nous verrons comment les fichiers de configuration sont décrits et comment les composants interagissent entre eux.

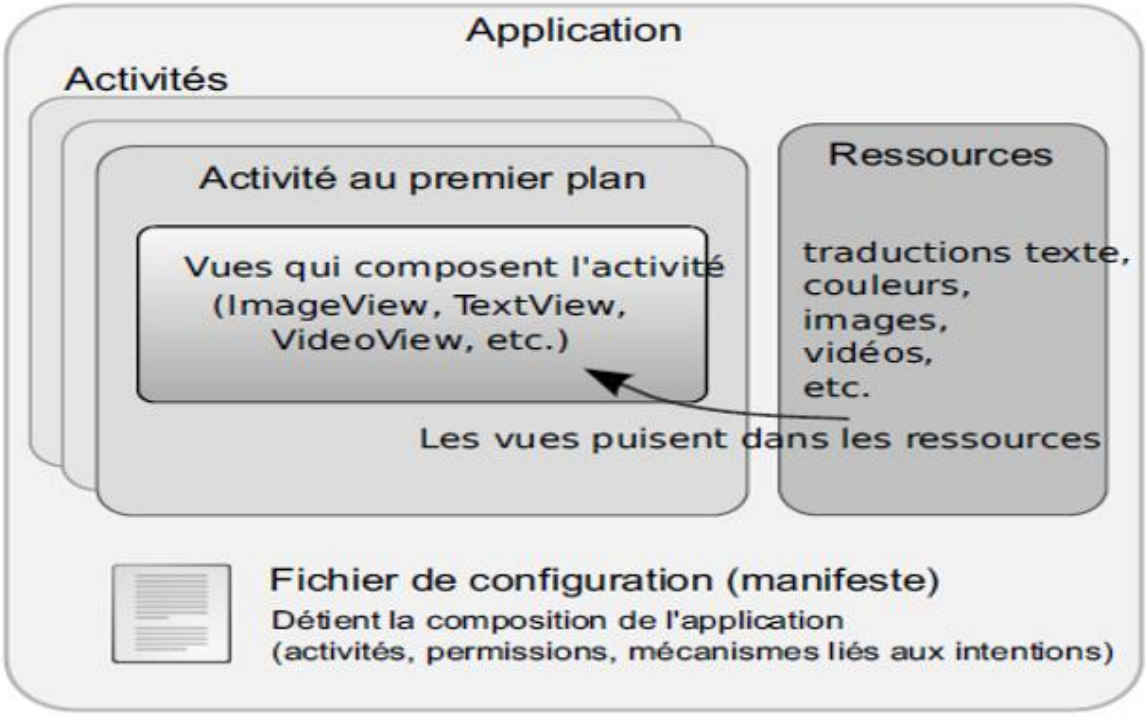

***Figure 6** : Composition d'une application Android*

Une application Android est composée de plusieurs éléments qu'il faut assembler. Nous pouvons donc citer les différents éléments suivants :

a) L'activité (*activity*) est la partie dédiée à la présentation du projet
b) Les services sont des opérations des applications effectuées en arrière-plan.
   Ce sont des C.R.O.N. qui tournent en fin de tâche et permettent ainsi la mise à jour des sources de données, et de déclencher des notifications aux composants les supervisant
c) Des fournisseurs de contenu (*content provider*) permettent le partage d'informations entre applications
d) *Object content* est une tâche ou une action spécifique
e) Les récepteurs d'intents (*broadcast receiver*) écoutent les autres applications afin de répondre aux intents
f) La notification

2) Le cycle de vie d'une application

Les applications Android ont un fonctionnement particulier : elles réagissent à des changements d'état imposé par le système (démarrage, pause, reprise, arrêt).

Cette séquence d'états peut s'illustrer à travers la figure suivante :

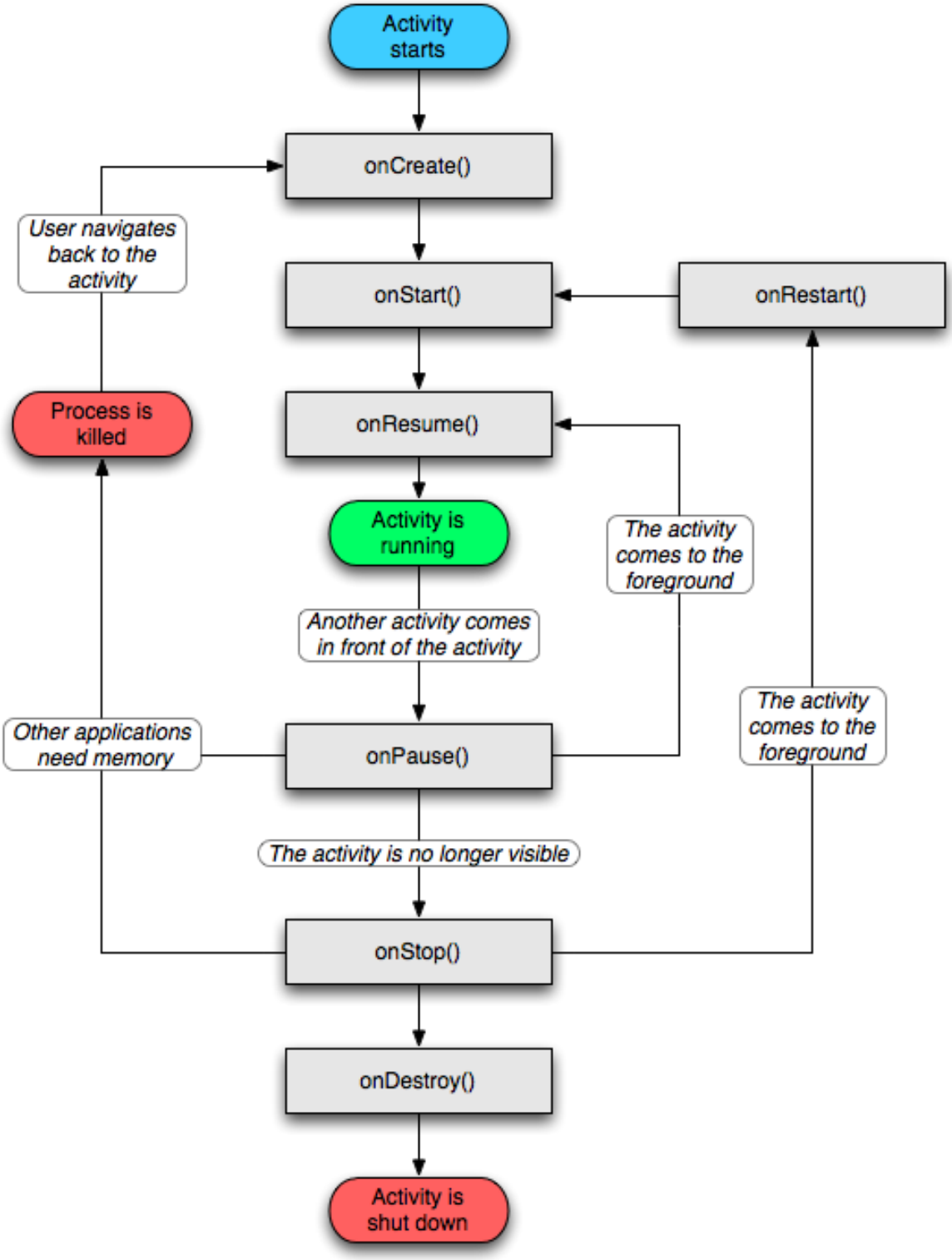

***Figure 7** : Cycle de vie d'une application Android*

Les différentes fonctions élémentaires auxquelles recourt l'application Android sont caractérisées comme suit :

onCreate() :

- S'exécute lorsque l'on lance l'application pour la première fois
- Est utilisée pour l'initialisation de :
    - La vue X.M.L.
    - Les fichiers ou données temporaires

onStart() :

- S'exécute suite à chaque appel à onCreate(), ou onRestart()
- Lance le chargement des données sauvegardées depuis le dernier arrêt de l'application

onRestart() :

- S'exécute au réveil de l'application, après un onStop()
- Redémarre, c'est-à-dire fait repasser l'application au premier plan

onResume() :

- S'exécute après chaque onStart()
- S'exécute à chaque passage au premier plan de l'activité
- Est utilisée pour l'initialisation :
    - La connexion à la base de données
    - La mise à jour des données qui auraient pu être modifiées entre temps, c'est-à-dire avant l'appel à onResume()

onPause() :

- S'exécute avant chaque onStop()
- S'exécute à chaque fois que l'utilisateur passe à une autre activité, ou qu'il fait appel à une terminaison de cette activité, ou encore lorsque le système a besoin de libérer de la mémoire pour de nouvelles ressources
- Est utilisée pour la libération de la mémoire, afin de permettre :
    - La sauvegarde des données qui seront perdues après l'arrêt si elles ne sont pas sauvegardées
    - La connexion à la base de données

onStop() :

- S'exécute avant chaque mise en sommeil de l'application

onDestroy() :

- S'exécute lors de l'arrêt de l'activité
- onCreate() devra être à nouveau exécuté pour récupérer à nouveau l'activité
- Est utilisée pour la libération des ressources, en l'occurrence, les fichiers temporaires