INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

PROJET D'APPROFONDISSEMENT ET D'OUVERTURE

COMPAGNON VIRTUEL SOUS ANDROID

Institut national des sciences appliquées de Rouen

Rédigé par :
Jitao XU
Chenxin LU

Proposé par :
Alexandre PAUCHET

ASI
1er janvier 2018

Table des matières

Introduction

Dans le cadre de notre scolarité au sein du département Architecture des Systèmes d'Information (ASI) de l'Institut National des Sciences Appliquées (INSA) de Rouen, nous devons réaliser au cours de notre premier semestre de quatrième année un Projet d'Approfondissement et d'Ouverture (PAO), c'est-à-dire un travail sur plusieurs mois, seul ou en équipe, dans lequel nous pouvons mettre en pratique nos connaissances apprises en cours et/ou apprendre de nouvelles connaissances sur des sujets en rapport avec notre formation. Nous avons donc tous les deux décidé de nous intéresser à un sujet proposé par M. Alexandre Pauchet : la création d'un compagnon virtuel sous Android. Ce PAO permet d'une part de mettre en application nos acquis en Java de troisième année, et d'autre part de découvrir la conception d'une application mobile sous Android, chose totalement nouvelle pour nous deux.

Chapitre1

1 Analsye des besoins

1.1 Liste des besoins

Le but principal du projet est de développer des nouvelles fonctionnalités sur l'application Compagnon Virtuel déjà existante, notre verision sera version 4. Avant de commencer le projet, l'application est déjà fonctionnelle, elle propose une interaction avec un personnage animé. Ce dernier est capable d'échanger via reconnaissance vocale et synthèse vocale. L'analyse des requêtes est déportée sur un serveur externe de dialogue intelligent : pandorabots. Le compagnon virtuel s'exprime aussi via des animations. Pendant la préparation du projet, nous avons proposé de réaliser les fonctionnalités suivantes :

- effectuer une recherche dans googlemaps et afficher une carte dans l'application.
- envoyer sms.
- ajouter et supprimer un réveil.
- afficher un calendrier au sein de l'application
- intégrer le processus de gestion de calendrier au chatbot externe.
- afficher le prochain événement dans le calendrier, il est capable de mettre à jour le prochain événement s'il y a une modification dans le calendrier.

Autrement, nous avons proposé de améliorer l'interface d'utilisateurs de l'application car nous n'étions pas satisfaits avec l'interface d'utilisateur actuelle.

1.2 Rendus

En résumé, les rendus demandés sont :

- une application fonctionnelle sous Android,
- un rapport sur le déroulement du projet,
- une documentation du code source,
- un guide utilisateur,
- une vidéo pour faire une démonstration d'application.

Ils doivent être fournis avant la fin du premier semestre de ASI 4.1, afin d'être compabilisés dans la moyenne semestrielle.

Chapitre 2

2 Spécification

2.1 Faisabilité du projet

Dans un premier temps, nous avons suivi un cours sur www.udacity.com pour apprendre la base des connaissances de développer une application sous Android. Ce cours nous permet de développer une simple application, par exemple un compteur de points qui est utilisé dans le match du basket sous Android nous-même. Donc, nous pensons que nous sommes capable de commencer notre PAO.

Ensuite, nous avons effectué une brève étude bibliographique quant à la faisabilité du développent de fonctionnalités que nous avons proposé. Nous avons trouvé des APIs du développement sous Android. Dans ce cas là, nous pensions que tous les fonctionnalités pouvaient être développés sous Android sans trop de difficulté.

Concernant la partie de la réponse intelligente, comme les anciens PAOs sur ce sujet, nous avons décidé d'utiliser le Pandorabots, outils Internet permettant d'héberger un grand nombre de fichiers AIML contenant les question/réponses, pour nous aider de réaliser tous les fonctionnalités. Le choix du IDE a été fait rapidement, nous avons choisi Android Studio qui est un IDE très moderne.

Notre application peut être fonctionnée sur Android 4.4 ou supérieur avec une connexion Internet permanante.

2.2 Analyse Descendante

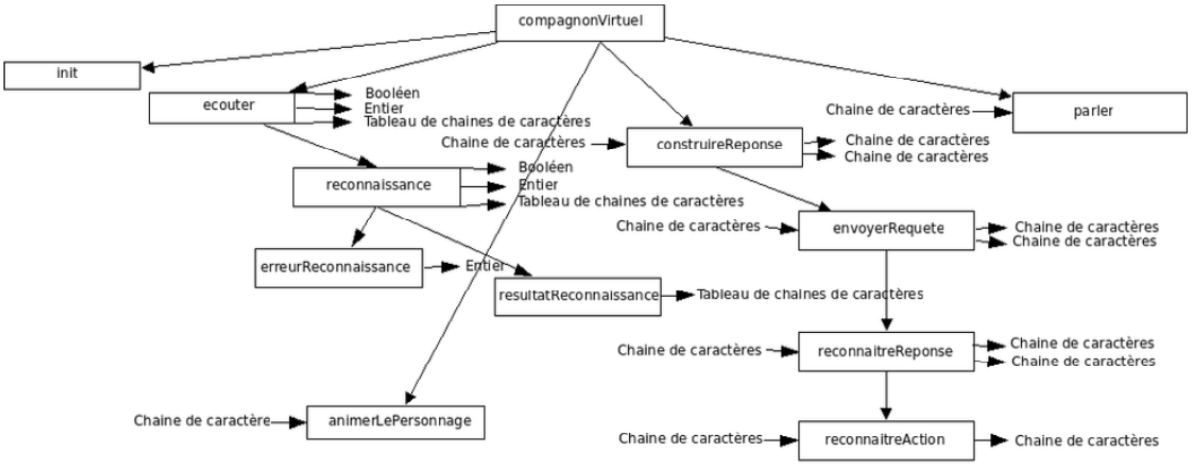

FIGURE 1 – Analyse descendante de la version ancienne.

L'analyse descendante a été décrit dans le rapport de 1ère version.

Chapitre 3

3 Conception Préliminaire

Cette étape nous permet, à partir des différents éléments de l'analyse, de mettre en forme les fonctions et procédures afin d'en expliciter les nouveaux fonctionnements.

3.1 Cas d'utilisation

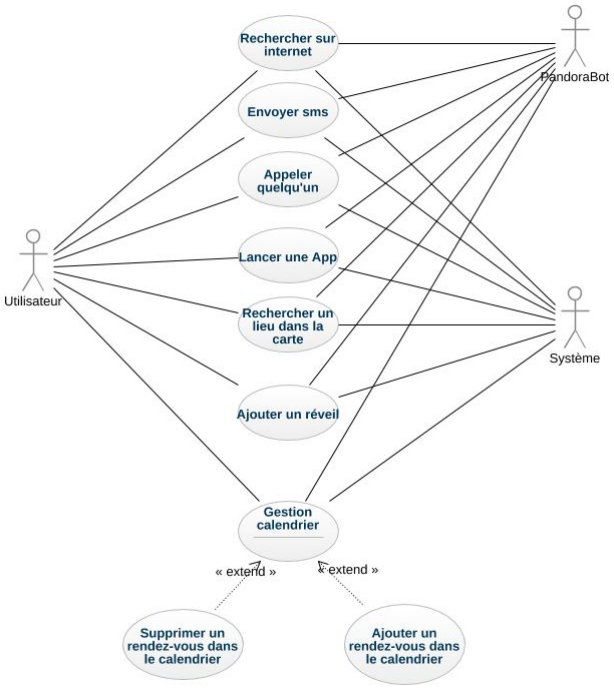

FIGURE 2 – Cas d'utilisation.

3.2 Diagrammes de Sequence

C'est un diagramme de sequence pour envoyer un sms. Les parties à la boite noire est la partie du code qui a été complété dans les dernières versions. Nous vous donnons ce diagramme comme un model de diagramme de sequence. Pour autres fonctions ou procédures, nous travaillons en même mode.

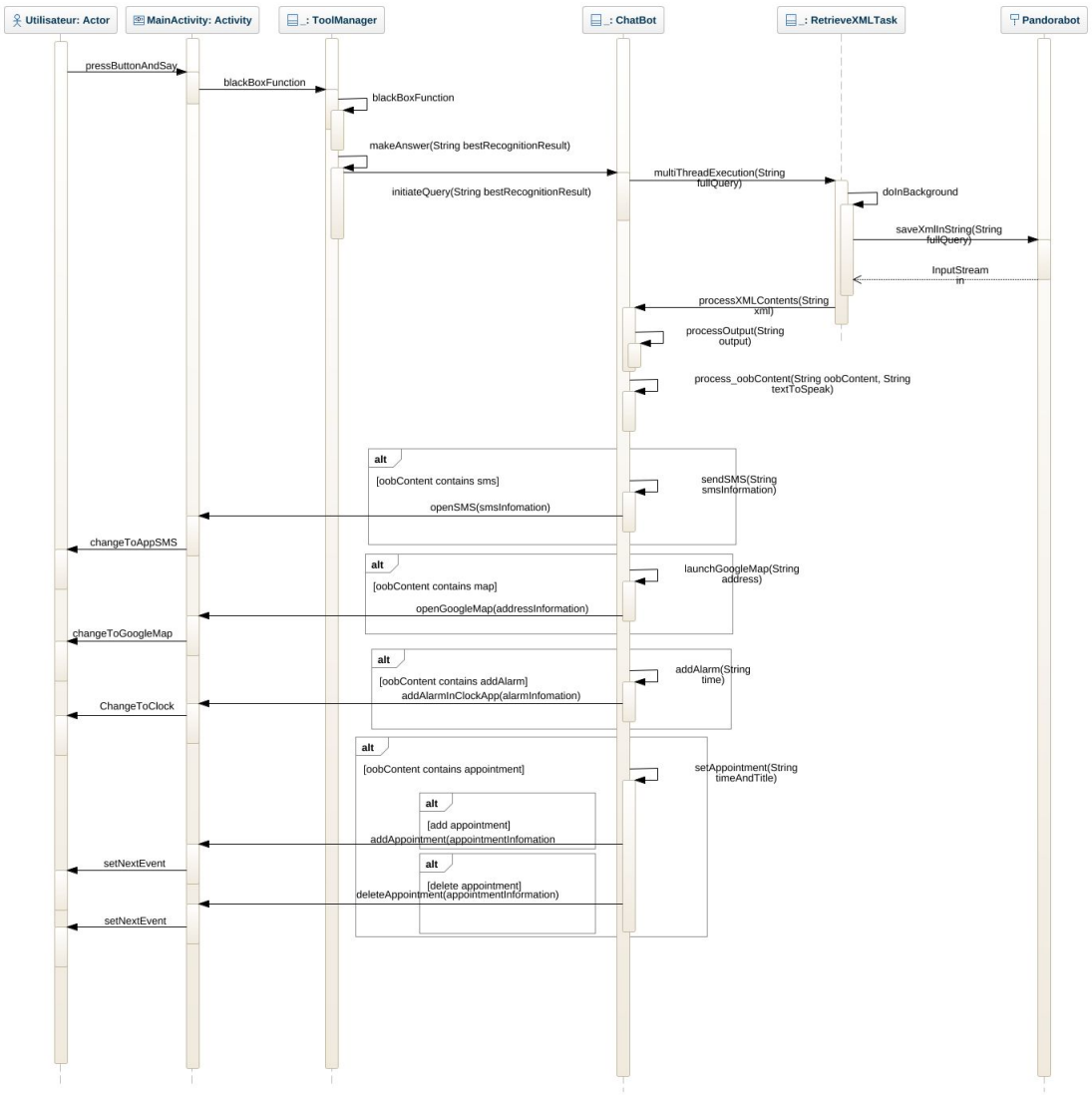

FIGURE 3 – Diagramme de Sequence pour envoyer un sms.

3.3 Signatures Partie Chatbot

procédure setAppointement (E/S Gcal : GestionCalendar, E oobContent : String, operationType : String)

fonction setBeginTimeAndGetTitle (E oobContent : String, beginTime : Calendar, operationType : String) : String

procédure googleQuery (E/S googleSearchText : String)

procédure lauchApp (E app : String)

procédure lauchUrl (E/S url : String)

procédure lauchGoogleMap (E/S address : String)

procédure sendSMS (E oobContent : String)

procédure makePhoneCall (E oobContent : String)

procédure addAnAlarm(E oobContent : String)

procédure deleteAnAlarm (E oobContent : String)

3.4 Signatures Partie ToolManager

procédure setNextEvent()
fonction getNextEvent() : String

Chapitre 4

4 Conception Détaillée

Cette étape nous permet, à partir de conception préliminaire, de préciser l'algorithme des fonctions ou procédures principales.

4.1 addAnAlarme

```
procédure addAnAlarm(E oobContent:String)
début
    Entier hours <- première partie d'information d'heures
    si deuxième partie d'information d'heures est vide
        Entier minutes <- 0
    sinon
        Entier minutes <- deuxième partie d'information d'heures
    finsi
    Intent intent <- new Intent(AlarmClock.ACTION_SET_ALARM)
    intent.putExtra(AlarmClock.EXTRA_HOUR, hours)
    intent.putExtra(AlarmClock.EXTRA_MINUTES, minutes)
    si oobContent contient balise "<repetition>"
        ChaineDeCaractère days <- chaine de caractère entre la balise "<repetition>"
        daysNumber <- nouvel tableau d'entier
        si days contient "jours"
            pour i de 1 à 7
                daysNumber.ajouter(i)
            finPour
        sinon
            si days contient "dimanche"
                daysNumber.ajouter(1)
            finsi
            si days contient "lundi"
                daysNumber.ajouter(2)
            finsi
            si days contient "mardi"
                daysNumber.ajouter(3)
            finsi
            si days contient "mecredi"
                daysNumber.ajouter(4)
            finsi
            si days contient "jeudi"
                daysNumber.ajouter(5)
            finsi
            si days contient "vendredi"
                daysNumber.ajouter(6)
            finsi
            si days contient "samedi"
                daysNumber.ajouter(7)
            finsi
        finsi
```

```
        intent.putExtra(AlarmClock.EXTRA_DAYS, daysNumber)
    finsi
    callingActivity.startActivity(intent)
fin
```

Listing 1 – addAnAlarm

4.2 sendSMS

```
procédure sendSMS(E oobContent : String)
début
    String sendSmsTo <- Chaine de caractère entre la balise "<people>"
    si la première caractère de sendSmsTo n'est pas une lettre
        si oobContent contient la balise "<message>"
            String smsContent <- Chaine de caractère entre la balise "<message>"
            Intent intent <- new Intent(Intent.ACTION_SENDTO, Uri.parse("smsto:" +
                sendSmsTo))
            intent.setData(Uri.parse("sms_body" + smsContent))
            commencer l'activité intent par callingActivity
        finsi
    sinon
        Cursor c <-
            callingActivity.getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
            null, null, null, null)
        String name <- ""
        String number <- ""
        String numberToSendSms <- ""
        String id
        bouge c au premier
        booléen trouve <- faux
        tant que non trouve
            si c.getString(0) est non null
                name <- CDC de la colonne de
                    ContactsContract.Contacts.DISPLAY_NAME
                id <- CDC de la colonne de ContactsContract.Contacts._ID
                si ContactsContract.Contacts.HAS_PHONE_NUMBER est vrai
                    Cursor pCur <-
                        callingActivity.getContentResolver().query(ContactsContract.
                        CommonDataKinds.Phone.CONTENT_URI, null,
                        ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "
                        =?", new String[]{id}, null);
                    tant que pCur peut bouger à la suivante
                        number <- CDC de la colonne de
                            ContactsContract.CommonDateKinds.Phone.NUMBER
                    fin tant que
                    ferme pCur
                finsi
                String[] words <- CDC de sendSmsTo séparée par " "
                StringBuilder sb <- new StringBuilder
                si longueur de words[0] est supérieur à 0
```

```
                    sb.append(words[0])
                    pour j de 1 à longueur de words
                        sb.append(" ")
                        sb.append(words[j])
                    finPour
                finsi
                String nom <- sb.toString()
                si name = nom
                    numberToSendSms <- number
                    trouve <- vrai
                finsi
            finsi
            bouge c à la suivante
        fin tant que
        c.close()
        si oonContent contient la balise "<message>"
            String smsContent <- CDC entre la balise "<message>"
            Intent intent <- new Intent(Intent.ACTION_SENDTO, Uri.parse("smsto:" +
                numberToSendSms))
            intent.putExtra("sms_body", smsContent)
            commencer l'activité intent par callingActivity
        finsi
    finsi
fin
```

Listing 2 – sendSMS

Cette fonction est modifiée selon la fonction *makePhoneCall(String oobContent)*. Le principe d'algorithme reste la même. Nous avons changé ce qui concernent à appeler à ce qui concernent à envoyer SMS.

4.3 setAppointement

```
procédure setAppointement(E/S Gcal : GestionCalendar, E oobContent, operationType :
    String)
début
    Calendar beginTime <- temps réel
    String title <- setBeginTimeAndGetTitle(oobContent, beginTime, operationType)
    si operationType contient la balise "</add>"
        Calendar endTime <- beginTime
        Gcal.ajouterRDV(title, beginTime, endTime)
    sinon
        Calendar endTime <- le dernier minute du jour de beginTime
        Gcal.supprimerRDV(title, beginTime, endTime)
    finsi
fin
```

Listing 3 – setAppointement

4.4 setBeginTimeAndGetTitle

```
fonction SetBeginTimeAndGetTitle(oobContent : String, beginTime : Calendar,
    operationType : String) : String
début
    String date <- Chaine de caractère entre la balise "<eventdate>"
    String title <- chaine vide
    Entier day <- 0
    Entier month <- 0
    Entier year <- 0
    Entier hour <- 0
    Entier minute <- 0
    Entier i <- 0
    String[] words <- séparer date par ' '
    si longueur de words = 0
        String sDay <- date
    sinon
        String sDay <- words[i]
    finsi
    si sDay est un Entier
        day <- parseInt(sDay)
        i <- i + 1
    sinon
        day = beginTime.DAY
    finSi
    Dictionnaire<String, Integer> months <- nouvel Hashmap
    months.put("janvier", 0)
    months.put("février", 1)
    months.put("mars", 2)
    months.put("avril", 3)
    months.put("mai", 4)
    months.put("juin", 5)
    months.put("juillet", 6)
    months.put("août", 7)
    months.put("septembre", 8)
    months.put("octobre", 9)
    months.put("novembre", 10)
    months.put("décembre", 11)
    si words[i] est dans months
        month <- months.get(words[i])
        i <- i + 1
    sion
        month <- beginTime.MONTH
    finSi
    si words[i] est un entier
        year <- parseInt(words[i])
        i <- i + 1
    sinon
        year = beginTime.YEAR
    finSi
    si oobContent contient la balise "<eventtime>"
```

```
        String time <- Chaine de caractère entre la balise "<eventtime>"
        String[] hr <- Chaine de caractère séparée par "h"
        si hr[0] est un entier
            hour <- parseInt(hr[0])
        sinon
            hour <- 0
        finSi
        si longueur de hr supérieure à 1
            minute <- parseInt(hr[1])
        sinon
            minute <- 0
        finsi
        si oobContent contient la balise "<event>"
            title <- Chaine de caractère entre la balise "<event>"
        sinon
            title <- Chaine de caractère entre la balise "</eventtime>" et la balise
                operationType
        finsi
    sinon
        hour <- beginTime.HOUR
        minute <- beginTime.MINUTE
        si i inférieur à la longueur de words
            title <- sous-chaine de date commence à indice de words[i]
        finsi
    finsi
    si operationType = "</delete>"
        beginTime.set(year, month, day, 0, 0)
    sinon
        beginTime.set(year, month, day, hour, minute)
    finsi
    retourner title
fin
```

Listing 4 – setBeginTimeAndGetTitle

4.5 getNextEvent

```
fonction getNextEvent() : String;
début
    Context context <- callingActivity.getApplicationContext()
    GestionCalendar gc <- new GestionCalendar(context)
    Cursor cursor <- gc.getEvents()
    mettre le curseur au premier
    Entier eventNumber <- nombre d'événement
    si eventNumber > 0
        StringBuilder sb = new StringBuilder
        Entier Long[] eventMillis <- tableau d'entité long à la taille eventNumber + 10
        Entier j <- 0
        répéter
            eventMillis[j] = le temps de début d'événement j
```

```
            j <- j + 1
        jusqu'à cursor arrive au dernier
        tirer tableau eventMillis
        mettre le curseur au premier
        Entier i <- 0
        tant que l'heure actuelle >= eventNumber[i]
            i <- i + 1
        fin tant que
        tant que le temps de début d'événement i <> eventMillis[i]
            cursor.moveToNext()
        fin tant que
        Entier Long endMillis <- le temps de fin d'événement trouvé
        String eventTitle <- titre d'événement trouvé
        Date startDate <- le temps de début d'événement trouvé
        Date endDate <- le temps de fin d'événement trouvé
        mettre les dates au format "d MM yyyy, H:m"
        String sb = "L'événement plus proche:\n" + eventTitle + startDate + endDate
        retourner sb
    sinon
        retourner vide
    finsi
fin
```

Listing 5 – getNextEvent

Chapitre 5

5 Développememnt

La partie développement s'est déroulée en quatre sous parties. Tout d'abord, nous nous sommes informés des méthodes proposées sur Android pour mettre en place les fonctionnalités précisées dans le Chapitre 4. Ensuite nous avons ajouté plusieurs catégories de question/réponse sur le site Pandorabots pour faire fonctionner notre fonctionnalités. Enfin, nous avons modifié l'interface d'utilisateurs de l'application pour améliorer l'expérience d'utilisation. Ainsi, Nous avons changer le personnage animé pour afficher plus d'information sur l'activité principale. Ces quatre sous parties sont étés implémenter itérativement tout au long de développement.