Université de Bordeaux 1

Projet de Programmation

Application CampusUB1

Auteurs :
Elyas Ben Hadj Yahia
Ryan Herbert
Paul Maribon-Ferret
Stefan Mijatovic
Tatiana Rocher

Clients :
Fabien Baldacci
Arnaud Casteigts

Encadreur :
Xavier Blanc

9 avril 2013

Préface

Ce document décrit le travail réalisé dans le contexte de notre projet de programmation. Le but de ce projet est de réaliser une application Android permettant d'accéder aux informations pertinentes des différents établissements du campus (Laboratoires, Universités, Ecoles). L'utilisateur choisit les établissements pour lesquels il veut accéder aux informations (dans un premier temps il n'y aura que l'université de Bordeaux 1 et le LaBRI). L'application permet ensuite d'accéder aux annonces d'événements (en offrant la possibilité des les ajouter à l'agenda) et aux annuaires (en offrant la possibilité d'ajouter aux contacts du smpartphone) sélectionnés.

L'idée est donc de rendre les informations liées aux établissements choisis plus faciles d'accès via un smartphone Android, afin de répondre à un besoin en déplacement, en donnant la possibilité d'enregistrer les informations importantes, et de les intégrer aux applications natives du smartphone, pour permettre une utilité de l'application lorsqu'une connexion Internet n'est pas disponible.

Remerciements

La réalisation de ce projet n'aurait pas été possible sans l'aide des personnes qui se sont impliqués dans notre travail. Pour cela, nous tenons à remercier :

Tout d'abord Monsieur Xavier Blanc pour son aide et ses précisions qui nous ont permis de réussir ce projet, sans lesquels nous n'aurions pu aller aussi loin dans l'aboutissement de notre logiciel.

Monsieur Lionel Clément pour ses conseils au sujet de l'analyse textuelle.

Nous remercions également nos clients Fabien Baldacci et Arnaud Casteigts pour l'intérêt qu'ils nous ont porté tout au long de notre projet.

Et finalement Monsieur Philippe Narbel pour ses conseils sur la conduite et le développement de ce projet.

Table des matières

1 | Etude de l'existant

1.1 Références

Android Application Development

L'ouvrage *Android Application Development* [1] est considéré comme l'une des références majeures dans le domaine du développement des applications sur Android. Il présente notamment l'architecture du système d'opération Android, ainsi que les différentes phases de développement des applications utilisateurs.

Site officiel de la SDK Android

Le site officiel de la SDK Android [2] met à disposition des développeurs une panoplie d'outils et de références. Ce site présente ses divers services, notamment les API d'Android et autres services de Google. On y trouve aussi plusieurs domaines d'application utilisés aujourd'hui qu'on peut intégrer dans notre application.

Article sur les applications mobiles des universités

Cet article sur les applications mobiles des universités [3] décrit les différentes contraintes liées au développement des applications mobiles d'universités. Etant donné que les étudiants sont de plus en plus équipés de smartphones, il devient intéressant de mettre en place une application mobile qui offre divers services utiles et faciles d'accès.

1.2 Applications existantes

uMontréal

Cette application propriétaire de l'université de Montréal [4] met à disposition de ses étudiants de nombreux services, tels que des flux d'actualités, un annuaire, un calendrier, et le plan du campus.

Plateforme Blackboard

Cette plateforme de développement est utilisée par la majorité des applications campus mobiles aux Etats-Unis [5]. On considère par exemple iStandford [6], application Android qui présente de nombreux services (internes et externes), dans la même philosophie que l'application uMontréal.

gReader, lecteur de flux RSS

Ce lecteur de flux RSS [7] offre une interface sobre et pratique pour gérer les abonnements aux flux RSS. Notre développement de l'interface graphique pourra s'inspirer du système des onglets et des toolbars de cette application.

Lecteurs de flux RSS open-source

Il existe déjà plusieurs lecteurs de flux RSS open-source, tels que Feedgoal [8] et Android-RSS [9]. les deux étant sous licence GNU GPL (v2 et v3, respectivement). On pourra étudier s'il est rentable de reprendre quelques modules, ou de repartir sur notre propre base.

1.3 Ressources existantes

LDAP

Le protocole LDAP est une ressource potentiellement importante à la conception de notre application, notamment pour la mise en oeuvre de l'annuaire. LDAP signifie Lightweight Directory Access Protocol ; c'est donc un protocole conçu uniquement pour les annuaires, et les annuaires du LaBRI et de Bordeaux1 s'avèrent être conformes aux normes LDAP. Ce sera donc une

des technologies importantes à explorer afin de réaliser une réponse robuste au parsage de l'annuaire du LaBRI.
Nous nous sommes posés la question de l'accessibilité de l'annuaire venant de l'extérieur, au travers du protocole LDAP ; cependant nous allons uniquement utiliser le protocole LDAP pour Bordeaux1 sachant que le serveur LDAP du LaBRI est inaccessible depuis l'extérieur sans passer par un système d'authentification sécurisée.

HTTP/HTML

Une alternative au protocole LDAP serait d'effectuer des requêtes HTTP, afin de parser des pages HTML. Cette méthode pourrait répondre aux besoins de parser les annuaires dont il en est question, si jamais nous ne pouvons nous reposer sur le LDAP. Le problème posé par cette méthode repose dans la robustesse de la solution. C'est-à-dire qu'un changement de l'implémentation des pages (changement/mise à jour des balises, ou même un changement de forme) pourrait rendre l'application inutilisable. L'utilisation de cette méthode demande énormément de travail de généricité du parsage des pages. Ceci pose plusieurs problèmes, notamment sur l'imprévisibilité des changements potentielles dans les pages. Il est impossible de prévoir à 100% les modifications qui pourraient avoir lieu sur les pages, et donc l'application pourrait rapidement devenir inutilisable. Il sera nécessaire de s'inspirer voire de réutiliser certains codes de web crawlers open source afin d'assurer d'avoir une application aussi robuste qu'elle puisse l'être en parsant des pages HTML.

Analyse textuelle

Au cours de l'implémentation de cette application, nous serons amenés à effectuer de l'analyse textuelle sur des textes afin d'en extraire les informations dont nous avons besoin, par exemple si une date n'est pas au sein d'une balise facilement repérable/exploitable. Il faudra donc tenter d'extraire des dates des textes disponibles. Cela pose plusieurs problèmes :

- Le texte peut faire référence à un événement passé, simplement à titre informatif, auquel cas, une date extraite n'aurait pas nécessairement de valeur à l'exploitation avec notre application.
- Le texte peut contenir plusieurs dates, faisant référence à plusieurs événements passés, présents, ou futurs et il sera donc difficile d'exploiter

les informations et de découper le texte en plusieurs événements, ou encore d'extraire la date pertinente à l'événement.
- Le texte peut ne contenir aucune date, n'étant simplement qu'une information quelconque.

Les événements inscrits dans les flux RSS du site de Bordeaux 1 possèdent une balise *pubDate*, qui devrait faire référence à la date de publication des articles, mais nous avons remarqué que pour la plupart des articles (notamment pour toutes les soutenances), cette date a été modifiée pour correspondre à la date de la soutenance. Cependant, ce n'est pas toujours le cas pour l'horaire. Certains articles possèdent des horaires qui correspondent à celle de l'événement (comme pour les soutenances, par exemple), mais d'autres ont un horaire qui ne correspond pas avec l'horaire de l'événement. Nous avons donc tiré la conclusion que l'horaire correspond parfois à l'horaire de publication de l'article.
Il sera donc possible de comparer les dates et horaires extraites par analyse textuelle avec les informations contenues dans la balise *pubDate* afin d'augmenter le taux de réussite de la proposition de date au moment de l'ajout dans l'agenda du smartphone.

Code 1.1 – Extrait d'un flux RSS de Bordeaux1 (29/01/2013)

```
<item>
  <title>Trophees du sport</title>
  <link>...</link>
  <description>
       Pour la 3e annee consecutive...
  </description>
  <content:encoded>
    <![CDATA[
      Ils seront recompenses a l occasion des Trophees du sport
      le jeudi 31 janvier 2013 a 18h dans l Atrium...
    ]]>
  </content:encoded>
  <pubDate>Thu, 31 Jan 2013 15:08:00 +0100</pubDate>
</item>
```

Sur cet exemple, l'heure qui apparait dans la balise *pubDate* ne correspond pas à celle de l'événement ; il va donc falloir analyser la description pour essayer de trouver l'horaire correspondant (ici 18h).

Etant donné ces problèmes, nous devrons surement trouver une approximation au problème, car nous ne pourrons proposer la bonne date à chaque fois qu'un utilisateur souhaite ajouter un élément à son agenda.
Il existe plusieurs outils utiles à l'implémentation d'une solution à ces problèmes, notamment des outils de parsing comme JFlex et CUP qui, d'après un livre d'Android [10] reste une solution possible. Ces outils ressortent pour nous, car leur utilisation nous est déjà familière. Ces outils cumulés avec les informations extraites d'un livre d'analyse textuelle [11] sur l'extraction d'information pourront nous permettre de mettre en place une solution efficace à ce problème.

2 | L'Application CampusUB1

2.1 Aperçu de l'application

2.1.1 Actualités

Une des fonctionnalités principales de l'application est la récupération, l'affichage et l'enregistrement des événements dans le calendrier de l'utilisateur. Pour cela, nous prévoyons de récupérer les flux RSS les plus utiles de l'université Bordeaux 1[12], notamment les flux disponibles sur la page d'accueil du site de l'université (dont les actualités). En ce qui concerne le site du LaBRI, il n'y a qu'un seul flux RSS pour les actualités, et le reste des événements se trouve sur d'autres pages HTML. Il faudra donc parser ces pages HTML pour en extraire les données importantes.

2.1.2 Annuaire

Il sera également possible de consulter les annuaires des établissements concernés, afin d'en extraire les informations désirées et les ajouter aux contacts du smartphone. Afin d'implémenter l'annuaire au sein de notre application Android, il est évident qu'il faut exploiter les annuaires de Bordeaux 1 [13] et du LaBRI. En ce qui concerne le LaBRI, le serveur LDAP n'est pas accessible depuis l'extérieur (sans authentification). Nous devrons donc intégrer l'annuaire à travers des requêtes GET, dans le but de parser les résultats obtenus. Pour le moment ceci semble être la meilleure solution envisageable. L'accès sans authentification au serveur LDAP de l'université de Bordeaux1 nous est permis, ce qui facilitera grandement l'intégration de la recherche dans l'annuaire pour cet établissement, ainsi que la maintenabilité du logiciel.

2.1.3 Utilitaires

Plan du campus

Dans un deuxième temps, nous prévoyons de mettre en place un plan du campus, potentiellement intégré à l'application Maps du smartphone. Ce service permettrait donc à l'utilisateur d'avoir un aperçu plus détaillé du campus en affichant les différents bâtiments et leur noms. On poura éventuellement aussi afficher les emplacements des différents services dédiés aux étudiants (cafétéria, restaurant universitaire, bibliothèque universitaire, ...).

Emploi du temps

Un autre service qui pourrait être très utile en particulier pour les étudiants de Bordeaux1 est l'accès et la sauvegarde locale de leur emplois du temps en fonction de leurs filières, des groupes, et des différentes options suivies. Pour cela, nous parserons les fichiers XML des emplois du temps utilisant la plateforme Celcat[14], afin de mettre à disposition une copie locale de l'emploi du temps dans l'agenda du smartphone.

3 | Analyse des besoins

Nous allons analyser dans ce chapitre les différents besoins fonctionnels et non-fonctionnels de notre application.
Nous allons donc utiliser un système de points pour évaluer la difficulté des tâches. On se basera sur une séquence de Fibonacci arrondie (1, 2, 3, 5, 8, 13, 20, 40, 60, 100), avec 100 reflétant une tâche d'une grande difficulté.
Les priorités des besoins sont classées sur cette échelle : optionnel, faible, moyen, important, vital.

3.1 Besoins fonctionnels

3.1.1 Fonctionnalités principales

Choix des établissements

Etant donné que l'application présentera des services en commun à plusieurs établissements, l'utilisateur doit pouvoir s'abonner aux établissements de son choix, s'il souhaite accéder à leurs propres informations [Fig. 9.1].

- Priorité : vital
- Difficulté : 1/100
- Risques : mise à jour des liens si les adresses des établissements sont modifiées.
- Validation : s'assurer que les sélections et les modifications des abonnements s'effectuent correctement.

Gestion des actualités

En accédant au menu des actualités, l'utilisateur disposera d'une interface à travers laquelle il recevra le flux des actualités des établissements auxquels il est abonné. Par défaut il aura alors la possibilité de voir les événements propres aux établissements auxquels il s'est abonné. Il pourra à tout moment changer de catégorie grâce au menu des catégories [Fig. 9.5] depuis la même fenêtre. De plus, il peut choisir entre voir la totalité des actualités des établissements auxquels il s'est souscrit, ou voir uniquement les actualités propres aux établissements qu'il a sélectionné dans les filtres. [Fig. 9.3]. L'utilisateur doit pouvoir lire l'annonce de ces événements [Fig. 9.4], et doit pouvoir facilement rajouter cet événement au calendrier de son smartphone (Google Calendar). Cette action redirigera l'utilisateur vers le calendrier, dans lequel l'événement sera créé avec les informations récoltées. L'utilisateur pourra donc modifier, valider ou annuler la procédure.

- Priorité : vital
- Difficulté : 40/100
- Difficultés techniques : extraction des informations pertinentes.
- Risques : changement de structure du contenu à parser (flux RSS et pages HTML).
- Validation : assurer l'intégrité des informations extraites.

Accès à l'annuaire

L'utilisateur doit pouvoir effectuer facilement les tâches suivantes :

- Rechercher un contact dans l'annuaire par son nom et/ou prénom.
- Visualiser les informations d'un contact.
- Rajouter un contact à la liste des contacts de l'utilisateur (créer un nouveau contact ou éditer existant).
- Envoyer un mail au contact.
- Appeler la personne (si un numéro de téléphone est disponible).
- Visiter le site web de la personne (à condition qu'il en existe un).

La recherche de personnes dans l'annuaire se fera en fonction des établissements auxquels l'utilisateur s'est abonné. On pourra donc choisir grâce à un filtre entre rechercher une personne dans l'annuaire propre à un ou plu-

sieurs établissements, ou la rechercher dans tous les annuaires. Lors de l'ajout d'un contact, l'utilisateur sera redirigé vers la page de création d'un nouveau contact, dans laquelle les champs pertinents seront pré-remplis, en fonction des informations collectées. L'utilisateur pourra donc modifier, valider ou annuler la procédure.

- Priorité : vital
- Difficulté : 40/100
- Difficultés techniques :
  - Avec LDAP : mise en place du protocole.
  - Sans LDAP : parsage des informations.
- Risques : changement de structure du contenu à parser.
- Validation : assurer l'intégrité des informations extraites.

Mises à jour

Notre application pouvant fonctionner hors connexion, les mises à jour de flux RSS étant de type pull, nous avons initialement pensé qu'il serait important de proposer les mises à jour des flux que nous parsons. Par exemple après une longue période sans connexion, nous voulions sauter sur l'occasion pour proposer des mises à jour à l'utilisateur. Or la solution que nous avons adoptée consiste à mettre à disposition un bouton *refresh* pour que l'utilisateur puisse effectuer une demande de mise à jour, non seulement cette option est préférable pour les personnes ayant un forfait Internet limité, ou encore pour l'utilisation à l'étranger, mais nous avons estimé que cette fonctionnalité serait plus utile que la première sachant que les établissements en question ne proposent que rarement des mises à jour (quelques unes par semaine). Cela ne justifie donc pas le besoin d'avoir un système de mises à jour automatique qui au lieu de faciliter l'expérience utilisateur, pourrait juste avoir des impacts sur les performances globales. En effet, faire des mises à jour trop souvent et de manière inutile fait perdre du temps à l'utilisateur et demande plus de travail de la part du CPU et mène donc à une consommation excessive de la batterie du téléphone. L'application devra garder en mémoire les derniers éléments téléchargés. Ainsi lorsque l'on n'a pas accès à Internet, on pourra accéder à ceux-ci. De plus, lors des mises à jour, on téléchargera uniquement les nouveaux événements. Ceci permettra à l'application d'économiser du temps d'exécution.

- Priorité : important
- Difficulté : 40/100
- Risques : mauvaise gestion de la mémoire.
- Difficultés techniques : éviter de télécharger de nouveau des informations déjà collectées.
- Validation : vérifier l'intégration des nouvelles informations.

3.1.2 Fonctionnalités secondaires

Dans un deuxième temps, nous comptons implémenter quelques options supplémentaires, qui ne rentrent pas dans le cadre du projet initial, mais qu'on estime être assez pratiques et utiles, et permettront de compléter l'application.

Abonnements et filtrage d'informations

Après avoir choisi lors du premier démarrage de l'application, les établissements auprès desquels l'utilisateur souhaite s'abonner [Fig. 9.1], il pourra à tout moment revenir sur son choix à travers le menu des préférences. Il pourra également en fonction de ses abonnements actifs visualiser la totalité des informations propres à un, plusieurs ou tous les établissements grâce à un outil de filtrage. Cet outil se présente sous la forme d'un menu quasiment identique à celui du choix des abonnements et sera accessible uniquement depuis l'écran d'accueil.
D'autres filtres concernant les événements seront proposés afin de permettre à l'utilisateur de maintenir une liste d'événements favoris marqués à l'aide d'une étoile qu'il pourra modifier à tout moment. Il pourra également choisir de visualiser uniquement les événements à venir, ou encore les événements non lus.

- Priorité : optionnel
- Difficulté : 60/100
- Difficultés techniques : s'assurer que l'abonnement simultané à un grand nombre d'établissements ait un impact mineur sur la stabilité de l'application.
- Risques : saturation trop rapide de la mémoire cache.
- Validation : affichage cohérent des informations liées aux abonnements actifs en un temps raisonnable (de l'ordre de 3 à 4 secondes par éta-

blissement).

Emploi du temps

L'utilisateur étudiant doit pouvoir récupérer son emploi de temps du semestre et l'intégrer à Google Calendar. Il devra simplement sélectionner dans la liste des emplois du temps, celui qui lui correspond.
Deux autres choix lui seront également proposés : il pourra visualiser l'emploi du temps directement en ligne dans un navigateur internet, ou alors télécharger une version au format *PDF* pour le stocker sur son téléphone.

- Priorité : optionnel
- Difficulté : 60/100
- Difficultés techniques : extraction des informations en fonction du choix de l'utilisateur.
- Risques : changement du format de l'emploi du temps.
- Validation : assurer l'intégrité des données extraites.

Plan du campus

L'utilisateur pourra naviguer sur le plan du campus en utilisant Google Maps. Des pointeurs vers les bâtiments principaux du campus ainsi que vers les points de restauration à proximité seront affichés et pourront être filtrés. Un outil de recherche sera également mis à disposition de l'utilisateur et lui permettra de rapidement trouver un bâtiment, un snack, ou un service proposé aux étudiants.
Le service de localisation intégré aidera également l'utilisateur à se repérer sur le campus en mettant fréquemment à jour sa position actuelle à l'aide des satellites GPS.

- Priorité : optionnel
- Difficulté : 60/100
- Difficultés techniques : maîtriser l'API de Google Maps.

3.2 Besoins non fonctionnels

3.2.1 Besoins de performance

L'application CampusUB1 est destinée à toute personne étudiant ou travaillant sur le campus de l'université. Elle devra donc être capable de tourner

sur les derniers smartphones, tout en restant compatible avec des smartphones plus anciens et donc moins puissants puisque tout étudiant n'a pas forcément les moyens d'aborder un smartphone récent ayant une forte puissance de calcul et une grande quantité de RAM. CampusUB1 ne sera donc pas gourmande en ressources et pourra exécuter ces services de manière assez rapide même sur les smartphones moins performants.
Nous testerons l'application sur les appareils suivants : LG Optimus 2X (P990), Sony Xperia S, Samsung Galaxy S3.

- Priorité : faible
- Difficulté : 40/100
- Difficultés techniques : bonne gestion de l'espace mémoire et de l'actualisation.
- Risques : implémention complexe et potentiellement longue.
- Validation : vérification que l'utilisation reste assez fluide et ne consomme pas beaucoup de ressources (à préciser un seuil de consommation de RAM).

3.2.2 Besoins de fiabilité

Etant donné que les différents services mis à disposition par CampusUB1 à ses clients s'avèrent souvent être d'un besoin professionnel, il est essentiel que les données affichées soient cohérentes et en accord avec les données d'origine. Si un chercheur du LaBRI recherche depuis notre application le numéro de la salle où a lieu sa conférence, il a tout intérêt à récupérer une information correcte. CampusUB1 ne peut tout de même pas garantir l'exactitude de toutes les informations données et ne se tient donc pas responsable d'éventuelles erreurs, sachant que l'application se base sur des algorithmes de reconnaissance de textes qui peuvent eux-mêmes être erronés d'origine. Il est évident que certaines questions de sécurité courantes se posent sachant que l'application nécessite une connexion à Internet continue, mais vu que CampusUB1 utilise uniquement des données publiques dans les services qu'elle fournit, la question de confidentialité des données ne se pose pas.

- Priorité : moyen
- Difficulté : 60/100
- Difficultés techniques : fiabilité de l'algorithme d'analyse textuelle pour détecter les dates des événements.

- Risques : création d'un événement avec des données erronées.
- Validation : simulation de l'algorithme d'analyse textuelle sur suffisamment de données pour avoir un taux de réussite acceptable (à déterminer).

3.2.3 Besoins d'intégrité de données

CampusUB1 doit accéder à certaines ressources personnelles du client pour pouvoir fonctionner, ce qui pose un problème de protection des données. Le calendrier et le répertoire du client sont les ressources auxquelles l'application est susceptible d'accéder, et on devra s'assurer que toute perte éventuelle de données déjà existantes ne soit pas engendrée par une erreur provenant de l'application. L'erreur humaine étant évidemment toujours possible. Toute modification éventuelle de données privées de l'utilisateur se fera donc impérativement avec son avis.

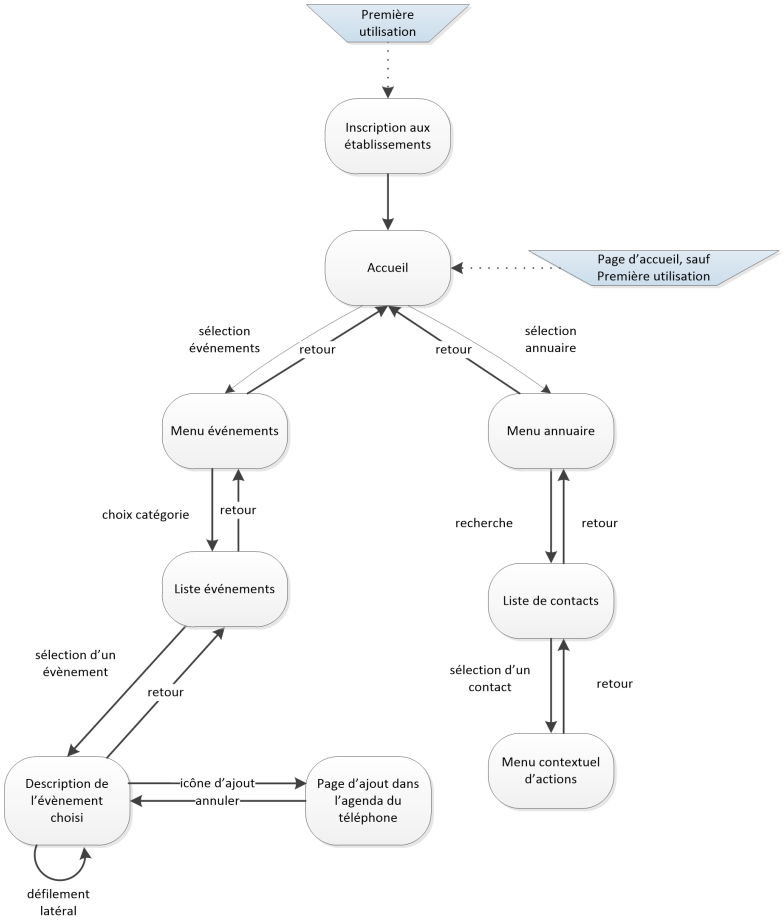
- Priorité : moyen
- Difficulté : 40/100
- Risques : perte des informations existantes sur le smartphone.
- Validation : monitoring du comportement de l'application.

3.2.4 Besoins organisationnels

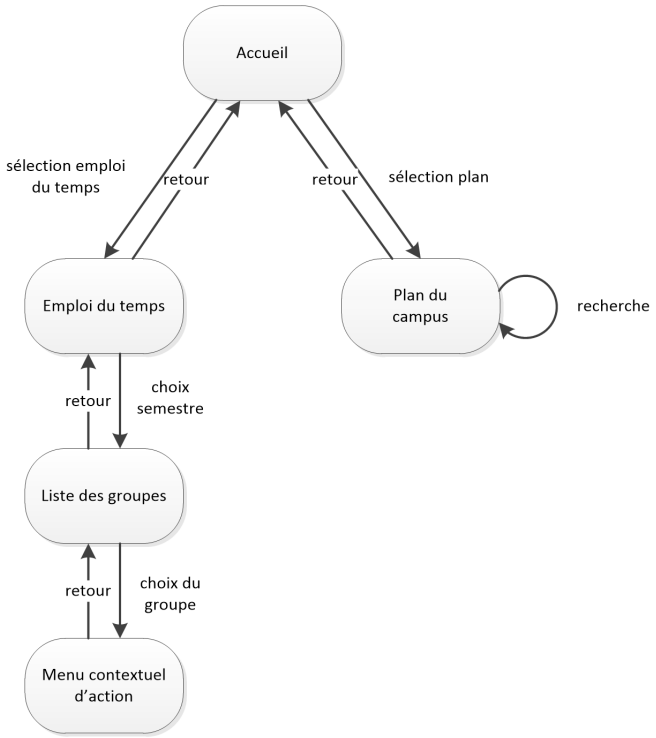
Etant donné que nous travaillerons sur la plateforme Android, le langage Java nous est imposé vu que l'API Android est fournie en Java. L'IDE dans lequel nous avons choisi d'implémenter notre application est Eclipse pour de nombreuses questions pratiques. Les développeurs d'Android disposent d'un plugin officiel pour Eclipse, qui facilite le travail du développeur en lui fournissant une panoplie d'outils tels que les différentes perspectives dont celle de débogage qui nous sera très utile. Ce plugin comporte également un émulateur d'appareils Android, ce qui nous permettra de facilement tester l'application sur une multitude d'appareils virtuels. De plus, l'API Android est très documentée et est maintenue fréquemment.

4 | Diagrammes de fonctionnement

4.1 Fonctionnalités de l'application

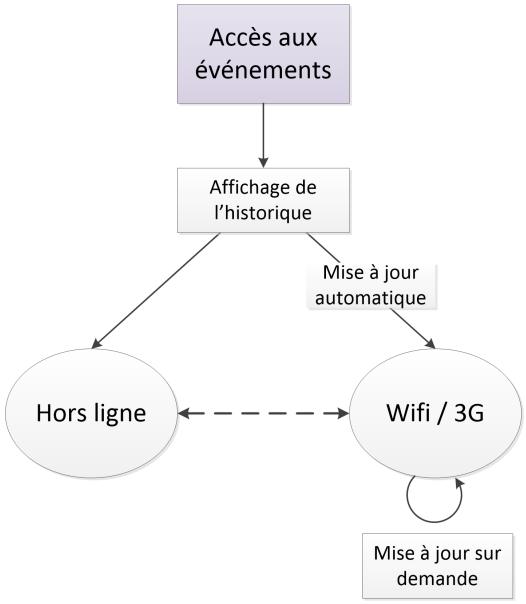


Le diagramme ci-dessus présente toutes les actions réalisables par l'utilisateur et leurs enchaînements. Lorsque l'utilisateur démarre l'application pour la première fois, il choisit l'établissement auprès duquel il veut s'abonner. Ce choix est alors retenu, et cette étape est donc omise lors des futurs démarrages de l'application. Une option existera tout de même pour modifier les abonnements si nécessaire. Depuis l'écran d'accueil, l'utilisateur pourra alors accéder aux services de son choix et pourra à tout moment revenir en arrière facilement.



Le diagramme ci-dessus présente les services offerts pour l'établissement Bordeaux1. Ces services supplémentaires seront implémentés après avoir assuré les fonctionnalités principales.

4.2 Diagramme des mises à jour



Le diagramme de mises à jour représente les différents modes de connexion dans lesquels on peut se trouver lorsqu'on accède aux événements. Il y a trois modes possibles gérés par le téléphone :

- Hors Ligne : pas d'accès à Internet.
- 3G : accès au réseau de données mobiles (lent).
- WiFi : accès à un réseau sans-fil (rapide).

5 | Tests et Validation

Test 01 : Choix de l'établissement

1. Démarrer l'application pour la première fois, un menu vous sera proposé pour effectuer le choix de l'établissement [Fig. 9.1].
2. Sélectionnez les établissements auxquels vous voulez vous abonner (on peut annuler pour quitter l'application).

Résultat attendu : Vous serez redirigé vers le menu principal de l'application.

Test 02 : Maintien du choix des établissements

1. Effectuer le Test 01.
2. Quittez l'application, et redémarrez-la.

Résultat attendu : L'application aura sauvegardé vos abonnements, et vous redirigera sur la page d'accueil de l'application.

Test 03 : Evénements

1. Une fois les abonnements configurés, sélectionner le menu des événements dans le menu principal [Fig. 9.2].

Résultats attendus :
- Si l'application est connectée à Internet : Les événements de tous les établissements auxquels l'application est abonnée sont affichés [Fig. 9.4].
- Sinon : Les événements ayant déjà été téléchargés sont affichés.

Test 04 : Validité des informations extraites d'un événement

1. Depuis l'accueil, appuyer sur Evénements.
2. Naviguer dans la liste des événements.
3. Appuyer sur un événement afin d'afficher la vue détaillée [Fig. 9.6].
4. Appuyer sur le bouton *Ajouter* dans la barre d'action.

Résultat attendu : Redirection vers la page de création d'événements de Google Calendar, avec les informations correspondantes à celles contenues dans l'événement.

Test 05 : Ajout d'un événement au smartphone

1. Effectuer le Test 04.
2. Valider l'ajout de l'événement.

Résultat attendu : L'événement sera ajouté à l'agenda du téléphone.

Test 06 : Annuaire

1. Sélectionner *Annuaire* dans le menu d'accueil.

Résultats attendus : On arrive sur l'écran de la figure 9.7.

Test 07 : Rechercher dans l'Annuaire

1. Accéder à l'annuaire à partir du menu principal.
2. Effectuer une recherche [Fig. 9.8], cela affichera tous les contacts qui correspondent à la recherche, parmis les établissements auxquels l'application est abonnée.

Résultat attendu : Les résultats se réduisent à ceux qui correspondent à la recherche.

Test 08 : Validité des informations extraites

1. Depuis l'accueil appuyer sur *Annuaire*
2. Effectuer une recherche de contact.

3. Sélectionner un contact.
4. Sélectionner *Ajouter contact*.

Résultat attendu : Redirection vers le répertoire du smartphone, avec possibilité d'éditer un contact existant, ou d'en créer un nouveau. Les champs seront pré-remplis avec les informations correspondantes à celles du résultat de recherche.

Test 09 : Ajout d'un contact au smartphone

1. Effectuer le Test 08.
2. Valider l'ajout/l'édition du contact

Résultat attendu : Les informations sont sauvegardées dans les contacts du smartphone.

Test 10 : Appeler un contact

1. Depuis le menu d'accueil, appuyer sur *Annuaire*.
2. Effectuer une recherche de contact.
3. Sélectionner un contact possédant un numéro de téléphone.
4. Sélectionner *Appeler*

Résultat attendu : Redirection vers le dialer, pré-rempli avec le numéro de téléphone du contact sélectionné.

Test 11 : Envoyer un mail à un contact

1. Depuis le menu d'accueil appuyer sur *Annuaire*.
2. Effectuer une recherche de contact.
3. Sélectionner un contact possédant une adresse mail.
4. Sélectionner *Envoyer un email*

Résultat attendu : L'application ouvre le gestionnaire d'email du smartphone avec le destinataire rempli avec l'adresse email du contact.

Test 12 : Importer un emploi du temps

1. Depuis le menu d'accueil, appuyer sur *Export emploi du temps*.
2. Sélectionner un semestre d'études [Fig. 9.9].
3. Sélectionner une filière et un groupe [Fig. 9.10].
4. Appuyer sur *Importer*

Résultat attendu : L'emploi du temps choisi est intégré au calendrier du smartphone.

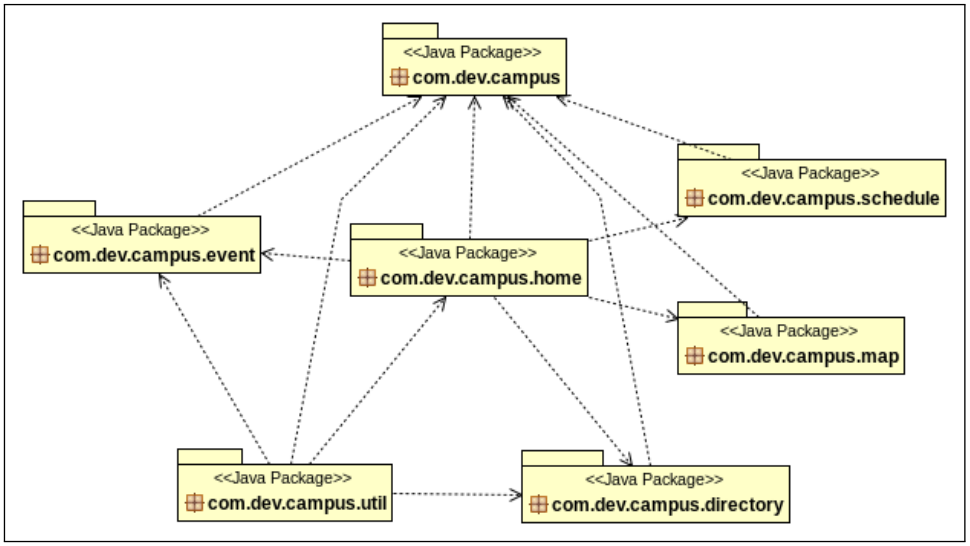
Test 13 : Afficher le plan du campus de Bordeaux1

1. Sélectionner *Plan du Campus* dans le menu d'accueil.

Résultat attendu :
- Si une connexion Internet est active : ouvre l'application Google Maps centrée sur le campus de Bordeaux1.
- Sinon : un message d'erreur s'affiche.

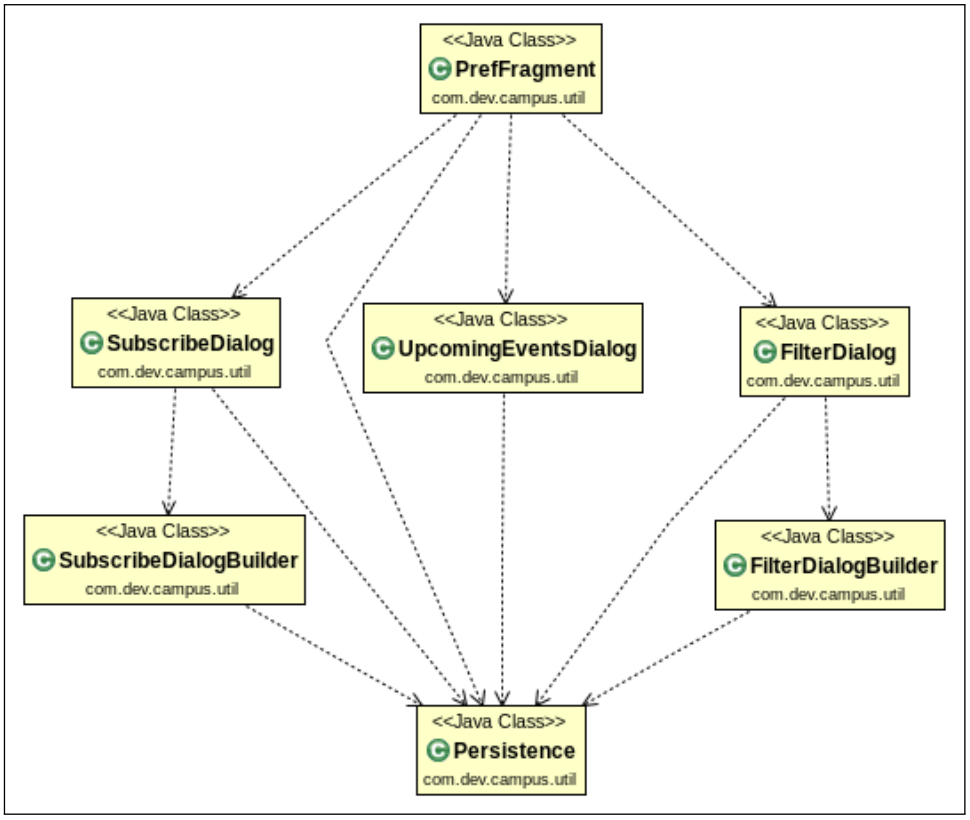
6 | Architecture

Aperçu de l'architecture



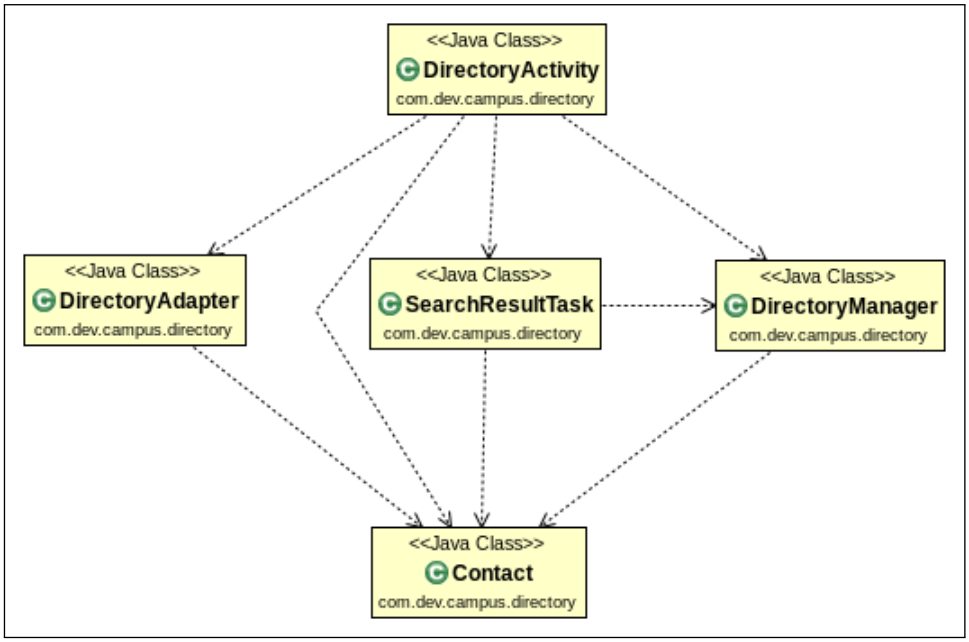
L'application est organisée en plusieurs packages, chacun définissant des services différents. On distingue les modules les plus importants : Settings, Directory, et Events.

Settings



Ce module s'occupe de la gestion des préférences de l'utilisateur. Toutes les opérations de lecture/écriture des préférences sont effectuées à travers ce module, notamment la gestion des abonnements et des filtres.

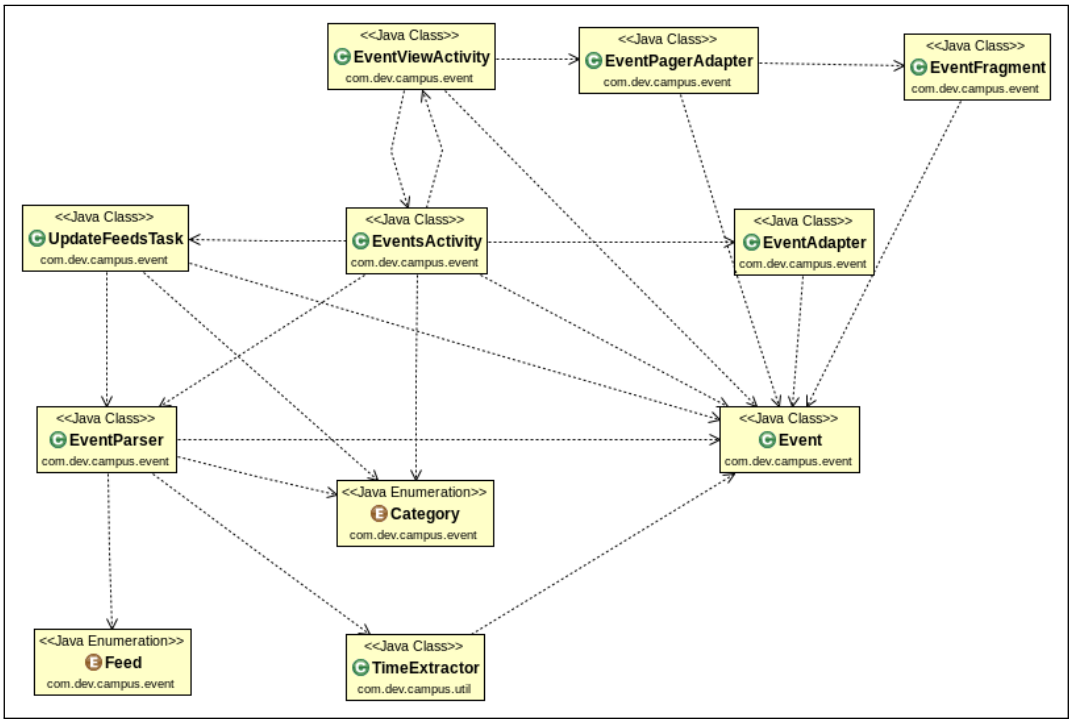
Directory



Ce module s'occupe de la gestion de l'annuaire. Il est responsable des services suivants :

- Parsage des réponses LDAP / pages HTML.
- Stockage et sauvegarde des données.
- Affichage graphique des contacts de l'annuaire.

Events



Ce module s'occupe de la gestion des événements. Il est responsable des services suivants :

- Parsage des flux RSS / pages HTML.
- Stockage et sauvegarde des données.
- Affichage graphique des événements.

7 | Descriptions techniques

7.1 Dépendances et librairies externes utilisées

Afin de pouvoir implémenter une partie de nos fonctionnalités principales et d'optimiser l'expérience utilisateur de l'application, nous avons eu à faire recours à plusieurs bibliothèques externes. Pour cela, il a avant tout fallu se documenter à propos des bibliothèques existantes et avoir des avis sur celles-ci afin de s'assurer qu'elles soient bien stables, à jour, et qu'elles fournissent tous les services qui nous sont nécessaires.

L'application utilise cinq bibliothèques externes :

- **UnboundID LDAP SDK** [15] : indispensable pour la connexion aux serveurs utilisant le protocole LDAP étant donné que la SDK de Java ne fournit aucun outil équivalent. On avait également encore le choix avec *JNDI LDAP* et *Spring LDAP* qui sont deux bibliothèques connues mais obsolètes et beaucoup plus complexes à utiliser.

- **Jsoup** [16] : la bibliothèque de référence pour l'analyse textuelle de textes HTML et XML. Elle est utilisée pour parser les événements des deux établissements ainsi que l'annuaire du LaBRI.

- **Google Play Services** : bibliothèque nécessaire pour utiliser les services de Google Maps et donc implémenter les services affectant le plan du campus.

- **Sliding Menu** [17] : bibliothèque open-source qui sert à améliorer l'esthétique et l'expérience utilisateur au sein du service qui affiche les événements. Elle permet de rajouter un menu latéral qui sert à basculer

entre les différentes catégories d'événements à tout moment en faisant un simple geste de glissement avec le doigt.

- **Robotium** [18] : bibliothèque open-source spécifique à la plateforme Android qui nous a été utile durant la phase finale du projet afin d'effectuer des tests en boîte noire pour notre interface graphique.

7.2 Recyclage de vues dans ListView

Android fournit des outils pour la gestion des listes graphiques, notamment ListView couplée avec l'interface ListAdapter.
ListView est conçue pour être extensible et performante, ce qui signifie :

1. ListView va essayer d'effectuer des inflations de vues aussi peu que possible.
2. ListView ne va dessiner et disposer ses fils que quand ils sont visibles sur l'écran (ou sur le point de l'être).

Le point 1 se justifie par le fait que les opérations d'inflation de layout sont coûteuses (de l'ordre de 1kB de RAM par vue). Ce problème est résolu par le recyclage des vues non visibles, qu'on appelle *ScrapView*. Cela signifie qu'on peut utiliser des vues recyclées et les mettre à jour, au lieu de faire des inflations de vues pour chaque rangée.

Afin d'implémenter le point 2, ListView utilise un recycleur de vues qui déplacera les vues actives dans une pool recyclable quand elles sortent de l'écran. Ainsi, ListView n'a besoin de garder en mémoire que les vues visibles à l'écran, ainsi que quelques vues recyclées - même quand ListAdapter contient des centaines d'items.

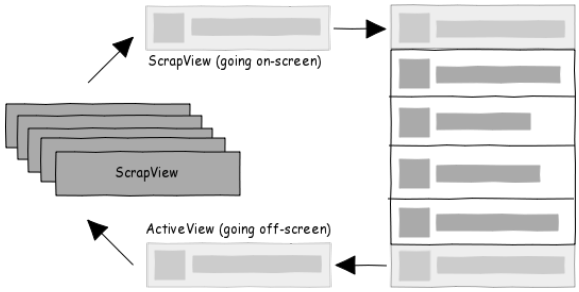


FIGURE 7.1 – Recyclage de vues lors du déroulement de ListView vers le bas

Implémentation de getView()

A chaque fois que ListView a besoin d'afficher une nouvelle ligne sur l'écran, elle appelle la méthode *getView()* depuis son adapter.

Code 7.1 – Version naïve

```
public View getView(int position, View convertView, ViewGroup parent){
    View item = mInflater.inflate(R.layout.item, null);
    ((TextView) item.findViewById(R.id.text)).setText(DATA[position]);
    ((ImageView) item.findViewById(R.id.icon)).setImageBitmap(mIcon);
    return item;
}
```

L'argument *convertView* est essentiellement une vue recyclée (*ScrapView*), comme indiqué précédemment. Quand il a une valeur non nulle, il faut en profiter pour simplement mettre à jour les données au lieu de faire une nouvelle inflation de layout. On peut donc optimiser le processus en employant ce paramètre :

Code 7.2 – Version correcte

```
public View getView(int position, View convertView, ViewGroup parent){
    View item = converView;
    if (item != null) {
        item = mInflater.inflate(R.layout.item, parent, false);
    }
    ((TextView) item.findViewById(R.id.text)).setText(DATA[position]);
    ((ImageView) item.findViewById(R.id.icon)).setImageBitmap(mIcon);
    return item;
}
```

View Holder pattern

L'opération de recherche d'une vue dans un layout s'effectue à l'aide de la méthode *findViewById()*. Cette méthode va chercher récursivement dans l'arbre des vues le nœud correspondant à l'ID spécifié. Comme ListView appelle souvent la méthode *getView()* de son adapter lorsqu'on défile la liste, *findViewById()* peut devenir alors assez coûteuse.
Le pattern View Holder consiste alors à réduire les appels de *findViewById()* dans *getView()*. Il s'agit donc d'une classe interne légère qui maintient des références directes à toutes les vues internes d'une rangée de la liste, et qu'on peut la stocker en tant que *tag* dans la vue de la rangée.

Code 7.3 – Version optimisée

```
static class ViewHolder {
    TextView text;
    ImageView icon;
}

public View getView(int position, View convertView, ViewGroup parent){
    ViewHolder holder;

    if (convertView != null) {
        convertView = mInflater.inflate(R.layout.item, parent, false);
        holder = new ViewHolder();
        holder.text = (TextView) convertView.findViewById(R.id.text);
        holder.icon = (ImageView) convertView.findViewById(R.id.icon);
        convertView.setTag(holder)
    } else {
        holder = (ViewHolder) convertView.getTag();
    }
    holder.text.setText(DATA[position]);
    holder.icon.setImageBitmap(mIcon);
    return convertView;
}
```

En mesurant les performances des 3 implémentations précédentes, on arrive à 20 FPS pour la version naïve, 50 FPS pour la version correcte, et 55 FPS pour la version optimisée.

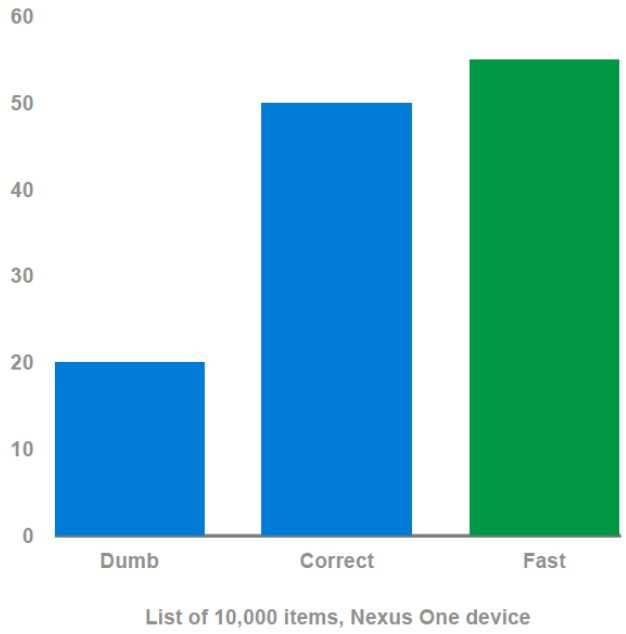


FIGURE 7.2 – Performance de *getView()* en fonction des implémentations. Source : Google I/O 2010 - The world of ListView.

8 | Profiling

En entammant la phase de test de notre application, nous avons tout de suite décidé d'attribuer à une personne de l'équipe les tâches de profiling qui sont cruciales afin d'optimiser au mieux les performances de l'application. Le plugin Android intégré à l'environnement de développement Eclipse nous fournit de base une panoplie d'outils utiles pour réaliser ces tâches. Ces outils se trouvent au sein de la perspective *DDMS* (Dalvik Debug Monitor Server) de Eclipse. Parmi ces outils, nous nous sommes essentiellement servi de quatre d'entre eux pour obtenir des informations sur le déroulement de notre application : Systrace, Method Profiling, Thread Tracking, et Network Statistics.

8.1 Systrace

L'outil Systrace trace une frise pendant une période donnée de marche de notre application, et nous montre les temps d'exécution de nos différents processus ainsi que des processus du système Android. De plus, il nous indique si l'utilisation du CPU durant l'exécution de ces tâches est plus ou moins intense. Dans notre cas, l'utilisation de cet outil nous a servi à déterminer et comparer les temps d'exécution de nos principaux services qui s'occupent de récupérer des donnés puis de les afficher (récupération et affichage des événements, recherche dans l'annuaire de personnes puis affichage de la liste des personnes trouvées...). En effet, nos temps d'exécution ainsi que d'affichage devaient respecter les temps évoqués dans le cahier des charges fourni aux clients.
Lors de la première capture du chargement des événements grâce à Systrace, nous avons pu constater que les performances de l'application pouvaient être optimisés lors de l'affichage des éléments qui est assez irrégulier (voir figure

8.1).

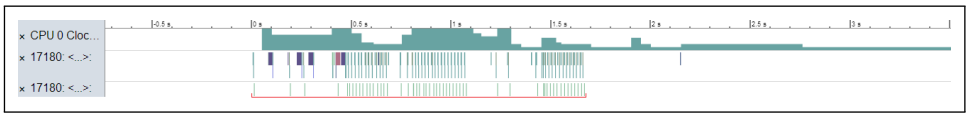


FIGURE 8.1 – Récupération et affichage des événements de l'université de Bordeaux 1 avant optimisation

L'élément qui nous a permis d'effectuer cette amélioration se trouve au niveau du *layout* qui affiche la liste des événements récupérés. Après s'être renseigné au sein de la documentation Android, nous avons appris que la modification de l'attribut booléen *android :baselineAligned* à *false* au sein du *layout* permet de gagner en performances en libérant l'application de la charge supplémentaire qu'elle avait à devoir aligner les *baselines* des éléments fils (Voir figure 8.2).

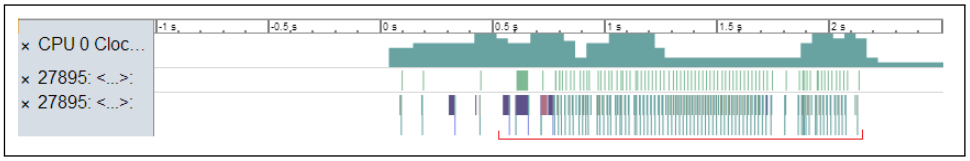


FIGURE 8.2 – Récupération et affichage des événements de l'université de Bordeaux 1 après optimisation

Comme on peut le voir sur l'enregistrement précédent, l'affichage des éléments se fait maintenant de manière nettement plus régulière que lors du premier enregistrement. L'optimisation de l'interface graphique a donc permis une amélioration des performances de l'application par la diminution du temps d'affichage.

8.2 Method profiling

Voici un tableau récapitulatif des temps d'exécution de nos fonctionnalités principales sur différents smartphones. Pour chacune d'elles, nous avons effectué des enregistrements qui montrent les temps d'exécution approximatifs des méthodes qui réalisent ces services. Les temps affichés correspondent donc au temps total d'exécution de nos threads appelés *AsyncTask* qui représentent

les tâches lourdes exécutes en arrière plan. Ces résultats sont approximatifs et peuvent varier étant donné que le temps d'exécution des méthodes dépend surtout de la fluidité de la connexion Internet utilisée.

	Samsung Galaxy S3 1.4Ghz QuadCore CPU	**Sony Xperia S** 1.5Ghz DualCore CPU	**LG Optimus 2X** 1Ghz DualCore CPU
Evénements UB1 (RSS)	**1.4sec**	**1.5sec**	**1.7sec**
Evénements LaBRI (RSS)	**2.1sec**	**2.3sec**	**2.4sec**
Evénements LaBRI (HTML)	**1.7sec**	**1.7sec**	**1.7sec**
Annuaire UB1 (LDAP)	**0.3sec**	**0.7sec**	**0.6sec**
Annuaire LaBRI (HTML)	**2.0sec**	**2.1sec**	**2.2sec**

Le temps obtenu pour le chargement des événements de l'Université de Bordeaux 1 est d'environ de 1.5 secondes (cf. figure 13.1), ce qui respecte largement les trois à quatre secondes indiquées dans le cahier des charges. Cela est du au fait que les événements de l'Université de Bordeaux 1 correspondent uniquement à des flux RSS à récupérer, ce qui représente une tâche peu couteuse et qui s'exécute relativement vite. En ce qui concerne l'abonnement au LaBRI, les événements sont récupérés de deux manières distinctes en fonction de la catégorie demandée. Ils pourront donc être récupérés soit via les flux RSS soit en parsant directement les pages HTML du site web. Cela explique alors le temps plus long (2.1 secondes) nécessaire au chargement des actualités du LaBRI sachant que le parsing de pages HTML est plus couteux qu'une simple récupération de flux RSS étant donné qu'il y a une grande quantité d'événements à parcourir, dont certains datant d'il y a plusieurs années. Dans tous les cas, on respecte toujours largement le temps évoqué dans le cahier des charges avec un temps d'exécution tout à fait correct. (cf. figure 13.2)

La recherche dans l'annuaire de l'Université de Bordeaux 1 s'effectue de manière quasiment instantanée : 0.3 sec pour une première recherche (cf. figure 13.3). Cela est dû au fait qu'on utilise une authentification auprès de l'annuaire de Bordeaux 1 en utilisant le protocole LDAP qui nous permet d'avoir une réponse rapide de la part du serveur. De plus, et étant donné

que nous sommes limités par le serveur quand au nombre de résultats retournés suite à une requête (dix résultats maximum), l'utilisation du système *Simple paged result control* nous permet de demander au serveur une page de résultats contenant donc dix contacts au maximum. Cela nous évite donc de surcharger le serveur et nous permet d'avoir sa réponse plus rapidement. L'utilisation d'un *Connection pool* nous permet également de maintenir plusieurs connexions actives simultanément auprès du serveur LDAP, et donc de diminuer le temps de retour des requêtes lors de recherches successives dans l'annuaire. (moins de 0.2 secondes lors de la deuxième requête). Il est à noter que le temps de réponse peut varier en fonction de l'état du serveur LDAP de Bordeaux 1 qui peut être plus ou moins occupé à un moment précis.

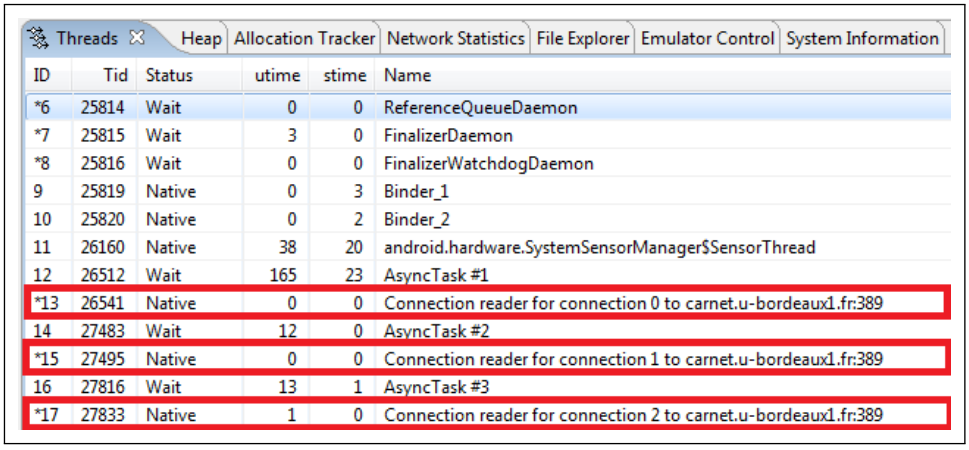
Etant donné que les contacts du LaBRI sont récupérés à l'aide du parsing HTML, il est évident qu'ils prennent plus de temps à être parcourus que ceux provenant de l'Université de Bordeaux 1 (cf. figure 13.4) En effet lorsqu'on effectue une requête auprès d'un serveur LDAP, on recherche uniquement la tranche de l'annuaire correspondant à la recherche effectuée, alors que dans le cas du parsing de l'annuaire du LaBRI, on parcourt la totalité des contacts présents sur la page HTML et on renvoie uniquement ceux correspondants aux critères de recherche. Cela explique donc les temps supérieurs de la recherche de contacts lorsqu'on est abonné au LaBRI (environ 2 secondes) qui découlent de la complexité accrue de l'algorithme d'analyse textuelle effectué lors de chaque requête. Cependant, ces deux secondes représentent un temps plus que correct qui est toujours suffisamment petit pour ne pas avoir d'impact sur l'expérience utilisateur.

8.3 Thread tracking

Grâce à cet outil, nous avons pu observer en temps réel l'activité des threads exécutés par notre application. Après avoir analysé plus précisément leur comportement pour chacun des services proposés, nous avons pu remarquer un défaut dans notre implémentation de la recherche dans l'annuaire de l'université de Bordeaux 1. A chaque fois que nous effectuons une recherche auprès du serveur LDAP, nous nous connections de nouveau auprès du serveur sans réutiliser la connexion précédemment établie. Nous pouvions alors constater (voir code et figure ci-dessous) qu'à chaque fois qu'une nouvelle

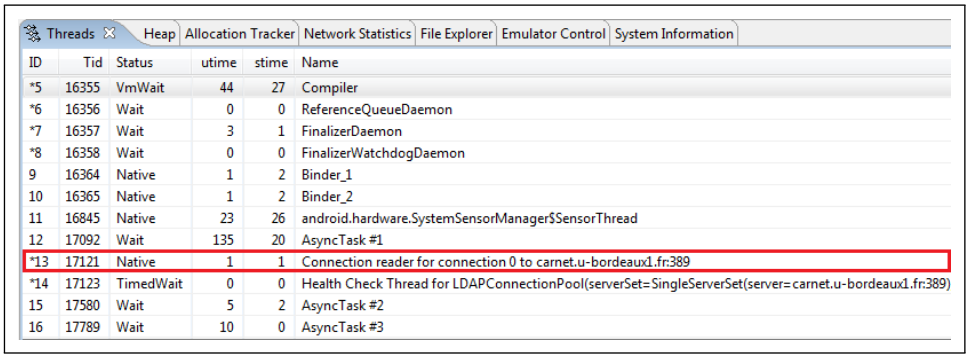
recherche était effectuée (représentée par le thread AsyncTask), un nouveau thread ConnectionReader était crée pour se ré-authentifier inutilement au serveur LDAP.

```
public List<Contact> searchUB1(String firstName, String lastName) throws LDAPException {
    LDAP = new LDAPConnection(UB1_LDAP_HOST, UB1_LDAP_PORT);
```



Cela nous a donc permis de rectifier ce problème en faisant en sorte de réutiliser l'authentification déjà existante, mais également d'apporter une optimisation en se servant de «pools de connexion» qui servent à maintenir plusieurs connexions établies auprès du serveur (10 au maximum), et qui peuvent être réutilisées par plusieurs threads.
On se retrouve donc à présent dans une situation où lorsqu'on effectue plusieurs recherches consécutives dans l'annuaire de Bordeaux 1, un seul thread ConnectionReader est créé pour établir la connexion, et les AsyncTask correspondants aux recherches réutilisent la connexion établie au préalable (voir code et figure ci-dessous).

```
public List<Contact> searchUB1(String firstName, String lastName) throws LDAPException {
    if(LDAP == null || connectionPool == null){
        LDAP = new LDAPConnection(UB1_LDAP_HOST, UB1_LDAP_PORT);
        connectionPool = new LDAPConnectionPool(LDAP, NUM_CONNECTIONS);
    }
```



ID	Tid	Status	utime	stime	Name
*5	16355	VmWait	44	27	Compiler
*6	16356	Wait	0	0	ReferenceQueueDaemon
*7	16357	Wait	3	1	FinalizerDaemon
*8	16358	Wait	0	0	FinalizerWatchdogDaemon
9	16364	Native	1	2	Binder_1
10	16365	Native	1	2	Binder_2
11	16845	Native	23	26	android.hardware.SystemSensorManager$SensorThread
12	17092	Wait	135	20	AsyncTask #1
*13	17121	Native	1	1	Connection reader for connection 0 to carnet.u-bordeaux1.fr:389
*14	17123	TimedWait	0	0	Health Check Thread for LDAPConnectionPool(serverSet=SingleServerSet(server=carnet.u-bordeaux1.fr:389)
15	17580	Wait	5	2	AsyncTask #2
16	17789	Wait	10	0	AsyncTask #3

8.4 Network statistics

L'outil d'analyse de paquets transitant sur le réseau à également été une statistique intéressante qui nous a permis de déterminer que notre application n'est pas en état de saturer un réseau, et de vérifier qu'elle n'est pas trop gourmande vis à vis du trafic de données.
D'après les analyses effectuées (voir figure 8.3), lorsqu'on est connecté en Wi-Fi à un réseau haut débit (ADSL dans le cas de notre test), et qu'on récupère la totalité des événements (abonné à l'université de Bordeaux 1 et au LaBRI), le réseau est utilisé pendant environ 7 secondes et de manière discontinue ce qui l'empêche d'être saturé. Durant ces quelques secondes, l'application reçoit en moyenne 57 Ko de données pour afficher la totalité des informations nécessaires, ce qui est relativement une petite quantité de données qui ne pose aucun problème pour être récupérée même sur des connexions à plus faible débit tels que les réseaux mobiles (3G, EDGE, GPRS).

Il en est de même pour l'utilisation de l'annuaire (Voir figure 8.4) où la quantité de données à récupérer est encore plus faible (environ 26 Ko) et varie de façon minimale en fonction de la recherche effectuée.

Ces éléments prouvent que notre application est apte à tourner de manière optimale sur des supports mobiles divers, utilisés en extérieur lorsqu'une connexion haut débit ne se trouve pas forcément à proximité.

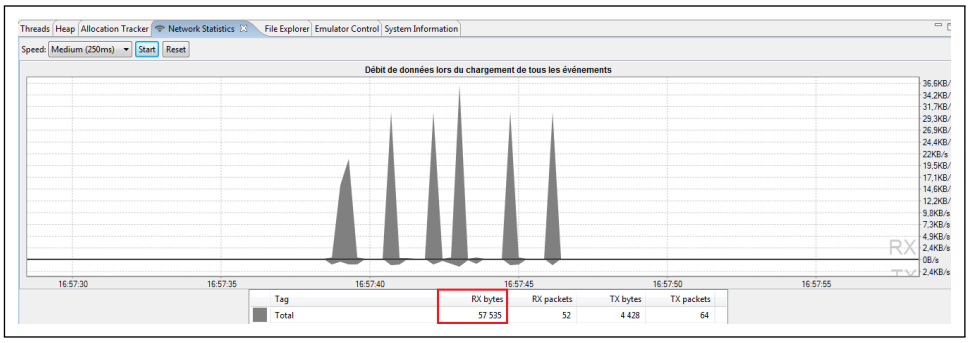


FIGURE 8.3 – Utilisation du réseau lors de la récupération de la totalité des événements du LaBRI et de Bordeaux 1

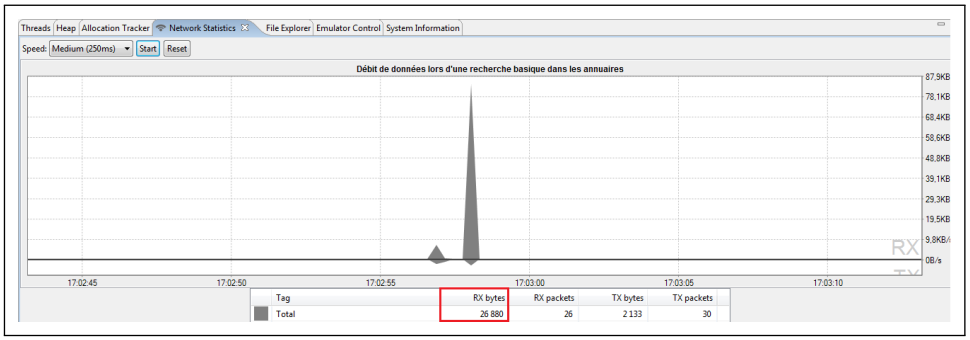


FIGURE 8.4 – Utilisation du réseau lors d'une recherche basique dans les deux annuaires à la fois

9 | Exemple d'utilisation

Les images suivantes représentent un premier aperçu de l'interface graphique de l'application. Nous imaginons donc un scénario d'utilisation qui montre le déroulement d'un usage typique.

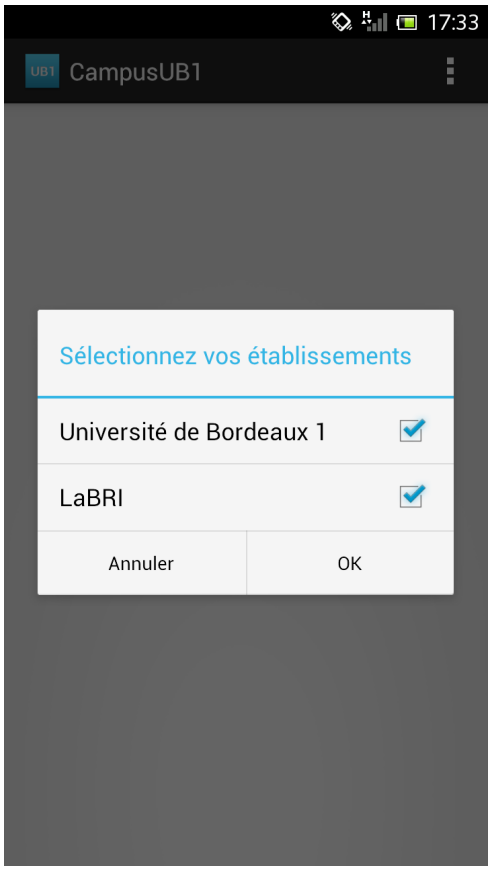


FIGURE 9.1 – Lors du premier lancement de l'application, l'utilisateur doit sélectionner ses établissements. Une fois ses choix effectués, il sont enregistrés et ne seront plus demandés. Les établissements sont modifiables via les préférences.

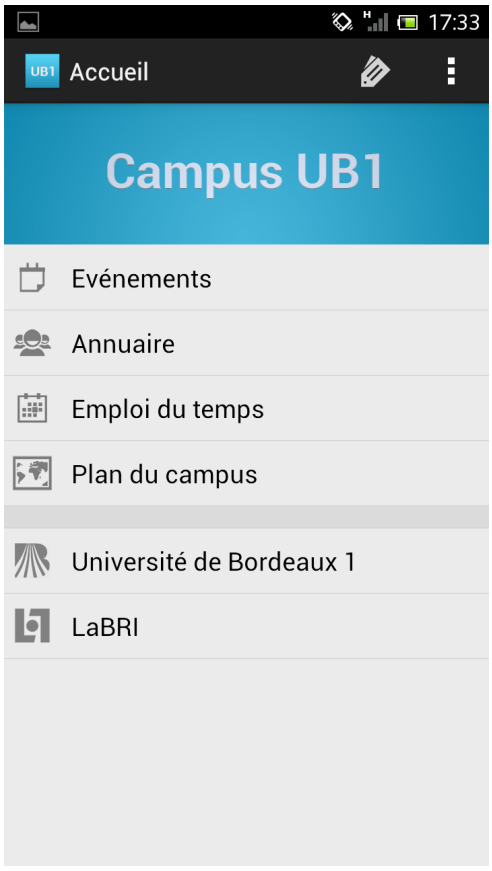


FIGURE 9.2 – Vue d'accueil de l'application

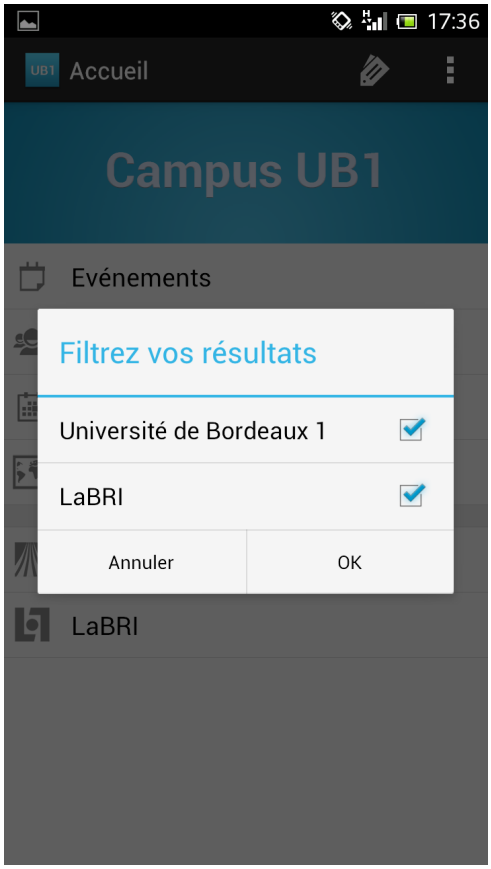


FIGURE 9.3 – Vue des filtres des résultats en fonction des établissements préalablement choisis. Cette fonctionnalité est accessible via la barre d'action.

FIGURE 9.4 – Vue des événements. Pour lire la news et arriver sur l'écran de la figure 9.6, il suffit d'appuyer sur la news correspondante.

FIGURE 9.5 – Liste des catégories des news.

FIGURE 9.6 – Vue détaillée d'une news. L'icone *Ajouter* de la barre d'action permet d'ajouter l'événement au calendrier du smartphone.

FIGURE 9.7 – Vue de l'annuaire.

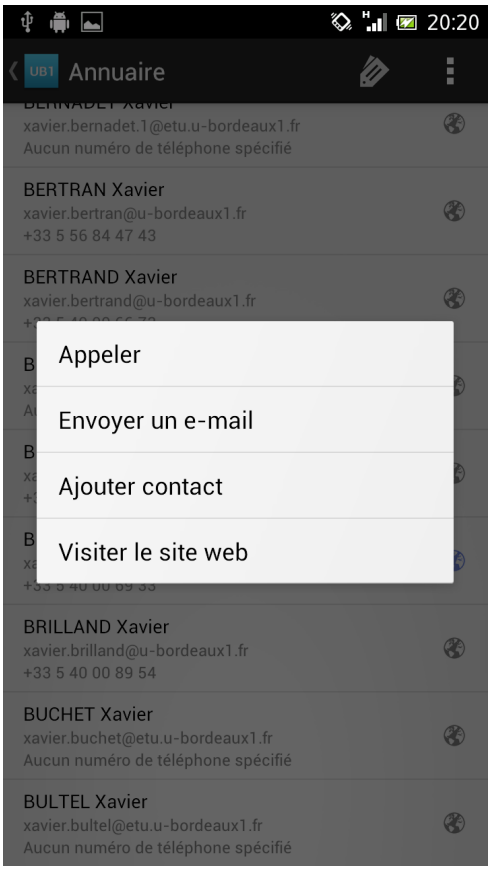


FIGURE 9.8 – Vue des résultats de l'annuaire.

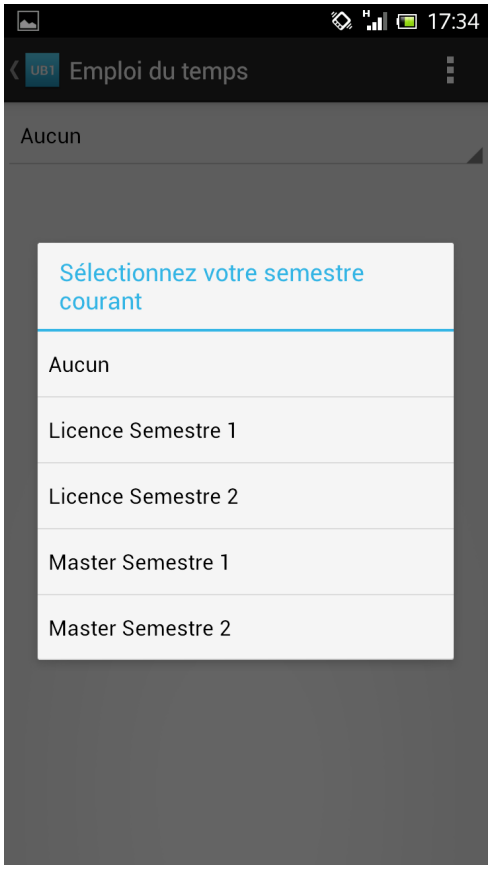


FIGURE 9.9 – Sélection du semestre de l'emploi du temps.

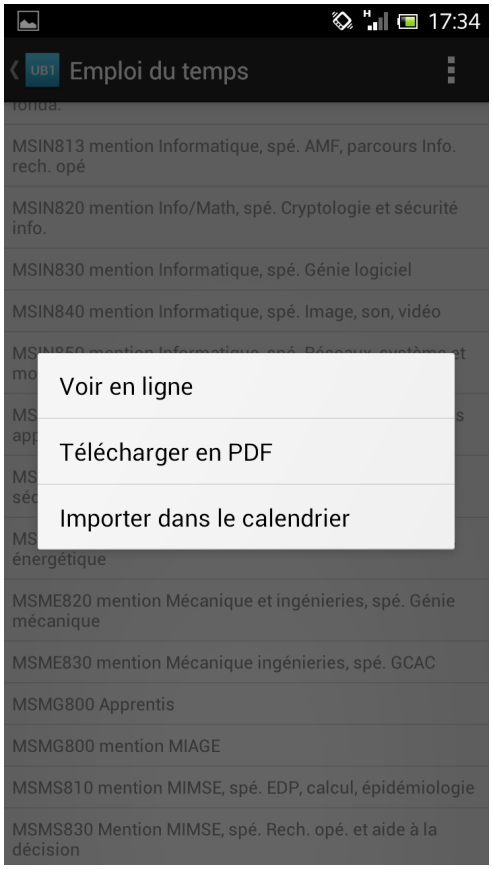


FIGURE 9.10 – Visualisations possibles de l'emploi du temps.

10 | Gestion du projet

10.1 Répartition des tâches

Dans un premier temps, nous avons segmenté l'implémentation de notre projet en travaillant sur les deux fonctionnalités principales : la récupération des événements, et la recherche dans l'annuaire.
Etant donné que pour chacune de ces deux tâches, il fallait s'occuper séparément de la partie concernant l'Université de Bordeaux 1 et de celle concernant le LaBRI, quatre tâches distinctes étaient à notre disposition ce qui nous a permis de aisément répartir le travail entre les membres de l'équipe. Le fait que certaines de ces tâches ayant été plus simples à implémenter que d'autres, nous a permis d'avoir beaucoup d'interactions entre développeurs et donc beaucoup d'entraide.

Une fois que nous avons constaté que l'implémentation des fonctionnalités principales évoluait vite et qu'elle était quasiment finalisée , nous avons orienté deux membres de l'équipe pour s'intéresser aux fonctionnalités secondaires de notre application : l'accès aux emplois du temps, et le plan du campus. Là aussi avaient lieu beaucoup d'interactions dans l'équipe afin de pouvoir disposer de l'avis de plusieurs personnes lors de l'implémentation de certaines fonctionnalités.
Une fois les tâches secondaires finies, l'application était à un stade très avancé, tous les besoins fonctionnels étaient achevés, il ne manquait alors plus que quelques touches de finalisation à apporter au sein des fonctionnalités principales. L'ensemble des membres de l'équipe ont alors pu contribuer à la mise au point finale.
Durant la phase finale de test, la répartition des tests à effectuer a été faite en fonction des tâches que chacun des membres a effectuées.

Le diagramme de Gantt ci-dessous présente une première estimation de la répartition des tâches sur le déroulement du projet. Chaque ligne représente une tâche à effectuer, une estimation de sa durée, et son attribution au sein de l'équipe de développement.

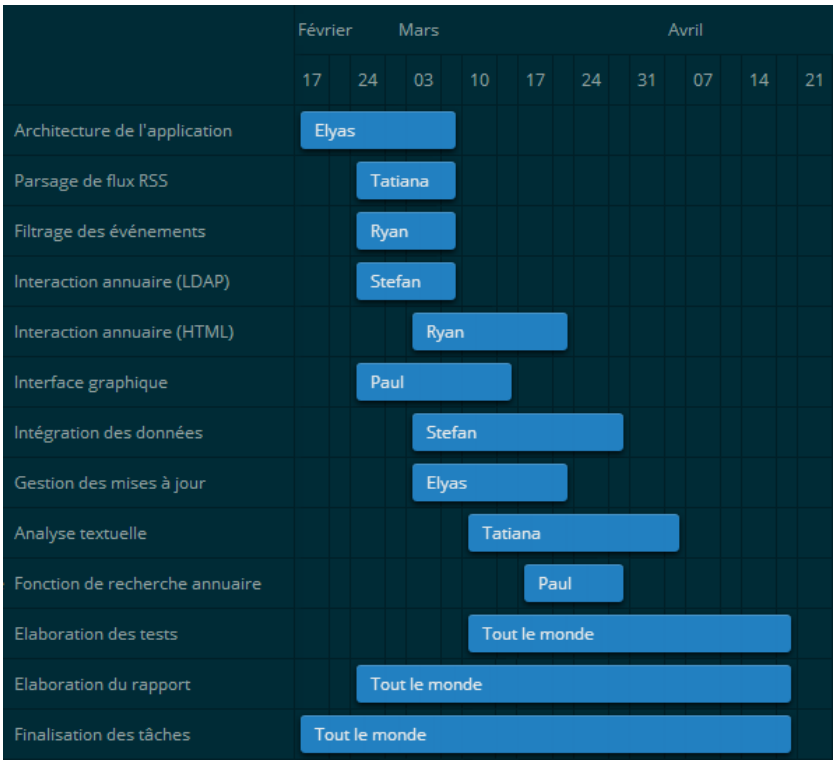


FIGURE 10.1 – Diagramme de Gantt des répartitions des tâches.

10.2 Outils de développement

Dès le début de notre projet, nous avons décidé d'utiliser *git* qui nous a paru être une solution plus complète comparée à *svn*. Au fur et à mesure que nous pensions à une tâche à effectuer, ou que nous détections un bug à corriger, nous avons ajouté une *issue* (problème ou tâche à effectuer) à notre projet sur GitHub. L'issue tracker de GitHub est très pratique, il permet de trier les issues par catégories, de les assigner à des membres du projet, ou

encore d'ajouter des commentaires. De plus il suffit de préciser le numéro de l'issue que l'on corrige lors d'un commit pour fermer cette tâche automatiquement. Git offre également la possibilité de créer des branches qui permettent d'implémenter des fonctionnalités séparément, puis de fusionner les branches une fois terminée.

10.3 Tests Unitaires

Afin de s'assurer que notre code soit robuste, nous avons effectué des tests unitaires, afin de s'assurer que toutes les parties de notre code fonctionnent correctement. Nous avons déterminé que les unités les plus petites de notre projet sont les classes ; les tests unitaires portent donc sur ces éléments. Les classes les plus importantes de l'application sont les *Activity*, car elles s'occupent du fonctionnement de chaque partie de l'application (notamment les événements et l'annuaire). Il a donc fallu tester ces classes en priorité.
Pour cela, nous avons utilisé Robotium, un outil open-source pour l'élaboration de tests sous Android. Il permet notamment de simuler toutes sortes d'intéractions avec l'interface graphique (clicks, touches longues, glissements et autres mouvements). Ainsi, nous avons pu implémenter des tests sur l'interface, réalisant quelques uns de nos tests d'utilisation.
Cet outil permet également de tester la couverture de code des applications Android, mais malheureusement ne supporte pas correctement les projets de tests ayant des dépendances externes, ce qui nous a donc empêché d'analyser la couverture du code de notre projet.

L'application étant dépendante d'un parsage de données téléchargées dont nous ne contrôlons pas le contenu, nous devons être sûrs que les parseurs fonctionnent au mieux possible. Nous avons donc effectué des tests à la fois avec du contenu contrôlé (que nous avons écrit et qui correspond aux formats des données que nous avons pu identifier sur les flux et les pages téléchargées), et sur du contenu réel, afin de s'assurer du bon fonctionnement de nos parseurs.
Il était également important de tester la classe *TimeExtractor*, afin d'être sûr que nous avions les dates correctes au cours du parsage et que l'ajout d'un événement important dans l'agenda du smartphone ne produirait pas une horaire décalée, afin d'éviter qu'un utilisateur soit en retard à un événement à cause de notre application.

11 | Difficultés rencontrées

Une des difficultés principales de ce projet tenait de l'apprentissage de l'API Android, et de son fonctionnement. L'API Android étant vaste, il n'était pas toujours facile de savoir si nous avions trouvé la solution optimale à un problème donné.
Nous avons décidé d'utiliser un *ViewPager* afin de rendre le défilement d'un événement à un autre plus ergonomique. Or pour cela il a fallu implémenter une nouvelle activité. Ceci a légèrement compliqué l'enregistrement des événements comme lus depuis cette activité. Nous avons cherché la meilleure méthode de transmettre des données d'une activité à une autre, et sommes arrivés à la sérialisation d'objets à l'aide des objets de type *Intent*. Cette méthode n'est pas forcément la solution idéale, mais semble être le moyen le plus apte à réaliser le travail nécessaire.

Lors de l'implémentation de l'extraction des dates, nous avons rencontré des problèmes concernant l'inclusion des librairies CUP et JFlex. Les méthodes n'étaient pas reconnues au sein du projet, rendant leur utilisation impossible. Nous avons donc opté pour l'utilisation d'expressions régulières qui reconnaissent des dates.

En ce qui concerne les événements, nous avons dû faire comme prévu, deux récupérations d'information différentes, le parsing des flux RSS et le parsing des pages HTML. Le parsing des flux et des pages a donné lieu à de nouvelles difficultés, notamment concernant les dates. Puisque beaucoup d'événements avaient la mauvaise date, voire même aucune date, nous avons dû essayer de déterminer la date correcte à partir du texte de l'événement.

Concernant les pages HTML, nous avons dû comprendre comment étaient organisées les pages, quels étaient les éléments à entrer dans la requête afin

de récupérer les événements sur une certaine durée. Nous avons utilisé *Jsoup* pour extraire les éléments. Ainsi, étant donné que les pages HTML du LaBRI sont agencées d'une certaine forme, il a fallu adapter le code pour parser ces pages. Bien que la tâche ne nous fut pas incombée, nous voulions vérifier que les événements étaient complets, afin de ne pas encombrer l'application d'informations incomplètes, voire inutilisables. En effet, certains d'eux ne possèdent pas de titre ou de descriptif, ou encore contiennent des informations inutilisables, voire insensées (voir exemple ci-dessous).

Code 11.1 – Exemple d'événement HTML sans contenu (descriptif vide)

```
<table width=100% cellspacing=0 style="border: solid 1px
  #D3CFC4;">
  <tr>
    <td bgcolor="#c0c0">
      2013-03-21 <!--date-->
    </td>
    <td
      colspan=4 class="surligne">
      <b>
        RENCONTRE P UNG <!-- titre -->
      </b>
    </td>
  </tr>
  <tr>
    <td width=100 valign=top>
      11:00-12:00 <!-- horaire -->
      <br>SALLE 1278 <!-- lieu -->
    </td>
    <td colspan=4 width=640>
      Intervenant: <br><br>
      <div align=justify>
      <!-- description -->
      </div>
    </td>
  </tr>
  <tr>
    <td></td>
    <td bgcolor=#cccccc width=75 align=center>
    <a href="javascript:openInfosActu('9121', '7371', 'groupe_details', '23', '0');">
      Plus d'infos</a>
    </td>
    <td width=435> <img src="../images/document.gif" title="Documentslies"></td>
</table>
```

Nous avons eu des difficultés avec l'utilisation de *ContentProvider* pour la gestion des événements du calendrier. En effet, selon les modèles de smartphone, l'utilisation de l'API est différente à cause notamment de la surcouche logicielle ajoutée par les constructeurs. Nous avons finalement décidé d'effectuer l'import de l'emploi du temps dans le calendrier par défaut. D'autres difficultés ont également été rencontrées pour insérer tous les événements en une fois (insertion par lots, ou *batch insert*). Mais après plusieurs expérimentations, de multiples recherches et la lecture de nombreux posts sur l'excellent forum StackOverflow, nous sommes parvenu à implémenter cette fonctionnalité.

Nous avons également eu quelques soucis pour réceptionner les réponses des requêtes que nous soumettions auprès du serveur LDAP de Bordeaux 1. Le serveur refusait de donner suite aux requêtes renvoyant plus de dix résultats et nous affichait le message d'erreur *size limit exceeded (maximum size is 10)*. C'est donc une restriction imposée par le serveur et il fallait faire avec. Cependant, il fallait tout de même trouver un système pour limiter la taille de nos requêtes à dix éléments pour éviter de lever une *LDAPException* qui nous empêchait d'accéder aux dix premiers contacts retournés. Une recherche plus approfondie au sein de la documentation de la SDK *UnboundID* nous a permis de trouver la solution au problème.

```
searchRequest.setControls(new Control[] { new SimplePagedResultsControl(MAX_PAGE_SIZE, null)});
```

L'utilisation du système *Simple paged result control* propre au protocole LDAP nous a permis de demander au serveur de nous renvoyer une seule page de taille *MAX_PAGE_SIZE=10* contenant les dix premiers résultats de la requête. Si la recherche effectuée contient donc plus de dix résultats, il seront ignorés à cause des restrictions techniques propres au serveur LDAP en question.

12 | Extensions et améliorations possibles

Il existe un grand nombre d'améliorations envisageables pour notre application qui est censée évoluer constamment après son déploiement. Dans un premier temps, il faudrait ajouter le support pour d'autres éventuels établissements appartenant au campus tels que l'IUT de Bordeaux 1 ou l'INRIA. Il est alors à prévoir qu'on risque très probablement d'avoir d'autres pages HTML organisées différemment à parser, et donc de nouveaux modules à intégrer.

Lorsqu'on a commencé à implémenter l'annuaire, notre choix initial était d'utiliser le serveur LDAP de l'Université de Bordeaux 1 ainsi que celui du LaBRI pour accéder aux contacts des établissements en question. Le problème étant que le serveur LDAP du LaBRI n'est pas accessible depuis l'extérieur sans authentification, il ne nous restait pas d'autre choix que de parser la page HTML du LaBRI en question. Une éventuelle amélioration possible aurait alors été d'utiliser le serveur LDAP sécurisé et de demander à l'utilisateur de s'authentifier s'il veut effectuer une recherche de contacts au sein du LaBRI. Cela aurait nettement amélioré le temps de recherche dans l'annuaire. Mais dans le cas pratique, l'utilisateur perdrait énormément de temps à entrer ses identifiants afin de lancer la recherche et accéderait finalement encore moins vite aux contacts qu'avec notre implémentation actuelle. Cette amélioration est donc à voir uniquement d'un point de vue d'optimisation des performances de l'application, et non du côté facilité d'utilisation.

Pour le plan du campus, on pourrait également donner plus de détails

concernant les marqueurs en donnant plus d'informations concernant celui-ci. Par exemple pour les bâtiments, on pourrait proposer à l'utilisateur de consulter le plan intérieur du bâtiment en question, ou pour les restaurants d'afficher les horaires d'ouverture et de fermeture ainsi que leur numéro de téléphone.
Une autre extension possible serait de pouvoir calculer les itinéraires routiers à pied ou en voiture afin de faciliter l'accès au campus, et à ses différents bâtiments. Il serait alors possible d'indiquer à l'utilisateur le chemin à effectuer pour se rendre à pied d'un bâtiment du campus à un autre.
Nous avons également réfléchi à une amélioration de l'outil de recherche intégré, qui serait capable de nous proposer dynamiquement une liste de suggestions correspondant à notre recherche en proposant un service d'auto-complétion.

En ce qui concerne l'emploi du temps, une fonctionnalité qui aurait pu accélérer l'affichage de l'emploi du temps ainsi que de l'améliorer esthétiquement aurait été de créer nous-même une vue de l'emploi du temps au sein de l'application en récupérant les informations nécessaires, et donc ne pas avoir à rediriger l'utilisateur vers le navigateur Internet qui prend du temps à s'ouvrir ainsi qu'à afficher les emplois du temps en format HTML. On pourra aussi sauvegarder les choix de l'utilisateur pour lui donner un accès rapide à son emploi du temps du semestre courant.

13 | Annexe

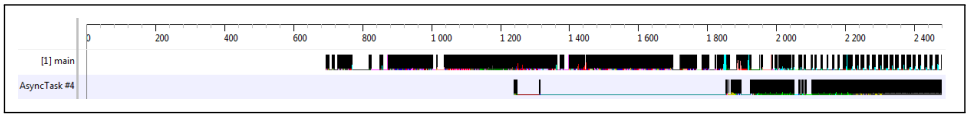


FIGURE 13.1 – Method profiling montrant le temps nécessaire à parser les événements de Bordeaux 1

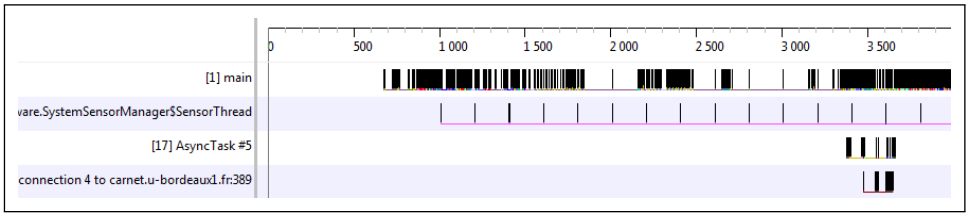


FIGURE 13.2 – Method profiling montrant le temps nécessaire pour récupérer les résultats d'une requête auprès du serveur LDAP de Bordeaux 1

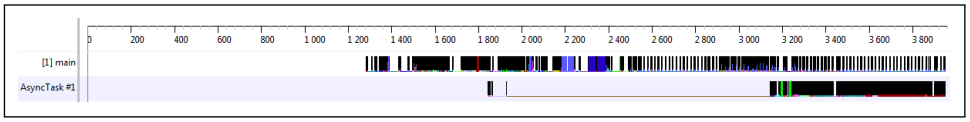


FIGURE 13.3 – Method profiling montrant le temps nécessaire à parser les événements du LaBRI

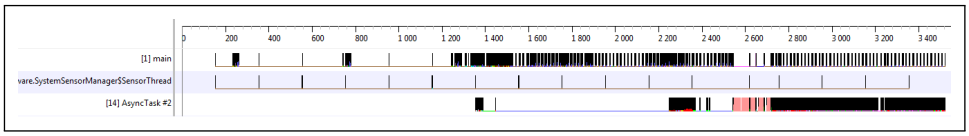


FIGURE 13.4 – Method profiling montrant le temps nécessaire à parser les contacts du LaBRI

Bibliographie

[1] Rick Rogers, John Lombardo, Zigurd Mednieks, and Blake Meike. *Android Application Development*. O'REILLY, first edition, 2009.

[2] Google. Android developers. http://developer.android.com/develop/index.html.

[3] Kerry Eustice. How to build a university mobile application : best practice and insight, Decembre 2011. http://www.guardian.co.uk/higher-education-network/2011/dec/08/building-university-mobile-apps.

[4] DGTIC Université de Montréal. uMontréal, Septembre 2012. http://www.umontreal.ca/mobile/.

[5] Blackboard. Blackboard Mobile, 2009. http://www.blackboard.com/platforms/mobile/overview.aspx.

[6] Stanford. iStanford, Juillet 2012. https://play.google.com/store/apps/details?id=com.blackboard.android.central.stanford&hl=en.

[7] Google. gReader, Septembre 2007. https://play.google.com/store/apps/details?id=com.noinnion.android.greader.reader&hl=en.

[8] FeedGoal. http://code.google.com/p/feedgoal/.

[9] Android-RSS. http://code.google.com/p/android-rss/.

[10] Mark Murphy. *L'art du développement Android 2e edition*. 2010. Chapitre 24.

[11] Daniel JURAFSKY & James H. MARTIN. *Speech and language processing*. 2000.

[12] Bordeaux 1. Flux RSS de Bordeaux 1. http://www.u-bordeaux1.fr/pages-speciales/abonnement-aux-flux-rss.html.

[13] Université de Bordeaux 1. Annuaire, 2013. http://www.u-bordeaux1.fr/no-cache/annuaires/annuaire-des-personnes.html.

[14] Celcat.fr. Emplois du temps. http://www.disvu.u-bordeaux1.fr/et/edt_etudiants2/Master/Semestre2/g56017.xml.

[15] UnboundID. Ldap sdk for java. https://www.unboundid.com/products/ldap-sdk/.

[16] Jsoup. http://jsoup.org/.

[17] Jeremy Feinstein. Slidingmenu. https://github.com/jfeinstein10/SlidingMenu.

[18] Robotium. https://code.google.com/p/robotium/.

[19] Savel Guilhem. Accès à l'annuaire des comptes mails, Octobre 2012. https://sos.labri.fr/sos/glpi/front/helpdesk.faq.php?id=78.