**Madi Anthony**
**N° 2760857**

**Leffad Malik**
**N°2965778**

**Mohammad Kabir**
**N°3361449**

# Projet PSTL
# Implémentation
# d'une famille d'application mobiles

**Master STL 1ere année**
**Promotion 2013-2014**

**Tewfik Ziadi**

# Sommaire

# I. Introduction.

Il arrive souvent quand nous développons une application mobile de vouloir implémenter plusieurs versions en fonction de plusieurs facteurs de variabilité :

- Implémenter une version complète .
- une version restreinte à seulement quelques fonctionnalités
- une version dans une autre langue
- une version plus adaptée à un modèle de téléphones
- etc...

La gestion de cette variabilité devient centrale dans ce contexte d'application mobile. Nous utilisons dans ce projet , l'environnement Feature IDE , qui est intégré à Eclipse et qui nous propose un cadre permettant d'implémenter ce qu'on nomme les "Lignes de Produits".

Une Ligne de Produits est une famille de logiciels partageant un ensemble de propriétés communes, satisfaisant des besoins spécifiques pour un domaine particulier et développée sur base d'un ensemble de composants clefs à l'aide de méthodes, outils et techniques de génie logiciel, et qui diffèrent par d'autres caractéristiques . une caractéristique d'un logiciel est définie comme importante pour distinguer les différents produits .

Nous allons voir ici comment s'est déroulé l'implémentation de ce concept au travers du développement d'une application Android de gestion de conférence.

# II. Conception.

## 1. Description et objectifs du projet.

Le projet consiste en la création d'une application mobile sous Android , permettant à l'utilisateur de gérer son emploi du temps par rapport à une conférence . Un utilisateur admin prendra le soin de créer une base de données des conférences , tandis qu'un utilisateur lambda pourra récupérer ces fichiers conférences sur son téléphone , et gérer selon ses envies les présentations auquelles il aura envie de participer.

L'application complète aura donc deux modes :

- Le mode « simple utilisateur » : qui permettra à l'utilisateur de récupérer des conférences , de choisir dans chaque conférence une présentation , ainsi de voir le lieu et la date de celle-ci , et de pouvoir ajouter à son agenda du téléphone la conférence à laquelle il voudra participer.

Il pourra gérer son emploi du temps des conférences , en cliquant sur le bouton calendrier.
Il pourra choisir de changer la langue de l'application dans les options.
Un clic sur le bouton « Conférence à Proximité » lui donnera le lieu de la conférence qui se trouve dans les environs .

- Le mode Administrateur : dans ce mode , l'utilisateur pourra non seulement récupérer des conférences , mais aussi en créer , en cliquant sur le bouton créer conférence , il aura ainsi le choix entre la créer manuellement et la créer à partir d'un fichier déjà prêt qu'une âme généreuse a pris le soin de préparer .

## 2. Les concepts liés au projet

### Ligne de produits logiciels et FeatureIDE

Une ligne de produits logiciels (LDP) est un ensemble de systèmes logiciels partageant un ensemble de propriétés communes, satisfaisant des besoins spécifiques pour un domaine particulier et développée sur une base d'un ensemble de composants clefs à l'aide de méthodes, outils et techniques de génie logiciel.

Un problème majeur de l'ingénierie de la famille des LDP est toujours le support d'outil manquant. La vision est un environnement de développement qui apporte toutes les phases du processus de développement ensemble, de manière cohérente et conviviale. FeatureIDE

est une de ces environnements de développement qui permet de réaliser cette tâche.

FeatureIDE est un plug-in d'Eclipse open source développé par l'Université de Magdeburg en Allemagne, qui aborde bien la problématique du développement de LDP. Elle est basée sur Eclipse et supporte toutes les phases de développement de logiciels fonctionnalité orientée pour le développement des LDP: analyse de domaine, la mise en œuvre de domaine, l'analyse des besoins, et la production de logiciels. Des techniques différentes de mise en œuvre d'un LDP sont intégrées tels que la programmation orientée feature, la programmation orientée aspect, la programmation orientée delta et préprocesseurs.

La technique de FeatureIDE utilisé dans notre projet est le celle de la programmation orientée feature (feature-oriented programming en anglais). Cette fonctionnalité intègre :

1. AHEAD
2. FeatureC++
3. FeatureHouse (avec support pour C, C#, Java 1.5, JML, Haskell, XML, JavaCC,…)

Parmi les ces intégrations, c'est l'intégration FeatureHouse qui nous intéresse le plus.

**FeatureHouse**

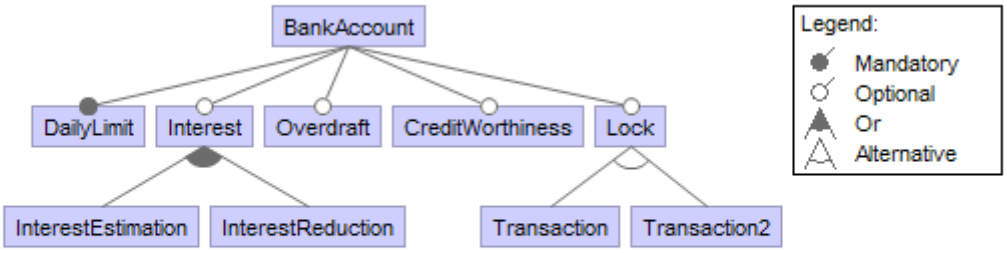
FeatureHouse est une approche générale de la composition des artefacts logiciels. C'est un mécanisme de dérivation automatique de code basé sur le raffinement de code. Elle est indépendante du langage de programmation en ce que les artefacts logiciels écrits en plusieurs langues peuvent être composé, par exemple, le code source, les cas de test, les modèles, la documentation et les fichiers makefile. Artefacts logiciels sont représentés comme un Feature Model.

Les Feature Models capturent l'essence de la structure modulaire d'un artefact sous la forme d'un arbre. Elles sont conçues pour représenter tout type d'artefact avec une structure hiérarchique.

Par exemple, un artefact écrit en Java contient les packages, classes, méthodes, etc., qui sont représentés par les nœuds de la Feature Model. Un document XML (par exemple, XHTML) peut contenir des éléments qui représentent la structure du document sous-jacent, par exemple, en-têtes, sections, paragraphes. Un makefile ou script de compilation se compose

de définitions et de règles qui peuvent être imbriqués.

Un feature est une caractéristique d'un logiciel définie par les experts de domaine comme importante pour distinguer les différents produits.




**Feature Model BankAccount**

Le diagramme ci-dessus est un exemple d'un Feature Model. Les nœuds sont les features que le logiciel peut potentiellement contenir. Chaque nœud a une propriété représenté par les icones affiché dans la légende. Un nœud père peut contenir une ou plusieurs nœuds fils. En d'autres termes, ce feature père peut contenir d'autres sous-features.

Un Feature Model contient les propriétés suivantes:

- Un cercle noir foncé (Logique AND) : signifie que ce feature est obligatoire et sera présent dans le produit final, exemple, DailyLimit
- Un cercle blanc : signifie que ce feature est optionnel, exemple, Interest, Overdraft, CreditWorthiness, Lock.
- Un arc blanc (Logique OR) entre deux feature fils signifie que si leur feature père devient obligatoire, alors **au moins** une de ces features sera disponible dans le produit final, exemple, Transaction et Transaction2
- Un arc noir foncé (alternative – Logique XOR) entre deux feature fils signifie que si leur feature père devient obligatoire, alors **au plus** une de ces features sera disponible dans le produit final, exemple, InterestEstimation et InterestReduction

Afin de choisir quels features seront présent dans la version finale d'un produit logiciel, on a recours à un fichier de configuration. Un fichier de configuration permet de choisir les

features (optionnelles, alternatives) pour ensuite générer un produit spécifique à partir de la ligne de produits. En d'autres termes, un fichier de configuration est une instanciation d'un Feature Model et il correspond à un produit spécifique.

Les choix doivent respecter les contraintes de cohérence

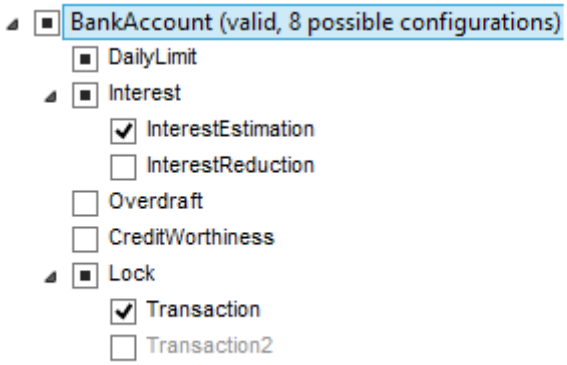



**Figure 1**

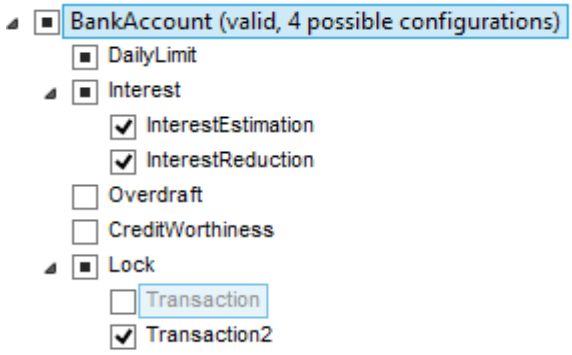



**Figure 2**

Le diagramme ci-dessus montre 2 fichiers de configuration possible pour BankAccount.

Dans le Figure 1 on peut remarquer que le feature « InterestEstimation » est sélectionnée et le feature « InterestReduction » ne l'est pas. Alors que dans la Figure 2 ces deux features sont sélectionnées (propriété cercle blanc - OR). Donc, le produit final pourra avoir soit le feature « InterestEstimation », soit le feature « InterestReduction », soit les deux features.

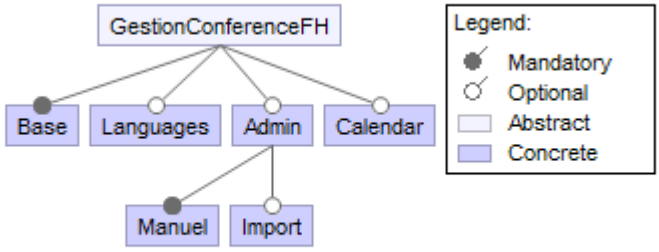
On peut aussi remarquer que dans la figure 1, « Transaction » est sélectionnée et « Transaction2 » est désactivé. Et dans la figure 2, « Transaction2 » est sélectionnée et « Transaction » est désactivé (propriété arc blanc – XOR). Donc, le produit final pourra avoir soit le feature « Transaction », soit le feature « Transaction2 » mais pas les deux.

Un LdP est développé avec FeatureHouse selon le model suivant :

- Un ensemble de fichiers de code source commun (DailyLimit)

- Chaque feature ajoute un raffinement (Interest, Overdraft…) :
    - Ajoutant des nouveaux fichiers de code source
    - Raffine les fichiers communs de base:
        - Ajouter des attributs
        - Ajouter des méthodes/fonctions
        - Modifier le code des méthodes existantes (surcharge)

**Feature Model de GestionConference**




Feature Model GestionConference

Le diagramme ci-dessus est le Feature Model de notre projet. Il contient les nœuds (features) suivants :

1. Base : Ce feature est obligatoire. Il contient des fonctionnalités de base de notre application. La fonctionnalité permet à l'utilisateur de consulter les conférences qu'il a créées. L'utilisateur ne peut rien fait d'autre.

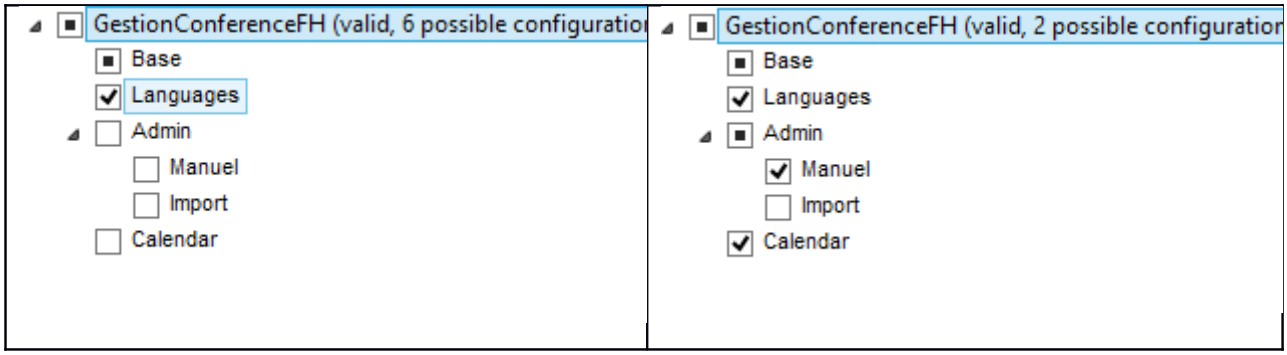
2. Languages : Ce feature est optionnel. Il permet à l'utilisateur de choisir une langue spécifique pour l'application

3. Admin : Ce feature est optionnel. Il permet à l'utilisateur de créer une conférence soit :

a. Manuellement (Manuel) : l'utilisateur rempli les champs nécessaires pour une conférence est ses présentations. Une fois rempli et valider, l'application créé un fichier XML et un fichier iCal.

b. En important un fichier iCal ou XML

4. Calendar : Ce feature est optionnel. Il permet à l'utilisateur d'ajouter une présentation d'une conférence à son calendrier. Il permet aussi à l'utilisateur de savoir pour un mois spécifique quelle présentation il doit assister.




**Configuration de GestionConference**

**Problèmes rencontrés avec FeatureIDE**

FeatureIDE supporte plusieurs langages de programmation dont Java. FeatureIDE connait toutes les classes de l'API Java. On pourra donc créer un projet Java en le combinant avec FeatureIDE sans problème.

Au contraire, FeatureIDE ne supporte pas le langage Android. Bien que Android soit principalement du Java, Android a ses propres classes qui ne peuvent pas être détecté par FeatureIDE. De plus, Android gère ses layouts avec des fichiers XML alors FeatureIDE ne gère que des fichiers Java (c'est possible de créer des layouts dynamiquement avec que du Java mais cette approche est fortement déconseillé).

Aussi, les structures de fichiers dans un projet Android sont complètement différentes de celle de FeatureIDE. C'est difficile voire impossible de mélanger un projet Android et FeatureIDE.

En conséquent, le développement d'un LdP Android dans un environnement FeatureIDE n'est pas évident.

Une solution utilisée est de créer deux projets différents. Un projet Android et un projet FeatureIDE.

Tous les fichiers XML, images etc. nécessaires pour créer les layout de l'application sont dans le projet Android.

Les fichiers java pour une version basique de l'application (Base) sont dans le projet FeatureIDE.

On ajoute d'autres classes Java propres aux autres features (Admin, Calendar etc…) au fur à mesure.

On choisit les features voulu dans le fichier de configuration. Le FeatureHouse raffine et génère les classes Java qui correspondent à la configuration choisit.

Enfin, on fait un copier-coller de ces fichiers Java dans le projet Android et on lance l'application.

Cette approche nous a  permis de générer un LdP pour notre application.

Un autre problème rencontré était de comprendre le fonctionnement de FeatureIDE. FeatureIDE se base sur le raffinement de code mais le raffinement est fait au niveau des méthodes de classe. Pour le moment FeatureIDE ne peut pas faire des raffinements au niveau des instructions de ces méthodes tel que les instructions sur une ligne, les instructions if…else, les instructions while etc.

Pour cela un FeatureIDE ne supporte pas bien les codes spaghetti (Un code spaghetti est un code peu clair et qui fait un usage excessif de sauts inconditionnels, d'exceptions en tous sens, de gestion des événements complexes et de threads divers). Ce qui n'est pas forcement mauvais car le code spaghetti n'est pas une bonne façon de programmer.

L'ajout d'un feature en FeatureIDE correspond à l'ajout des variables, la réutilisation de ces

variables dans d'autre class et l'ajout et/ou surcharge des méthodes des classes Java de l'application. Donc, l'utilisation de FeatureIDE ne favorise pas un code spaghetti.

La solution envisagée est de diviser notre code en plusieurs méthodes et de déclarer certaines variables dans ces méthodes comme attribut de la classe. Chaque méthode à une fonctionnalité bien définie. On pourra ajouter d'autres méthodes pour d'autre feature sans problèmes.

Par exemple au lieu d'avoir ce code :

```
public void ajouterListenerBoutons() {
  Button boutonOptions = (Button) findViewByID(R.id.boutonOptions);
  //instructions
  //instructions
  //instructions
  boutonOptions.setOnTouchListener(new View.OnTouchListener {
  @Override
  public boolean onTouch(View v, MotionEvent event) {
    int action = event.getActionMasked();
    if (action == MotionEvent.ACTION_DOWN) {
      v.setBackgroundResource(R.drawable.rounded_button_clicked);
    }
    if (action == MotionEvent.ACTION_UP) {
      v.setBackgroundResource(R.drawable.rounded_button);

      if (v == (Button) myFragmentView.findViewById(R.id.boutonOptions)) {
        Intent intent = new Intent(getActivity(), Option.class);
        startActivity(intent);
      }
    }
```

```
    return true;
  }});
```

Il faudra séparer le code :

```
  private Button boutonOption;


public void initialise() {
  boutonOptions = (Button) findViewByID(R.id.boutonOptions);
  //instructions
  //instructions
  //instructions
}
public void ajouterListenerBoutons() {
  initialise();
  //instructions
  //instructions
  //instructions
  boutonOptions.setOnTouchListener(new OptionClickListener());
}
public class OptionClickListener implements View.OnTouchListener {
  @Override
  public boolean onTouch(View v, MotionEvent event) {
    int action = event.getActionMasked();
    if (action == MotionEvent.ACTION_DOWN) {
```

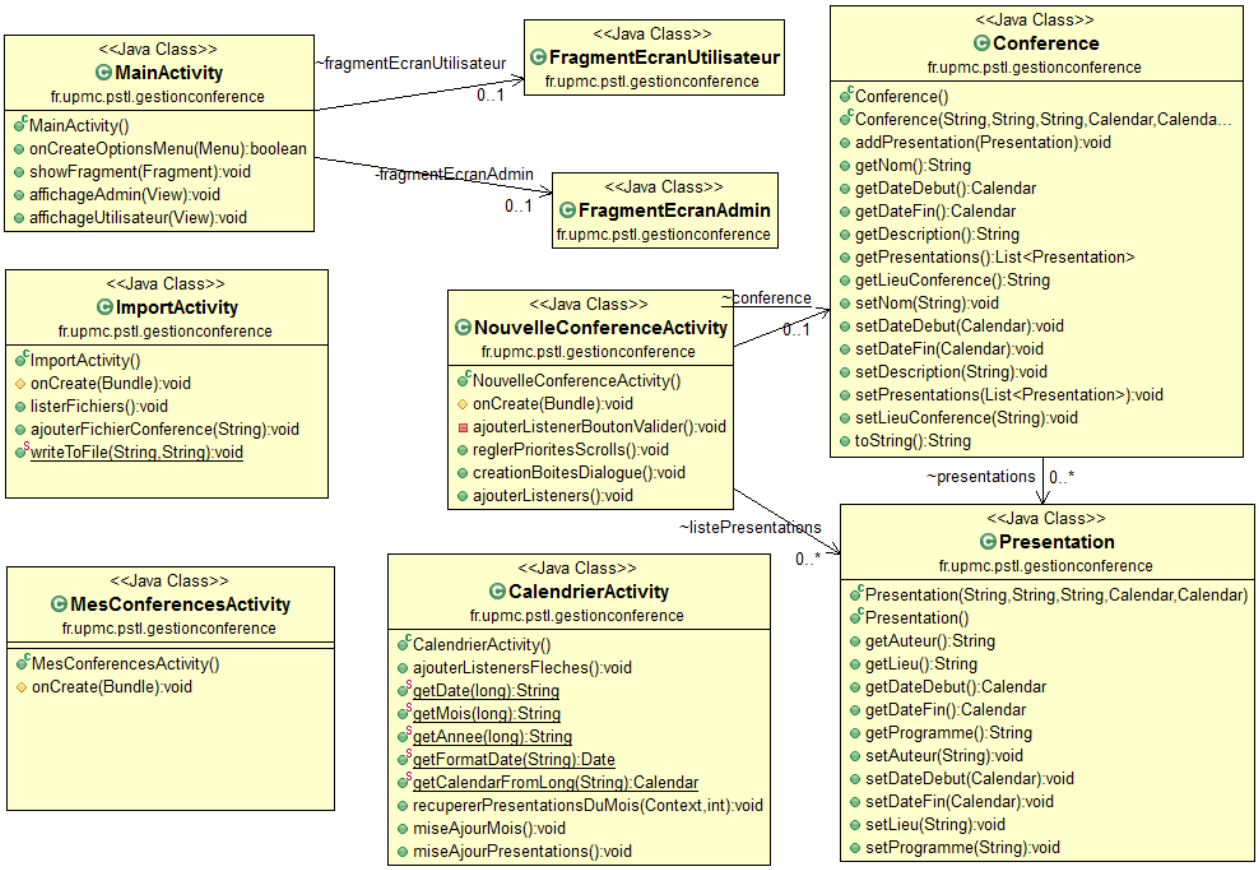
```
        v.setBackgroundResource(R.drawable.rounded_button_clicked);
      }
      if (action == MotionEvent.ACTION_UP) {
        v.setBackgroundResource(R.drawable.rounded_button);
        if (v == (Button) myFragmentView.findViewById(R.id.boutonOptions)) {
          Intent intent = new Intent(getActivity(), Option.class);
          startActivity(intent);
        }
      }
      return true;
    }

  }
```

La séparation de code ci-dessus est juste un moyen de le faire. Il n'y a pas un moyen global pour diviser le code. Tout dépend de la partie du code nécessite le raffinement.

## 3. Les classes principales.

Voici un diagramme de classes représentant les classes importantes :




Nous avons la classe **MainActivity** qui représente la page d'accueil et qui est lancé à chaque démarrage de l'application ,et qui contient deux fragments , un fragment « **FragmentEcranUtilisateur** » , là ou se trouvent les fonctionnalités dont a besoin un utilisateur simple , ainsi qu'un fragment « **FragmentEcranAdmin** » qui s'occupera des fonctionnalités relatives à l'utilisateur admin .

Nous avons décidé d'utiliser deux fragments dans cette partie car nous ne voulons pas créer une activité à chaque fois que l'utilisateur passe d'un mode à un autre , et aussi cela a aidé au niveau de la variabilité , lorsque nous n'avons pas besoin du mode admin.

L'objet Conférence :

Pour gérer une conférence , nous avons décidé de créer une classe Conférence qui possédera des attributs qui caractériseront une conférence , comme le nom , le lieu etc ... Cette classe Conférence possédera quant à elle , des présentations , chaque présentation aura elle aussi des attributs qui la caractériseront .

La classe **MesConférencesActivity** est l'activité qui affiche les conférences disponibles .

La classe **ImportActivity** est l'activité qui gère les imports de fichiers . Lorsque l'activité démarre , elle affiche des fichiers déjà crées , qui sont soit au format XML , soit au format ICS , ce dernier est un standard pour les échanges de données de calendrier qui est aussi connu sous le nom d'**iCal**.

Il est supporté par un grand nombre de logiciels, tels que iCal d'Apple , Chandler, Lotus ou encore Zimbra.

Quelque soit le fichier affiché , lorsqu'on décide de l'importer , celui-ci est converti en XML qui sera sur le téléphone , et qu'on pourra afficher en cliquant sur le bouton Mes Conférences .

La classe **CalendrierActivity** est l'activité qui permettra à l'utilisateur d'afficher la liste des présentations qui l'intéressent . En Cliquant sur les flèches , on fait défiler les mois , affichant ainsi les différentes présentations du mois .

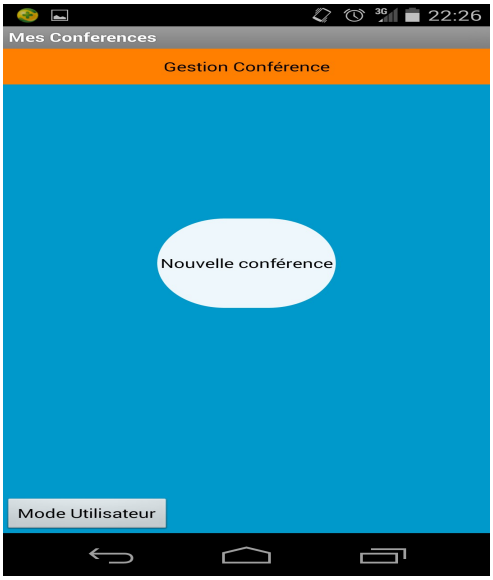
La classe **NouvelleConférenceActivity** est l'activité qui permettra de créer une conférence manuellement , en entrant directement les données sur le téléphone , comme le titre , le lieu , la description ou encore les différentes présentations .

Pour ajouter une présentation on clique sur le bouton « Ajouter Presentation » qui affiche une boite de dialogue qu'il faudra remplir . A la fin , lorsque toutes les données ont été correctement insérées , il faudra cliquer sur le bouton valider , qui créera un fichier au format XML , qu'on pourra par la suite consulter dans Mes Conférences .

## 4. Présentation de l'interface.

Au lancement de l'application , c'est l'écran utilisateur qui apparaît , celui-ci possède 5 boutons , 4 boutons ronds et un en bas à gauche pour passer au mode administrateur .


En cliquant sur le bouton Mode Admin , l'application bascule vers le mode administrateur qui permet à l'utilisateur de créer une nouvelle conférence en cliquant sur le bouton Nouvelle Conférence :



A partir du mode utilisateur simple , on peut accéder à la partie Mes Conférences , qui nous affiche une liste de toutes les conférences qui sont disponibles :


On peut voir les détails de la conférence simplement en cliquant sur l'une d'elles , et ainsi voir les différentes présentations :

Plusieurs présentations sont disponibles , il suffit de cliquer sur l'une d'elles , pour voir les détails tels que les horaires et le lieu , et on peut choisir d'ajouter cette présentation à notre agenda en cliquant sur le bouton en bas.

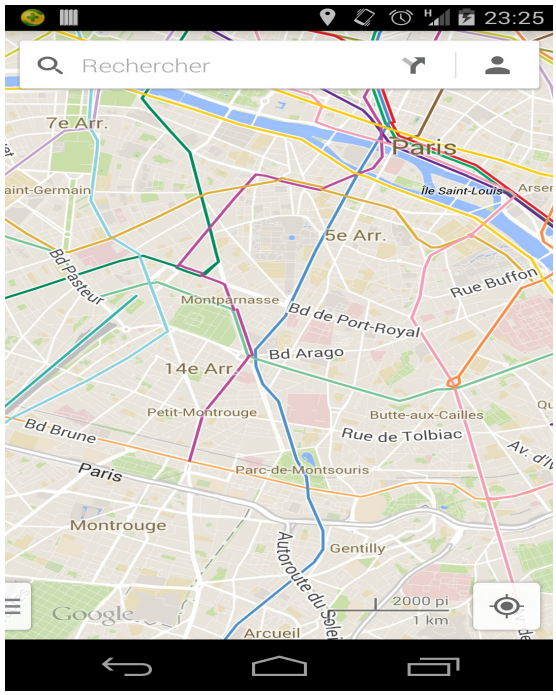

On peut gérer les différentes présentations enregistrées sur l'agenda en cliquant sur le bouton Calendrier, et ainsi voir l'emploi du temps :

A partir du mode administrateur , l'utilisateur peut créer une conférence en cliquant sur le bouton central , il peut choisir de la créer manuellement , en entrant les données directement sur le téléphone :


Ou alors il peut décider d'importer des fichiers conférences déjà prêts :

L'utilisateur peut décider de changer la langue de l'application dans les options :


Et enfin , un clic sur les conférences à proximité mènera vers l'application Google Maps du téléphone :



# III. Implémentation et réalisation.

## 1. Gestion du XML

Une conférence est représentée par son nom, sa date (date du début de la conférence et date de la fin de la conférence), ses présentations et son lieu. Une conférence peut contenir une ou plusieurs présentations. Le modèle XML ressemble au code XML ci-dessous :

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<Conference>
        <nom> </nom>  <!-- le nom de la conférence -->
        <datedebut> </datedebut> <!—la date du début de la conférence -->
        <datefin> </datefin>  <!—la date du fin de la conférence -->
        <description> </description>  <!—la description de la conférence -->
        <Presentations>
                <!—les presentations de la conférence -->
        </Presentations>
        <lieuConference> </lieuConference>  <!-- le lieu de la conférence -->
</Conference>
```

Une présentation est représentée par son auteur, sa date (date et heure pour le début et la fin de la présentation), son lieu et son programme. Cette présentation sera représentée en XML avec la balise « presentation ».

La balise « Presentations » dans le figure ci-dessus contiendra donc une ou plusieurs balises « presentation ».

***<u>Remarque</u> : Il y a une balise « Presentations » qui commence une majuscule et termine avec « s ». Cette balise contient des balises « presentation » (tout en minuscule sans « s »).***

```
<Presentations>
        <presentation>
                <auteur> </auteur> <!—l'auteur de la présentation -->

                <!—l'heure début de la présentation -->
                <datedebutPresentation> </datedebutPresentation>

                <!—l'heure début de la présentation -->
                <datefinPresentation> </datefinPresentation>
                <lieuPresentation> </lieuPresentation><!-- le lieu de la conférence -->
                <programme> </programme> <!-- le programme de la présentation -->
        </presentation>
</Presentations>
```

Enfin un fichier XML d'une conférence créée par utilisateur ressemble à la figure ci-dessous :

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<Conference>
        <nom>Nuclear power</nom>
        <datedebut>12/03/2014</datedebut>
        <datefin>30/03/2014</datefin>
        <description>
        Talking about nuclear power. Should we continue or should we start
        thinking about renewable energy sources.
```

```
        </description>
        <Presentations>
                <presentation>
                        <auteur>Martin Scorsese</auteur>
                        <datedebutPresentation>10/03/2014 16:05</datedebutPresentation>
                        <datefinPresentation>10/03/2014 18:06</datefinPresentation>
                        <lieuPresentation>Chicago</lieuPresentation>
                        <programme>Nuclear power</programme>
                </presentation>
                <presentation>
                        <auteur>Martin Scorsese</auteur>
                        <datedebutPresentation>17/03/2014 04:05</datedebutPresentation>
                        <datefinPresentation>17/03/2014 06:10</datefinPresentation>
                        <lieuPresentation>Illinois</lieuPresentation>
                        <programme>Renewable Energy</programme>
                </presentation>

                <presentation>
                        <auteur>Bradley Cooper</auteur>
                        <datedebutPresentation>24/01/2014 20:05</datedebutPresentation>
                        <datefinPresentation>24/01/2014 21:10</datefinPresentation>
                        <lieuPresentation>San Diego</lieuPresentation>
                        <programme>Wind Power</programme>
                </presentation>
        </Presentations>
        <lieuConference>United States of America</lieuConference>
</Conference>
```

Ce fichier est sauvegardé dans un dossier créé pendant le lancement de l'application.

Si l'utilisateur veut consulter ses conférences, l'application fera une analyse syntaxique des fichiers XML présent et affiche la conférence dans une interface graphique.

La classe XmlFileAnalyser permet d'écrire et d'analyser un fichier XML.

## 2. Gestion de calendrier iCal

Outre le traitement des fichiers XML, l'application permet d'importer et de générer des fichiers iCal, iCal est un logiciel de gestion de calendrier, développé par la société Apple. Cette fonctionnalité nous permet de pouvoir importer un grand nombre de conférences car beaucoup d'organismes qui organisent des conférences proposent des fichiers iCal au télechargement.

Voila un exemple de fichier iCal :

```
  BEGIN:VCALENDAR

PRODID:-//Events Calendar//iCal4j 1.0//EN

CALSCALE:GREGORIAN

VERSION:2.0

X-WR-CALNAME:Conference de la paix

X-LIC-LOCATION:Paris

X-WR-CALDESC:Comment sauver la Crimée

BEGIN:VEVENT

DTSTAMP:20140429T091015Z

DTSTART:20140402T200000

DTEND:20140402T210015

SUMMARY:La paix rédide dans le yoga

TZID:Europe/Paris

LOCATION:Paris Bercy

DESCRIPTION: Mr Freedom

UID:20140429T091017Z-uidGen@134.157.255.169

END:VEVENT

END:VCALENDAR
```

Les principaux élement d'un calendrier iCal sont les VEVENT qui représentent pour nous nos présentations. On trouve dans un VEVENT :

- Une date de début : DTSART
- Une date de fin : DTEND
- Un lieu : LOCATION
- Un auteur : DESCRIPTION
- Un nom : SUMMARY

Donc tous les éléments dont nous avons besoin pour enregistrer nos présentations. Afin de générer nos fichiers iCal on utilise la classe IcsManipulator qui nous permet de passer d'un objet conférence à un fichier iCal ou d'un fichier Ical à un objet conférence.

Afin de pouvoir parser un fichier iCal, on utilise une API qui se nomme cal4j, voilà par exemple comment se crée un calendrier iCal en fonction d'un objet conférence :

```
public static String generateIcsString(Conference conference) {
    net.fortuna.ical4j.model.Calendar icsCalendar ;
    icsCalendar = new net.fortuna.ical4j.model.Calendar();
    icsCalendar.getProperties().add(new ProdId("-//Events Calendar//iCal4j 1.0//EN"));
    icsCalendar.getProperties().add(CalScale.GREGORIAN);
    icsCalendar.getProperties().add(Version.VERSION_2_0);
    icsCalendar.getProperties().add(new XProperty("X-WR-CALNAME",conference.getNom()));
    icsCalendar.getProperties().add(new XProperty("X-LIC-LOCATION",conference.getLieuConference()));
    icsCalendar.getProperties().add(new XProperty("X-WR-CALDESC",conference.getDescription()));

    for(int i =0; i<conference.getPresentations().size();i++){

        TimeZoneRegistry registry = TimeZoneRegistryFactory.getInstance().createRegistry();
        TimeZone timezone = registry.getTimeZone("Europe/Paris");
        VTimeZone tz = timezone.getVTimeZone();

        java.util.Calendar startDate = new GregorianCalendar();
        startDate.setTimeZone(timezone);
        startDate.setTime(conference.getPresentations().get(i).getDateDebut().getTime());
        startDate.set(conference.getPresentations().get(i).getDateDebut().getTime().getYear(), confer

        java.util.Calendar endDate = new GregorianCalendar();
        endDate.setTimeZone(timezone);
        endDate.setTime(conference.getPresentations().get(i).getDateFin().getTime());
        endDate.set(conference.getPresentations().get(i).getDateFin().getTime().getYear(), conference

        DateTime start = new DateTime(startDate.getTime());
        DateTime end = new DateTime(endDate.getTime());
        VEvent meeting = new VEvent(start, end, conference.getPresentations().get(i).getProgramme());

        meeting.getProperties().add(tz.getTimeZoneId());
        meeting.getProperties().add(new Location(conference.getPresentations().get(i).getLieu()));
        meeting.getProperties().add(new Description(conference.getPresentations().get(i).getAuteur())
        icsCalendar.getComponents().add(meeting);
    }
    System.out.println(icsCalendar.toString());
    return icsCalendar.toString();
}
```

la première partie crée « la base » du calendrier, on enregistre les informations de la conférence. Ensuite vient la boucle for pour enregistrer chaque présentation sous la forme d'un VEVENT. Une fois le calendrier terminé on le retourne sous la forme d'un String qui peut alors être sauvegardé pour un usage ultérieur.

## 3. Problèmes rencontrés dans le développement de l'application

Une des problèmes rencontrés était au niveau des dates en Java. La classe java.util.Date en Java n'est pas très bien optimiser. On a tout d'abord décidé d'utiliser la bibliothèque Java open source qui s'appelle « Joda ». Cette bibliothèque représente bien les dates en Java mais quelques problèmes apparaissent quand on veut l'utiliser avec Android. On a fini par utiliser les classes GregorianCalendar.java et Calendar.java.

Un autre problème rencontré était au niveau de calendrier Android. Le smartphone Android d'un utilisateur peut contient une ou plusieurs calendriers ; le calendrier par défaut de Google, un calendrier pour les anniversaire, un calendrier pour les événements de Facebook ou Twitter, un calendrier Hotmail etc. Chaque calendrier à une id unique.

La question se pose alors : dans quel calendrier peut-on enregistrer un événement pour une présentation lorsque l'utilisateur décide de l'ajouter à son calendrier ?

On pourra l'enregistrer dans le calendrier par défaut de Google. Pour cela, au démarrage de l'application, on scanne la liste de calendriers disponible et récupère l'id du calendrier Google et une fois l'utilisateur clique sur le bouton « Ajouter au calendrier », on l'insère dans le calendrier Google.

Cette méthode marche mais le problème est que même si Android créé un calendrier par défaut Google pour l'utilisateur, l'utilisateur peut décider de supprimer ce calendrier pour une raison ou autre. Alors l'enregistrement échouera si c'est le cas.

La solution envisagée est de créer lors de démarrage de l'application un calendrier pour notre application dans le smartphone.

Le calendrier portera le nom de notre application pour que l'utilisateur se rende bien compte que ce calendrier est pour l'application. Bien sur l'utilisateur peut supprimer ce calendrier par erreur (ou intentionnellement).

On a donc utilisé cette approche pour le résoudre :

1. Lors du lancement de l'application, on cherche si le calendrier existe dans le smartphone

2. S'il n'existe pas on crée le calendrier, récupère l'id du calendrier et on le sauvegarde dans le SharedPreferences de l'application. SharedPreferences nous permet de stocker des données d'une application. Ils nous permettent de sauvegarder et de récupérer des données sous forme de clé-valeur.

3. Si l'utilisateur relance notre application, l'application cherche dans son SharedPreferences si l'id de calendrier existe. S'il existe alors il n'y a pas besoin de créer le calendrier.

4. Si l'utilisateur supprime le calendrier et relance notre application, l'application détecte que le calendrier a été supprimé. Il recréer le calendrier et mettre à jour le SharedPreferences de l'application. Cependant, l'utilisateur perd tous les données qui étaient stocké dans le calendrier.

Cette approche nous a permis de facilement récupérer les événements concernant notre application à partir d'un calendrier. Comme on a l'id du calendrier, au lieu de faire un parcours de tous le calendrier du smartphone, on pourra cibler un calendrier précis (le calendrier créer par l'application) et récupérer facilement ces événements.

# IV. Conclusion.

La conception de ce projet fut longue mais réellement intéressante dans le sens où nous avons découvert une nouvelle manière de coder, certe nous connaissions tous le langage JAVA mais coupler à Android et FeatureIDE ce n'était pas de tout repos mais instructif. Nous avons tenté de mettre en place une interface graphique intuitive et ergonomique afin de garantir la meilleure utilisation du programme possible.