ISIMA

Institut **S**upérieur
d'**I**nformatique, de
Modélisation et de
leurs **A**pplications

BP 10125
63173 Aubière Cedex

Rapport d'ingénieur
Projet de 3ème année
Filière : Génie Logiciel et Systèmes Informatiques
Filière : Réseaux et Télécommunications

Communication entre smartphone et ordinateur

Présenté par : Iori MATSUHARA, Julien PINGUET
Sous la direction de : Loïc YON

Octobre 2012-Mars 2013

Remerciements

Avant toute chose, nous tenons à remercier les personnes qui ont acceptés de nous encadrer pour effectuer ce projet.

Nous remercions tout d'abord Loïc YON, enseignant chercheur en informatique à l'ISIMA, qui nous a encadré durant ce projet.

Nous remercions aussi Patrice LAURENÇOT, enseignant chercheur à l'ISIMA, pour avoir accepté notre proposition de projet.

Résumé

Plus les années passent, plus on on observe que le temps et le confort sont parmi les plus importantes ressources que nous souhaitions obtenir. La réalisation de ce projet essaye de subvenir à ces besoins en rendant la vie de ses utilisateurs plus agréable.

Ce travail consiste ainsi à créer un moyen de communication basé sur deux concepts. Nous voulons profiter de la disponibilité et la présence d'un correspondant sur son téléphone ainsi que de la praticité qu'apportent nos ordinateurs pour converser.

Alors comment réunir ces conditions afin de pouvoir communiquer plus rapidement et plus confortablement ? Nous avons donc choisi de rendre possible l'écriture de SMS depuis un ordinateur.

Pour atteindre cet objectif, la réalisation du projet s'est déroulée en plusieurs étapes. Nous avons tout d'abord étudié les protocoles de communications que nous souhaitions utiliser et mis en relation notre étude avec nos contraintes de réalisation. Suite à cela, nous avons mis en place un schéma global de fonctionnement de notre projet qui nous a permis d'estimer chacun des modules composant notre travail. Enfin, il a fallu établir deux systèmes de communication. Tout d'abord entre le téléphone mobile et une entité persistante sur internet puis entre cette même entité et l'utilisateur. Cette étude terminée, nous avons alors commencé notre travail d'écriture et tenté de rendre le fonctionnement du projet possible sur plusieurs types de plateforme mobile (Android et iPhone).

Ce projet a donc été effectué dans les temps et est aujourd'hui entièrement fonctionnel. Les résultats ont été concluants et nous avons un système capable d'envoyer et de recevoir des SMS depuis un site internet. Néanmoins, le fonctionnement de ce projet nécessite quelque modifications du téléphone utilisé. En effet il nous est apparu impossible de supplanter des contraintes imposées par les diverses systèmes d'exploitation sur lesquels nous avons travaillé.

Malgré la mise en place d'outils de travail par les constructeurs de téléphone portable, on a pu observer de nettes différences de politique. En effet, selon la plateforme, les contraintes seront beaucoup plus importantes et empêcheront le développeur de sortir des sentiers battus. Les téléphones fonctionnant sous iOS démontrent clairement cette état d'esprit puisqu'il nous a

été très difficile d'outrepasser les contraintes afin de réaliser notre projet sur cette plateforme.

Mots clé : SMS, XMPP, Android, iOS, Play Framework

Abstract

Years after years we observe that time and comfort are among the most important resources we are willing to obtain. For us, doing this project was all about gaining these precious resources and making our project customers life more comfortable.

This work consists in using a new way of communicating while using these two previous concept. Indeed we are willing to take advantage of the cellphones high availability with the convenience of computer communication.

So how could we meet these two conditions in order to offer a comfortable and fast way of communication ? We simply chose to create a software allowing us to send and receive SMS from a computer.

The project realisation has been done in few steps. First, we have been studying many communication protocol we wanted to use. With this analysis, we were able to choose the technologies which could overcome our main constraints. After that, we have been designing a global scheme representating our project allowing us to estimate each distinct part of the work to do. Finally, we had to established two systems of communication. The first one permits the message to be shared between the cellphone and whatsoever entity running on the internet. The second one forwards the same message from the entity to the user. This study over, we actually started to write programs and try to make this project possible on multiple mobile platform (Android and iOS).

This project have been done in time and is now entirely functional. Results are conclusive, we indeed have a project that allows us to send SMS from a website. Nonetheless, to be able to run correctly the software, we need to modify the behaviour of the cellphone used. In fact, it appears that it was impossible to overcome some of the constraints imposed by the various cellphone operating systems we worked on.

Even though, the system usually provides several tools to build applications, we notified huge politicy difference beetween the providers. Depending on the plateform, constraints will indeed become more important and prevent the developer from thinking outside the box. Cellphone using iOS are clearly representating this policy. For our project, despite our success in making this

project a reality, we really had difficult time trying to bypass the unique iOS way of doing things.

Keywords : SMS, XMPP, Android, iOS, Play Framework

Table des figures

Table des matières

Introduction

L'envoi de SMS depuis un ordinateur est très souvent fastidieux car il n'existe quasiment aucune solution simple, gratuite et pratique. En effet, les solutions existantes sont soit payantes, soit limitent leurs services à quelques envois.

Les opérateurs de téléphonie mobile proposent tous des forfait avec un envoi illimité de SMS, mais ne proposent aucune solution d'envoi à partir d'un ordinateur.

Comment pourrait-on utiliser notre forfait téléphonique pour envoyer des SMS depuis l'ordinateur ? Quelle serait la solution la plus pratique pour l'utilisateur ?

L'objet de cette étude sera d'étudier et de développer une solution d'envoi de SMS depuis un ordinateur, simple pour l'utilisateur et utilisable par tous. Cette solution se basera sur la communication entre le smartphone et l'ordinateur, qui s'échangeront les informations du SMS.

L'étude des différentes solutions possibles permettra de les tester et de comparer leurs avantages et défauts pour mieux nous amener vers la solution pratique et optimale. Une fois la solution choisie, elle sera développée pour permettre son utilisation. Enfin il ne restera plus qu'à l'optimiser et lui apporter de nouvelles fonctionnalités.

Tous d'abord nous présenterons la démarche suivie pour nous orienter vers la solution envisagée. Dans un deuxième temps, nous présenterons les outils utilisés et les techniques mises en œuvre pour développer le projet. Pour terminer, nous présenterons la solution finale obtenue et discuterons des améliorations possibles.

I Introduction de l'étude

I.1 Besoin de l'utilisateur

L'envoi de SMS depuis son ordinateur possède plusieurs avantages par rapport à l'envoi depuis un smartphone.

Il est beaucoup plus aisé d'écrire un texte depuis un clavier d'ordinateur, plutôt que sur un écran tactile ou un clavier à très petites touches. Cela peut aussi aider les personnes ayant des difficultés avec les smartphones, comme par exemple les personnes malvoyantes.

Tout le monde n'a pas accès instantanément à son smartphone lorsque l'on utilise un ordinateur. Celui-ci peut être dans notre poche, dans un sac, posé sur une table distante, ...et l'on aimerait écrire lire ou répondre à des SMS sans avoir à se déplacer tout en gardant les yeux sur l'écran et les mains sur le clavier.

I.2 Solutions existantes

Les solutions actuellement proposées aux utilisateurs sont très limitées, ce qui explique le fait que très peux d'entre-elles sont utilisées.

Chez certains opérateurs de téléphonie mobile le service SMS peut couter plus de 0,10€par envoi, et les tarifs des sites indépendants affichent des prix voisins. Des solutions gratuites existent et permettent des envois gratuits sur internet, mais ceux-ci sont limités à quelques envois ou bien demande une confirmation pour éviter les abus.

Un inconvénient remarquable parmi toutes les solutions existantes est l'absence de gestion des contacts. En effet l'utilisateur est obligé de spécifier "à la main" le numéro de téléphone du destinataire. Cela est fastidieux car il faut soit connaitre le numéro par cœur, soit accéder à son carnet d'adresse qui se trouve souvent sur son smartphone.

La dernière remarque que nous pouvons faire sur les solutions existantes est le fait que la réception des SMS n'est gérée. Cela peut s'expliquer par le fait que la réception de SMS nécessite

une mémorisation du message (dans le logiciel ou sur le serveur web), contrairement à l'envoi.

I.3 Solution envisagée

Pour établir notre cahier des charges, nous avons étudié les solutions existantes puis étudiés leurs principaux défauts pour produite une solution optimale pour l'utilisateur.

Sachant que la quasi totalité des forfaits mobiles autorisent un nombre illimité d'envois de SMS, nous voulons donc que les utilisateurs utilisent leur propre forfait mobile pour effectuer les envois et réceptions de SMS depuis l'ordinateur, en utilisant un protocole de communication qui transférerait les données.

Nous avons décidé d'intégrer la réception des SMS dans notre solution. Cette fonctionnalité devra se faire de manière sécurisée et confidentielle.

Enfin nous avons voulu proposer une gestion simple des contacts de l'utilisateur lui permettant de ne pas à avoir à entrer le numéro de téléphone. De même que pour la réception des SMS, les contacts devront être sûr.

Le schéma ci-dessous représente le fonctionnement très général de la solution. Les flèches vertes indiquent le cheminement du message lors de l'envoi, et les flèches bleues indiquent le cheminement du message lors de la réception.

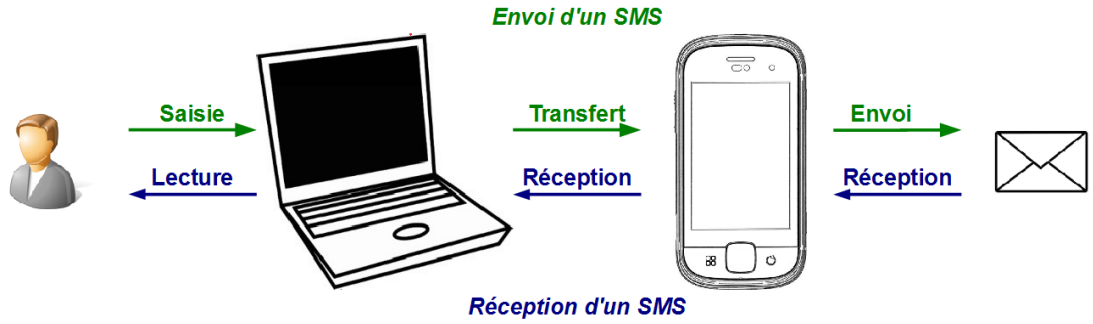

FIGURE I.1 – Fonctionnement général

II Recherche de la solution optimale

La première partie de notre projet a consisté à étudier les différentes étapes du fonctionnement de notre solution : de la réception d'un SMS sur le smartphone à son affichage sur l'ordinateur, mais aussi de l'écriture d'un message sur l'ordinateur jusqu'à son envoi par le smartphone. L'objectif de cette étude était de trouver la solution optimale que nous développeront pour ce projet. Nous détaillerons plus précisément le choix du service proposé à l'utilisateur pour l'envoi du SMS depuis l'ordinateur.

II.1 Les application mobiles

Il est nécessaire que l'utilisateur installe une application sur son smartphone pour réagir aux réception de SMS et aux demandes d'envoi de l'utilisateur.

Lors de la réception d'un SMS, l'application va recevoir une événement puis va lire le dernier SMS reçu. Une fois le contenu et l'expéditeur du SMS lus, l'application va envoyer le message "sur l'ordinateur" de l'utilisateur pour l'avertir de la réception.

Lors de la réception d'un message (contenu du SMS et destinataire) provenant de "l'ordinateur" de l'utilisateur, l'application va envoyer le SMS au destinataire. Cet envoi doit bien évidemment se faire de manière automatique et transparente.

Le schéma II.1 représente le fonctionnement global des applications mobiles qui seront utilisées dans notre projet.

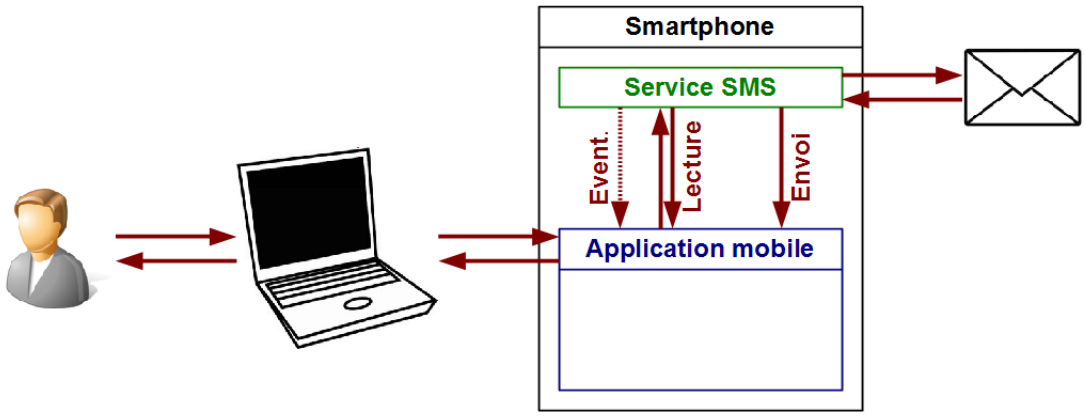

FIGURE II.1 – Applications mobiles : fonctionnement

II.2 Transfert du message

II.2.1 Protocole XMPP

II.2.1.1 Présentation

Jabber, maintenant appelé *XMPP*[1] (eXtensible Messaging and Presence Protocol) suite à sa standardisation, est un protocole de messagerie instantanée et de présence sur internet. Bien que peu connu du public, ce protocole possède de très nombreux avantages :

- standard ouvert : son fonctionnement est accessible par tous ;
- simple d'utilisation : les clients XMPP sont très simple d'utilisation car toute la complexité du protocole est situé coté serveur ;
- décentralisé ;
- confidentialité et sécurité ;
- ...

Le protocole XMPP est de plus en plus utilisé par les services de messagerie instantanée, tels que iChat sur les produits Apple, le chat Facebook, ou Gtalk de Google que nous utiliserons dans ce projet.

II.2.1.2 Bibliothèques

En raison de la nature dynamique, évolutive et ouverte du protocole XMPP, de très nombreuses bibliothèques sont disponibles (référencées sur le site officiel de la fondation) dans la majorité des langages de programmation.

1. Site web : http://xmpp.org/

Dans ce projet nous utiliserons *aSmack*, un portage sous Android de la bibliothèque Java *Smack*, ainsi que *XMPPFramework* une bibliothèque pour Objective-C.

II.2.2 Choix du protocole

La première raison qui nous a poussé à choisir le protocole XMPP pour échanger les messages entre l'ordinateur et le smartphone est le fait qu'il s'agit du protocole de base de GTalk. Initialement nous voulions envoyer et recevoir les messages depuis GTalk, comme nous l'expliquerons juste après dans la partie II.3.1. Ce choix nous a donc paru judicieux car l'application mobile pourra échanger facilement des messages avec l'ordinateur sans aucun problème d'adaptation.

La seconde raison est le fait que la grande majorité des personnes possèdent un compte Google (Gmail), du fait de l'importance de Google sur l'univers d'Internet. De plus les smartphones Android nécessitent l'association à un compte Google pour fonctionner (téléchargement d'application, mises à jour, ...), donc les utilisateurs n'auront pas à créer de compte Google ni même de compte XMPP.

II.2.3 Formatage

Un SMS contient plusieurs informations : le numéro de téléphone de l'expéditeur ou du destinataire et son contenu. Il va donc falloir définir un format à respecter dans lequel seront échangés les messages XMPP et qui va contenir l'ensemble de ces informations.

Il existe plusieurs formats de données comme le XML (Extensible Markup Language), le JSON (JavaScript Object Notation) ou encore YAML (YAML Ain't Markup Language). Nous avons décidé d'utiliser le *JSON* du fait de sa popularité "à la mode", son aspect peux verbeux et sa bonne lisibilité.

Voici l'exemple d'un message XMPP qui transitera du smartphone de l'utilisateur jusqu'à son ordinateur :

```
{
    "action": "receive-sms-action",
    "authorPhoneNumber": "0123456789",
    "recipient": "987654321",
    "body": "Ceci est le contenu du SMS"
}
```

La première ligne corresponde au type de message : "receive-sms-action" lorsque l'utilisateur reçoit un SMS, "send-sms-action" lorsqu'il souhaite en envoyer un. La seconde ligne correspond au numéro de l'auteur du SMS (facultatif lors d'un envoi). La troisième ligne correspond au

numéro du destinataire (facultatif lors de la réception). Et enfin la dernière ligne correspond au contenu du SMS.

II.2.4 Fonctionnement

Les messages sont échangés entre les deux clients XMPP : un présent dans l'application mobile et l'autre dans la solution présente sur l'ordinateur de l'utilisateur.

Le schéma II.2 décrit les entités qui échangeront des messages en utilisant le protocole XMPP.

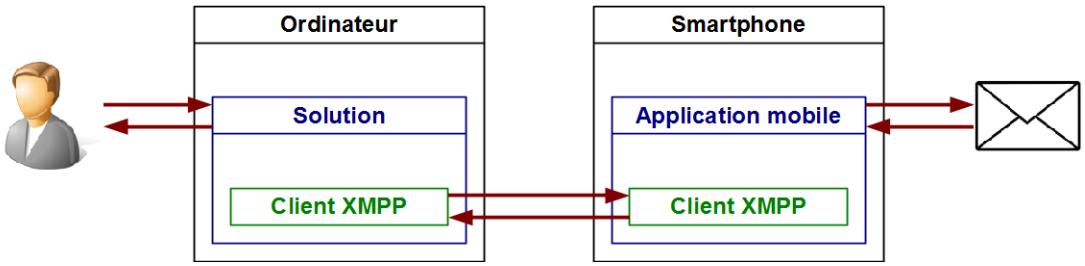

FIGURE II.2 – Transfert du message : fonctionnement

II.3 Service proposé à l'utilisateur

II.3.1 GTalk

Google Talk, aussi appelé GTalk, est le client de messagerie instantanée proposé par Google. Il est disponible à partir de la page web de Gmail, mais aussi en client Windows que l'on peut installer sur son ordinateur (Windows).

Initialement nous voulions utiliser ce client pour envoyer nos SMS pour la simple et bonne raison que de nombreux utilisateurs ont toujours le navigateur web ouvert, ainsi qu'un onglet avec leur boite mail. De plus, nous voulions nous inspirer d'une solution existante qui utilise GTalk pour l'approfondir et lui ajouter de nouvelles fonctionnalités.

II.3.2 Problèmes

II.3.2.1 Parler à soi-même

Le protocole XMPP autorise et offre la possibilité de s'envoyer des messages instantanés à soi-même si l'expéditeur et le destinataire sont les mêmes comptes.

Cependant GTalk ne le permet pas, pour au moins deux raisons :

- Pour envoyer un message à une personne il faut cliquer sur son lien dans la liste des contacts. Or notre propre adresse email n'y apparait pas même si l'on s'est ajouté dans nos contacts ;
- Gtalk n'affiche pas les messages que l'on s'envoie à soi-même, depuis un autre client XMPP comme une application mobile par exemple.

De ce fait, il est donc nécessaire d'utiliser un compte intermédiaire. On peut ainsi créer un compte spécial qui représentera notre smartphone, et qui sera utilisé uniquement par l'application. Le fonctionnement global de la solution se résumerait au schéma II.3.

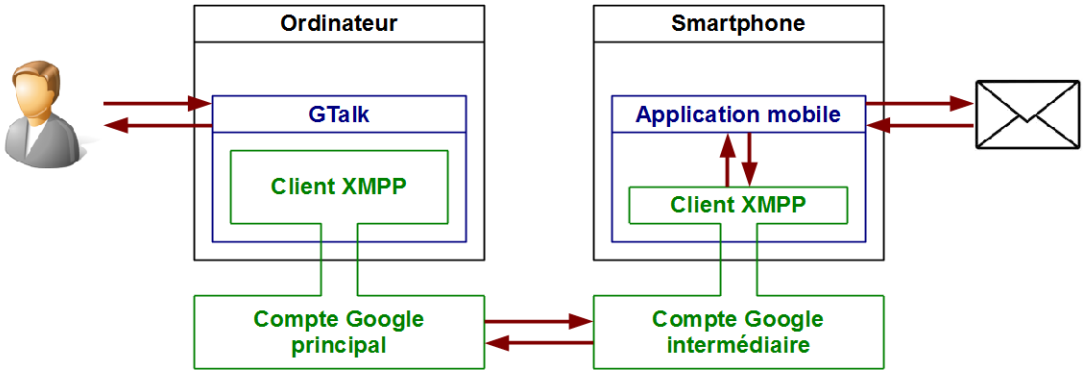

FIGURE II.3 – GTalk : Utilisation d'un compte intermédiaire

II.3.2.2 Praticité

Pour différencier les SMS que l'utilisateur voudra envoyer et les messages privés qui pourront transiter par le biais du compte intermédiaire, nous voulions imposer une "règle", c'est à dire un format comme par exemple :

```
sms 0123456789 Coucou, comment vas-tu ?
```

Mais cela imposera à l'utilisateur de saisir un message correctement formaté. De plus l'utilisateur devra saisir le numéro de téléphone du destinataire pour pouvoir envoyer le message, ce qui soulèvera les problèmes cités dans la partie I du rapport.

II.3.3 Solution envisagée

Suite aux inconvénients de GTalk mentionnés précédemment dans la partie II.3.2, nous avons dû nous orienter vers une autre solution, indépendante de GTalk. Plusieurs solutions sont possibles offrant chacune des avantages et des inconvénients.

Le client lourd est un logiciel autonome que l'utilisateur doit "installer" sur son ordinateur. Son fonctionnement est indépendant de tout serveur, hormis les échanges de données nécessaires

aux traitements. Pour notre projet, l'inconvénient de cette solution est l'obligation d'installation, ce qui peut poser problème lorsque l'on se trouve par exemple au travail ou que l'on utilise un ordinateur qui n'est pas le notre.

Le site web est une application fournie par un serveur distant et accessible depuis n'importe quel navigateur internet. Aucun programme ne devra être installé car cette solution est totalement indépendante de l'ordinateur de l'utilisateur. Le seule pré-requis est un accès à internet, qui, quelque soit le type de solution proposé à l'utilisateur (client lourd, client léger, ...), est obligatoire pour l'utilisation du protocole XMPP. C'est cette solution que nous utiliserons dans ce projet.

Les différentes étapes du fonctionnement du site web est représenté sur le schéma II.4

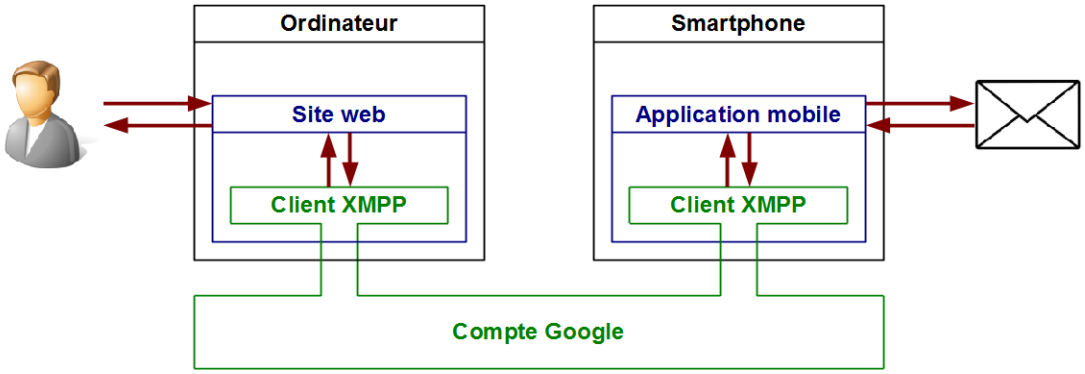

FIGURE II.4 – Site web : Réception d'un SMS

III Réalisation de la solution

Pour ce projet, nous avions initialement établi de pouvoir le faire fonctionner sur plusieurs architectures mobiles. Ayant des cours sur les deux plateformes phares actuelles, nous avons choisi d'étudier les mobiles de types iOS et Android. De plus, comme nous possédions chacun un téléphone sur les deux systèmes d'exploitation, il nous est apparu intelligent de se répartir le travail en travaillant chacun sur un OS. Nous détaillerons tout d'abord le développement du webservice puis celui des ces applications mobiles.

III.1 Site web

Pour pouvoir transmettre les messages depuis l'utilisateur vers le téléphone portable, nous avons choisi de mettre en place un site web à l'allure d'un gestionnaire de conversation.

Le fonctionnement de ce projet repose grandement sur la partie qui suit.

III.1.1 Choix du framework

Pour créer notre webservice, nous souhaitions utiliser des technologies récentes. Nous nous sommes alors confronté à deux choix complètement opposés.

Premièrement, utiliser le jeune Play Framework [1] qui est un framework d'application web RESTFul ce qui implique que le serveur ne gère pas d'état pour les clients. Il est basé sur les langages Java et Scala et permet le développement sur ces deux langages. Le deuxième candidat, Primefaces [2], est un framework basé sur les JSF. Il contient une bibliothèque graphique très riche.

III.1.2 Authentification avec OAuth 2.0

L'authentification des utilisateurs est un point sensible lors du développement d'un service car il implique souvent le stockage des mots de passes et donc de sécuriser l'application, veiller à corriger les failles de sécurité, ...

1. Site web : http://www.playframework.com/
2. Site web : http://primefaces.org/

III.1.2.1 OAuth 2.0

OAuth[3] est un protocole standard et libre d'accès aux données. Il permet l'authentification à une API en demandant à l'utilisateur les données qu'il souhaite partager. La gestion des utilisateurs se fait ensuite grâce à un "jeton", appelé "token" par la suite, unique délivré par le service tiers.

Beaucoup de services internet populaires implémentent ce système ce qui permet de aux utilisateurs de se connecter à un service en utilisant un de leurs comptes déjà existants. C'est le cas par exemple de Deezer ou de Stack Overflow qui proposent de s'identifier avec un compte Google, Facebook, Twitter, Windows Live ou autre. Ce protocole facilite donc la connexion pour le service qui l'utilise et les utilisateurs qui n'ont pas à créer un nouveau compte sur chaque nouveau site web par exemple.

Nous avons alors décidé d'utiliser OAuth 2.0 pour deux raisons importantes. La première concerne la gestion des utilisateurs sur notre service web qui n'aura pas lieu d'être, ce qui facilite le développement en laissant Google s'occuper de la sécurité, et le fait que nous voulions proposer une solution sans enregistrement des informations des utilisateurs. La seconde raison concerne l'accès aux données et aux services du compte Google, qui nous permettra l'accès aux contacts et à GTalk. OAuth s'est donc avérée comme la solution idéale pour notre projet.

III.1.2.2 Fonctionnement

L'authentification se déroule en plusieurs étapes présentées dans le schéma III.1.

Premièrement, l'utilisateur doit s'authentifier sur la plateforme de l'API proposant le service OAuth. Dans notre cas il s'agit de Google, comme le montre la capture III.2.

Ensuite l'utilisateur est redirigé vers la page des autorisations où il doit accepter les demandes de l'application. Dans notre cas, il s'agit de récupérer les contacts et d'utiliser le compte GTalk de l'utilisateur, comme le montre la capture d'écran III.3.

Si l'authentification réussie alors l'application se voit offrir un token. Elle doit alors l'échanger avec les serveurs de Google dans le but d'obtenir un token d'accès. Si tout se déroule correctement, elle obtient le token d'accès.

Lors de chaque requête pour récupérer des informations sur l'utilisateur, le webservice devra alors joindre son token d'accès à la requête pour avoir une réponse. De cette manière, nous pouvons récupérer des informations de l'utilisateur sans utiliser son mot de passe et son identifiant. Nous déléguons donc la sécurisation du mot de passe à Google.

3. Site web : http://oauth.net/

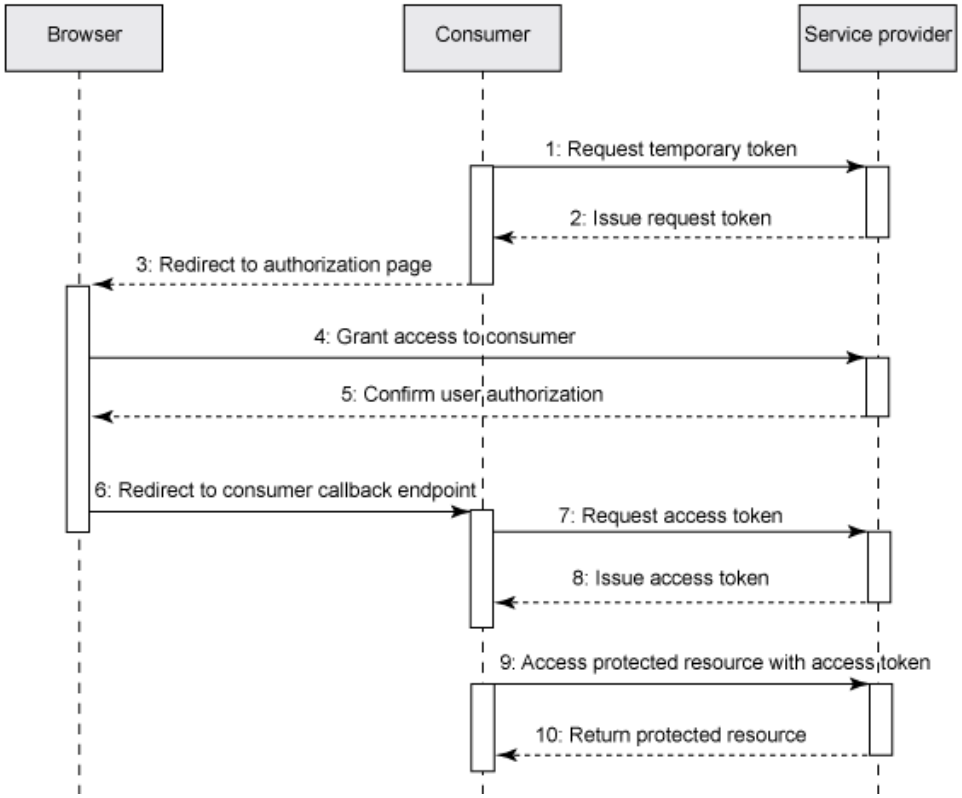

FIGURE III.1 – OAuth : fonctionnement
Source : http://www.ibm.com

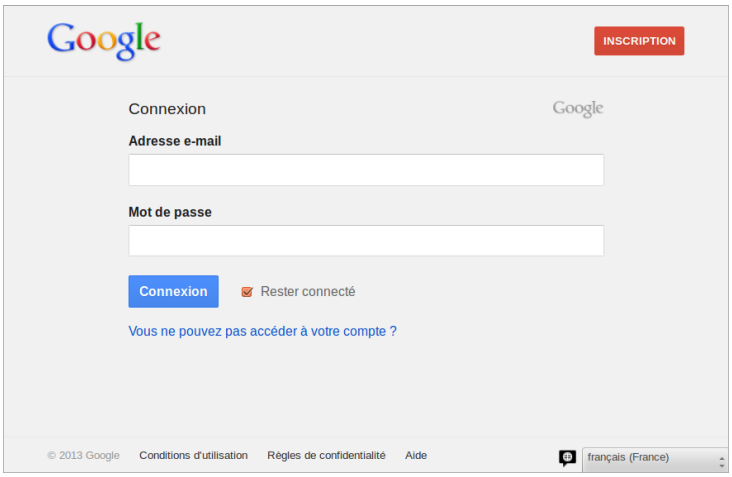

FIGURE III.2 – OAuth : authentification sur Google

Nous avons choisi d'utiliser l'authentification avec la plateforme Google car nous pouvions ainsi successivement, s'authentifier sur GTalk et récupérer les contacts de l'utilisateur.

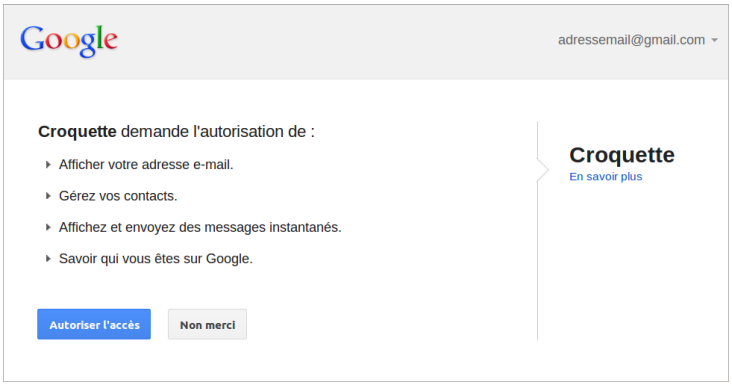

FIGURE III.3 – OAuth : demande d'autorisation

III.1.3 Gestion des contacts

La gestion des contacts de l'utilisateur est un point important de notre projet. De plus, notre cahier des charges spécifie leur gestion simple et transparente.

III.1.3.1 Téléchargement des contacts Google

Par défaut sous Android les contacts du téléphone sont dupliqués sur le serveur Google grâce au compte associé au smartphone, ce qui permet d'avoir une synchronisation sur tous les services Google dont Gmail.

Étant connecté sur notre site internet via son compte Google, notre application web va ainsi pouvoir récupérer la liste des contacts de l'utilisateur grâce au token délivré par Google lors de l'authentification.

III.1.3.2 Tri et Formatage

Le carnet de contacts de l'utilisateur peut contenir de nombreux types de numéros : téléphones mobiles, téléphones fixes, numéros étrangers, numéros surtaxés, ... Nous avons donc filtré la liste des contacts aux numéros de mobiles, c'est-à-dire ceux commençant par "0" ou "+33" suivi de "6" ou "7" et enfin de 8 chiffres.

Cela permet de ne pas surcharger la liste des contacts avec de nombreux numéros à qui l'on envoie jamais de SMS. De plus c'est un aspect sécuritaire car l'utilisateur ne pourra pas envoyer des SMS surtaxés depuis le site web. Un autre point important et le formatage des contacts. En effet nous allons systématiquement récupérer les contacts et les modifier afin de les rendre uniformes. Nous voulions avoir pour chaque contact, un nom et un prénom dont les premières lettres uniquement soit en majuscules. De plus, nous souhaitions récupérer plusieurs fois un contact qui possède plusieurs numéros de téléphones.

III.1.3.3 Google Guava

Une fois la liste des contacts téléchargée il est nécessaire de la stocker sur le serveur pour ne pas avoir à la télécharger après chaque rafraichissement de page. Il existe plusieurs méthodes permettant le stockage.

Une *base de données* permet de stocker efficacement des données. Nous avons voulu proposer aux utilisateurs une solution "sûre" dans laquelle aucune donnée ne sera enregistrée sur le serveur, ce qui implique que l'on n'utilisera pas cette méthode.

La *session* est une courte période pendant laquelle le serveur web communique avec le client (l'utilisateur). Durant cette période, une quantité de mémoire est allouée au client pour y stocker différentes informations, que l'on appelle "variable de session". Dans Play, la notion de "session" est quasiment inexistante en raison de sa nature RESTful.

Pour palier ce manque, Play propose l'utilisation du *cache*, qui est une zone mémoire à association clé-valeur commune à toutes les utilisateurs mais avec une durée de vie limitée. Lorsque le délai est écoulé, les données sont effacées automatiquement. Le problème de cette solution sont les nombreux tests à effectuer pour vérifier que les données sont toujours présentes et effectuer la mise à jour à chaque fois que le cache est vidé.

Nous avons utilisé la classe `CacheBuilder` de la bibliothèque Java *Guava* proposée par Google. Cette classe agit globalement comme un cache (association clé-valeur, durée de vie limitée) tout en proposant de nombreuses fonctionnalités. Il est notamment possible de définir une fonction qui sera exécutée lors de chaque "mise à jour" du cache, la durée de vie du cache, le nombre maximum de données, ...

III.1.4 Communication entre le serveur web et le navigateur du client

L'un des points clés du framework Play est sa capacité à gérer les websockets. Il fait en effet partie des rares frameworks offrant cette fonctionnalité relativement récente. Le protocole websocket vise à établir une communication bidirectionnelle et full duplex entre un navigateur web et un serveur. Cela signifie qu'un lien de communication s'établit entre les deux entités et permet la communication simultanée dans les deux sens.

III.1.4.1 Websocket

Pour ce projet, les fonctionnalités des websockets étaient une véritable aubaine car elles nous permettaient lors d'un changement d'état coté serveur d'avertir le client web. Dans notre cas, le serveur souhaite avertir le client lors de la réception d'un SMS et le client souhaite avertir le serveur lors de l'envoi d'un message XMPP. Les websockets sont donc adaptées à cette situation. Elles sont la clé de voute de ce projet car elles s'occupent des communications entre le client et le serveur.

Une fois que l'utilisateur s'est authentifié, plus aucun rechargement de pages ne sera fait. Tout est géré dynamiquement via les websockets.

La prise en charge du coté serveur se fait avec le framework Play qui gère nativement les websockets. Coté client, c'est le navigateur qui à l'aide du langage JavaScript va gérer les websockets. Celles-ci sont en cours de standardisation du coté du W3C et les différents navigateurs les implémentent donc chacun à leur façon.

En ce qui nous concerne, nous avons choisi de ne travailler que sur les navigateurs Mozilla Firefox et Google Chrome. Ce projet est donc inutilisable aujourd'hui sur d'autres navigateurs.

III.1.4.2 Notifications

Les messages XMPP et SMS ne sont pas les seuls à transiter sur les websockets. En effet, nous avons implémenter de la même manière un système de notification de nouveau message. Le système est simple, lors de la réception d'un nouveau message, si l'utilisateur ne dialogue pas au même moment avec l'auteur du nouveau message, une notification va s'afficher. Comme le montre l'image III.4, le contact qui a envoyé le message verra un compteur apparaitre à coté de son nom. Cela permet simplement d'avertir l'utilisateur courant du nombre de messages non lus envoyés par un de ses contacts.

FIGURE III.4 – Site web : notifications

III.1.5 Design du site

Le design du site est l'un des thèmes que nous avons travaillés pour cette partie la du projet. Nous souhaitions initialement avoir une interface simple, sans fioriture car cela nous semblait secondaire. Nous avons donc utilisé deux frameworks pour nous faciliter la tache et accélérer la mise en place du design.

Nous avons utilisé les deux frameworks qui suivent.

III.1.5.1 Twitter Bootstrap

Twitter Bootstrap[4] est un framework web utilisé pour le développement front end. C'est un outil écrit en JavaScript et CSS qui permet de réaliser facilement des interfaces élégantes néanmoins simples.

Il nous a été utile principalement pour pouvoir concevoir l'affichage des zones d'entrée de textes ou encore pour organiser l'affichage des contacts proprement.

III.1.5.2 jQuery Layout

jQuery Layout[5] est aussi un framework web qui, contrairement à Twitter Bootstrap, permet d'organiser les sites en plusieurs interfaces distinctes. L'utilité de JQuery Layout réside dans sa possibilité à fractionner un espace en plusieurs fenêtres. Cela permet de retrouver l'ergonomie et le coté simple et ordonné des logiciels.

Ce framework très riche nous a donc été très profitable pour pouvoir obtenir une bonne ergonomie sans avoir à trop passer du temps sur cette partie. Le schéma III.5 représente l'agencement général de notre site web grâce à ce framework.

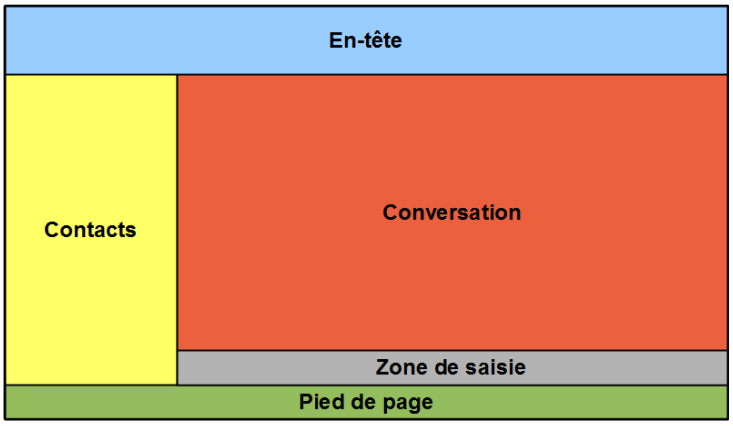

FIGURE III.5 – jQuery Layout : structure du site web

III.2 Android

III.2.1 Fonctionnement global

Le fonctionnement global du projet sur la plateforme Android est relativement simple. Nous souhaitons simplement pouvoir utiliser le téléphone portable comme un intermédiaire entre notre

4. Site web : http://twitter.github.com/bootstrap/
5. Site web : http://layout.jquery-dev.net/

site web et le correspondant avec qui nous discutons. Pour cela, le téléphone fait office de proxy entre les deux entités éloignées. Un proxy est un composant qui se place entre un interlocuteur et son auditeur. Le proxy sert principalement à relayer l'information d'un élément vers un autre, il sert d'intermédiaire. Dans notre cas, le proxy s'occupe de recevoir des SMS et de les rediriger vers le site web. Réciproquement, il récupère les messages envoyés par le site web et envoie un SMS vers le correspondant défini dans le message reçu.

III.2.2 Création du service

Les applications fonctionnant sous Android régissent par un principe simple, celui des "activités". Pour simplifier, une activité représente une fonctionnalité graphique d'une application. Par exemple, lorsqu'une application est ouverte, on tombe sur un menu avec différentes possibilités d'actions qui nous sont offertes. Le menu représente donc ici une activité.

Il faut savoir qu'une activité, lorsqu'elle est active, occupe le système qui attend des événements. Si l'utilisateur ne fait rien alors que le menu est affiché, le système peut s'occuper d'autres tâches annexes. Si en revanche l'utilisateur navigue dans le menu, alors l'activité est considérée comme en fonctionnement et bloque toutes autres opérations non relatives à l'application exécutée.

Dans le cadre de notre application, nous devons à son lancement, effectuer une série d'opérations afin de mettre l'application en fonctionnement. Cela consiste notamment, à vérifier l'état de la connectivité internet, récupérer les identifiants et se connecter sur le service GTalk. Ces opérations rendent le système bloquant durant leur exécution. Cela n'est pas un problème lorsque la connexion s'effectue sans erreur. Le traitement durant moins d'une seconde, le système n'est pas gravement impacté. En revanche, si une erreur intervient, le programme va alors essayer de la résoudre. Cela n'est pas concevable. En effet, une perte soudaine de réseau pourrait par exemple mettre l'application dans un état de recherche ce qui est potentiellement très lent. Cette situation rendrait le terminal complètement bloqué durant la recherche.

Ce comportement n'étant pas souhaitable, nous avons dû trouver une solution pour pouvoir outrepasser ce blocage.

Nous avons alors utilisé le concept Android de "services". Celui-ci consiste à exécuter une fonctionnalité de manière asynchrone. Le service n'impacte pas le système, au contraire il fonctionne avec lui.

Comme montré dans la figure III.6, le service n'est pas bloquant. Lors de son exécution, le système continue de fonctionner sur d'autres tâches en paire avec le service lancé.

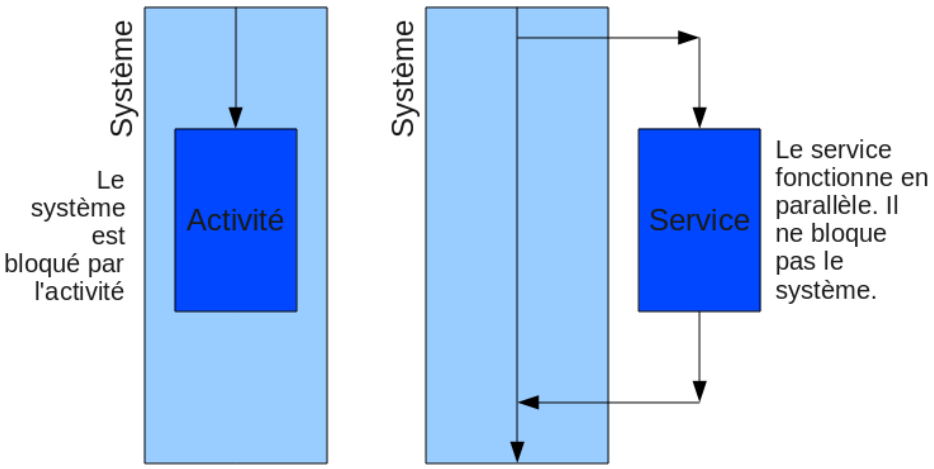

FIGURE III.6 – Fonctionnement d'un service Android

Dans notre cas, le service sert à deux tâches principales. Tout d'abord, il va configurer l'authentification de l'utilisateur sur GTalk. Ensuite il va simplement se mettre en attente d'événements et le cas échéant, traiter ce même événement.

III.2.3 Authentification sur GTalk

Pour pouvoir converser avec le site web, l'application a besoin de s'authentifier auprès du service GTalk. Pour cela, nous sommes passés par deux étapes.

Initialement, nous utilisions la méthode classique d'authentification. Lorsque l'utilisateur lance l'application, celle-ci lui demande de rentrer ses identifiant et mot de passe. Cela effectué, l'application se charge de se connecter et de s'authentifier sur les serveurs GTalk en utilisant les paramètres rentrés par l'utilisateur.

Cette solution est la plus simple à implémenter mais n'est malheureusement pas la plus agréable à utiliser ni la plus sure. Nous forçons en effet l'utilisateur à rentrer ses identifiants ce qui pourrait paraître légitime mais reste contraignant pour l'utilisateur. Celui-ci ne sait pas ce que nous faisons avec son mot de passe. Une application malveillante pourrait récupérer les identifiants et s'en servir à des fins illégales. Dans notre cas, même si nous n'avions aucune mauvaise intention, nous ne pouvons demander aux utilisateurs de nous faire confiance. Nous avons donc opté pour une deuxième solution, l'identification avec OAuth 2.0.

Comme détaillé dans la partie III.1.2, OAuth permet de réaliser des actions en agissant à la place de l'utilisateur. Ici nous souhaitions simplement utiliser le compte GTalk de l'utilisateur sans que celui-ci ait à rentrer ses identifiants.

Contrairement à l'authentification sur le site web, l'authentification sur une plateforme Android est théoriquement plus simple. En effet, une fois que l'utilisateur a donné son accord, le serveur d'authentification de Google va directement envoyer le token d'authentification. Il n'y a alors pas besoin d'effectuer une deuxième étape pour récupérer ce dernier comme le montre le diagramme III.7.

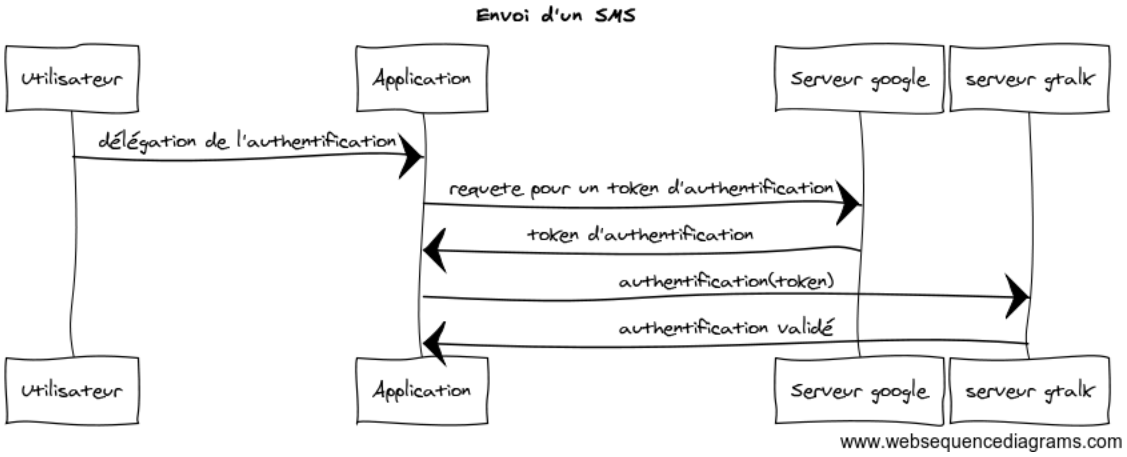

FIGURE III.7 – Procédure d'obtention d'un token avec un appareil Android

Néanmoins, une des difficultés d'implémenter ce protocole est que la bibliothèque de gestion de compte XMPP pour Android dénommée Asmack est différente de celle utilisée pour le site web. Asmack est un fork de la bibliothèque Smack pour Android et est l'unique bibliothèque gérant les comptes XMPP présente sur Android. Ne souhaitant pas ré-implémenter toutes les fonctionnalités présentes dans celle-ci, nous avons choisi de l'utiliser. Malheureusement, les différences entre les deux bibliothèque nous ont empêché de ré-implémenter notre précédent travail effectué pour le site web qui était fonctionnel.

Le langage Java fonctionne sur une machine virtuelle. Celle-ci contient des fonctionnalités basiques sur lesquelles le langage peut s'appuyer. Dalvik, la machine virtuelle pour Java présente sur Android n'est pas exactement la même que la machine virtuelle basique. En effet, pour des soucis de rapidité et de légèreté, certaines parties de Dalvik ont été réécrites pour en faire une machine virtuelle plus adaptée aux plateformes mobiles.

Smack a été développé sur la machine virtuelle Java basique tandis que Asmack a été développé sur Dalvik. Cette différence implique des changements quant aux fonctionnements de celles-ci.

Finalement, en ré-implémentant certaines fonctionnalités, nous avons réussi à nous authentifier avec OAuth.

Authentifier un utilisateur via OAuth2.0 sur Android a été un réel challenge. Il nous a fallu comprendre et trouver des équivalences au fonctionnement de Smack.

III.2.4 Envoi et réception d'un SMS

III.2.4.1 Réception d'un SMS

Une fois authentifié, nous souhaitons pouvoir utiliser notre téléphone comme proxy. Notre application doit en effet servir d'intermédiaire entre le site web et un correspondant. Pour cela elle doit tout d'abord surveiller l'arrivée de nouveaux messages.

Google autorise la surveillance du comportement du téléphone sur Android, permettant de réagir lors de différents événements. Dans notre cas, nous avons pu utiliser le principe de "Broadcast Receiver" du système. Un broadcast receiver est un outil qui permet de rendre une application consciente de son environnement. Cela permet généralement à l'application qui l'utilise d'être prévenue lors de nouvelles notifications ou d'un comportement particulier du téléphone. Son fonctionnement est simple : il permet de s'enregistrer auprès du téléphone qui maintient une base de donnée des membres à avertir lors d'un changement particulier.

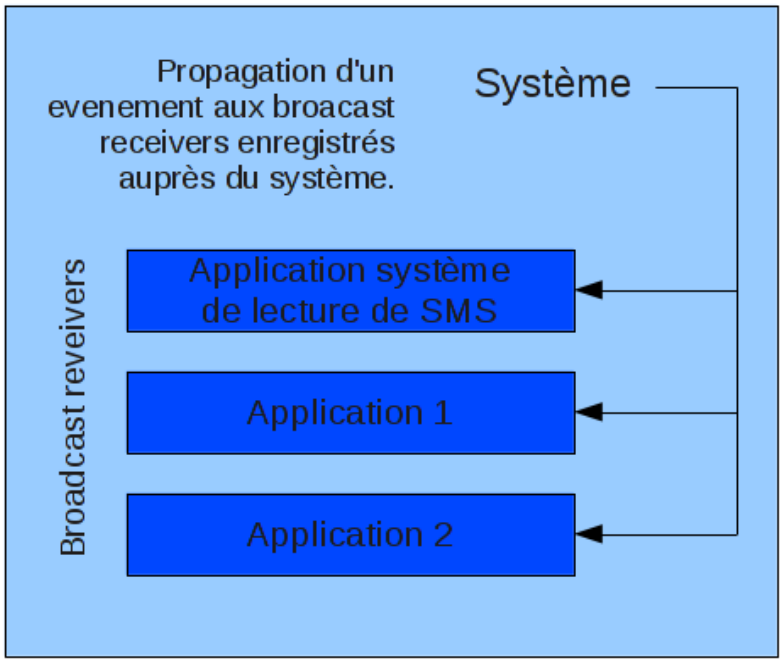

FIGURE III.8 – Android : fonctionnement des broadcasts receivers

Le schéma III.8 permet de voir le système du téléphone qui contient les différents broadcast receiver à avertir en cas de nouveaux messages. Ici l'application de rédaction de SMS ainsi que deux autres applications seront averties.

Dans notre cas, nous nous en sommes servis pour d'une part être avertis de l'arrivée de nouveaux SMS et d'autre part pour pouvoir récupérer le dit message et le traiter.

III.2.4.2 Envoi d'un SMS

L'envoi quant à lui est beaucoup plus trivial. Android met à disposition un outil permettant d'envoyer des messages.

III.2.5 Fonctionnement

III.2.5.1 Globalité

Le fonctionnement global de notre application est relativement simple. Elle a été tout d'abord implémentée de manière à répondre aux besoins du projet et être fonctionnelle pour donner une première version.

Lors de la réception d'un SMS sur le smartphone l'application va tout d'abord récupérer le contenu du message ainsi que son auteur. Puis les données vont être encapsulées dans un message au format JSON. Enfin le message est envoyé vers le webservice grâce à GTalk, qui s'occupera de notifier à l'utilisateur l'arrivée d'un nouveau message.

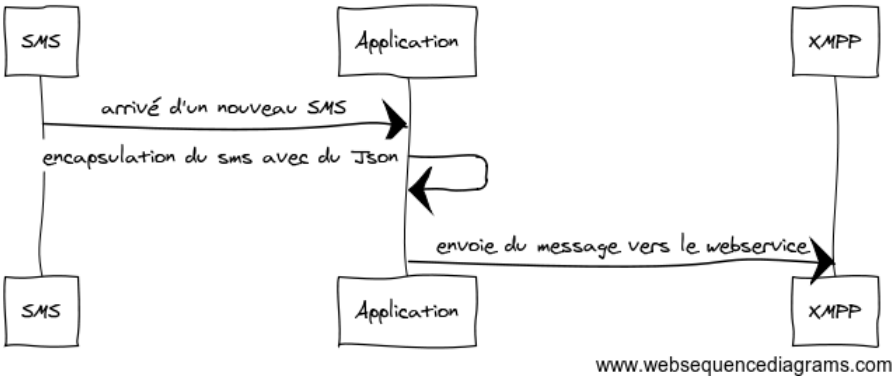

FIGURE III.9 – Traitements lors de la réception d'un SMS

Lors de la réception d'un nouveau message XMPP la méthode est l'inverse de celle de la réception d'un SMS. L'application va désencapsuler le message XMPP pour y récupérer l'information utile. Cela fait, notre application va ensuite envoyer le SMS au destinataire.

III.2.5.2 Architecture

Dans le cadre de notre projet, nous n'avions comme fonctionnalité de l'application que l'envoi de SMS. Lors de la réception d'un message XMPP, nous souhaitions pouvoir réagir différemment en fonction du contenu du message. Notre travail respectait donc ce principe mais ne permettait aucune évolutivité.

A supposer qu'un tiers soit intéressé par la communication entre le webservice et le téléphone par l'intermédiaire de XMPP, mais que son but ne soit pas d'envoyer de simple SMS, nous

avons implémenté une architecture permettant de rajouter facilement de nouvelles fonctionnalités. Pour donner des exemples, un utilisateur pourrait décider d'implémenter une fonctionnalité lui permettant de déclencher la capture de photos de son téléphone à partir du webserver.

Pour permettre cette évolutivité, nous avons réalisé une architecture qui s'adapte automatiquement en fonction des messages reçus. Il s'agit ici des patrons de conceptions fabrique et stratégie.

Pattern stratégie

Le patron de stratégie est un patron de conception de type comportemental. Il permet d'adopter dynamiquement un comportement particulier en fonction des conditions de son exécution. Concrètement, quelque soit la classe implémentant l'interface de base, l'exécution d'un algorithme s'effectue simplement depuis une seule méthode. Le schéma III.10 décrit le fonctionnement du modèle.

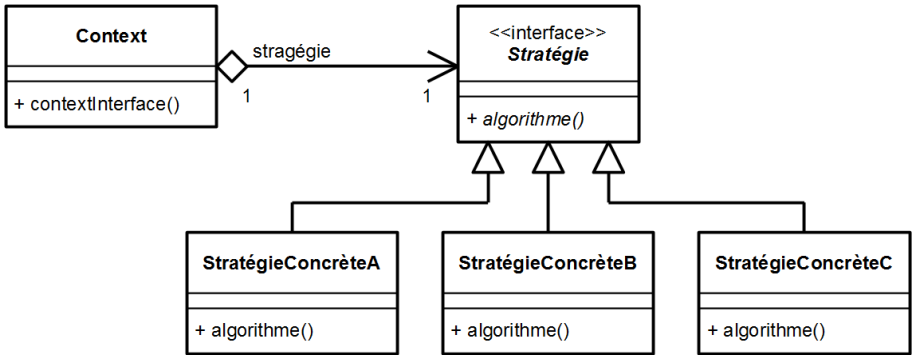

FIGURE III.10 – Pattern stratégie : diagramme UML

Dans notre cas, il permet de créer différents types de stratégies qui pourront potentiellement être utilisés. Notre projet ne contient pour l'instant que les stratégies "envoi de SMS" et "envoi de messages XMPP", mais avec cette architecture il est possible de rajouter d'autres stratégies, comme par exemple l'extinction du téléphone ou encore l'envoi d'un email par exemple.

Pattern fabrique

L'avantage du pattern stratégie est qu'il permet de regrouper les algorithmes à exécuter dans des classes séparées plutôt que de le mélanger dans du code différent. Mais son inconvénient réside dans le fait qu'il est nécessaire d'effectuer un test sur le type d'algorithme à exécuter à chaque endroit où l'on doit instancier les stratégies.

Le pattern fabrique permet de palier ce problème en définissant les règles d'instanciation des différentes stratégies. Ainsi, les différents tests ne se retrouvent pas dupliqués à chaque partie du code, mais regroupé dans une classe qui se chargera de définir la stratégie à appliquer en fonction

des données qu'on lui transmet. Le schéma III.11 représente le diagramme de classe du pattern fabrique.

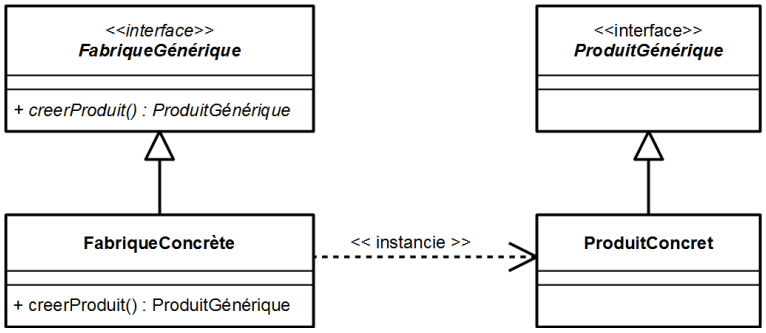

FIGURE III.11 – Pattern fabrique : diagramme UML

Dans notre cas, la fabrique va analyser chaque nouveau message. En fonction de leur contenu elle devra décider d'une action à effectuer sur le smartphone.

Ajout de fonctionnalités

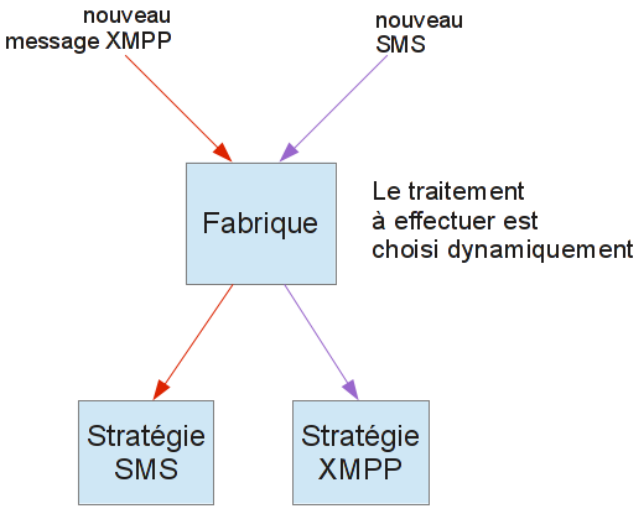

FIGURE III.12 – Pattern fabrique : fonctionnement dynamique

La figure III.12 montre le fonctionnement du projet lors de l'arrivée d'un nouveau message (SMS ou XMPP). L'intérêt principal de ce pattern est l'évolutivité qu'il apporte. Le choix dynamique à l'exécution de la stratégie à un intérêt pour le développeur. En effet, celui-ci n'a plus besoin de se soucier de l'implémentation du choix de la stratégie ni de son exécution.

L'ajout d'une fonctionnalité ne se fait plus dans chaque partie du code source, mais en définissant une nouvelle stratégie (classe représentative) puis en ajoutant sa définition dans la

fabrique.

III.3 iOS

III.3.1 Objective-C et Xcode

Objective-C est un langage de programmation orienté objet et réflexif. Compilé et libre, il hérite du langage C (comme le C++) en lui rajoutant l'aspect objet, ce qui lui permet d'être interfacé avec du Langage C ou du C++. Principalement présent sur les systèmes d'exploitation des produits de la marque Apple (OS X et iOS), ce langage est multi-plateforme.

Xcode est l'environnement de développement intégré (EDI, ou IDE pour Integrated Development Environment) pour Mac OS X. Il permet le développement d'application en Langage C, C++ et bien évidemment en Objective-C pour OS X ou iOS. C'est un IDE très complet qui comporte un éditeur de texte, un débogueur, un compilateur, un gestionnaire de projet et de version, un gestionnaire de documents pour la documentation, et d'autres outils facilitant le développement.

III.3.2 Les frameworks

Un *framework* est un ensemble d'outils et de composants structurels qui permettent le développement de logiciels ou de sites web.

Apple propose de nombreux frameworks permettant de simplifier la tâche des développeurs. Ils respectent les *design patterns* pour structurer l'agencement des différents objets, comme par exemple l'architecture MVC des interfaces graphiques. Ces frameworks font la force des applications iOS car ils permettent d'interagir très facilement avec les composants des terminaux, comme par exemple le gyroscope, l'appareil photo, l'envoi de mail, l'affichage d'une carte, ...

Il existe deux types de frameworks : les frameworks dits publiques et les frameworks dits privés.

Les frameworks *publics* doivent être documentés et approuvés par Apple pour que les applications qui les utilisent puissent être publiables sur l'Apple Store. Cette politique très ferme d'Apple permet d'éviter toute faille de sécurité dans le système et permet de proposer des applications sûres aux utilisateurs.

Les frameworks *privés* proposent des outils de bas niveau non-documentés. Apple en dispose de plusieurs dans leur kit de développement mais ceux-ci ne peuvent pas être directement utilisés : il est nécessaire d'importer "à la main" des fichiers d'entête (headers ".h") pour pouvoir compiler le projet.

III.3.3 Manipulation des SMS

III.3.3.1 Envoi de SMS

Apple n'autorise l'envoi de SMS qu'en utilisant leur framework officiel *MessageUI*. Lorsque l'application désire envoyer le SMS une fenêtre s'affiche et l'utilisateur doit confirmer l'envoi en appuyant sur le bouton prévu à cet effet. Cette méthode ne peut pas être utilisée dans ce projet, car l'application mobile fonctionne en tâche de fond et doit envoyer les SMS de manière automatique.

Cependant l'utilisation du framework privé *CoreTelephony* le permet. [iMessage] L'envoi de SMS peut ainsi se faire grâce à une ligne de code et de manière totalement automatique.

III.3.3.2 Réception de SMS

La réception des SMS se décomposent en deux étapes : l'événement de réception, puis la lecture.

Événement de réception

Le système envoie des notifications lors des appels ou de la réception des SMS. Pour que notre programme reçoive ces notifications il est nécessaire de définir une fonction qui sera appelée lorsque ces événements seront émis. Cela se fait grâce à la fonction `CTTelephonyCenterAddObserver()` dans laquelle un des paramètres correspond à la fonction appelée.

Lecture des SMS

Les SMS sont stockés dans une base de données SQLite dont le fichier est `/private/var/mobile/Library/SMS/sms.db` dans le répertoire du terminal. Pour parcourir les messages, il est nécessaire d'effectuer des requêtes SQL comme dans n'importe quelle base de données.

Mais un problème critique survient : il est impossible d'accéder à ce fichier depuis une application. En effet, chaque application iOS fonctionne séparément dans sa *sandbox*, qui est un environnement virtuel clos permettant la protection contre les processus malveillants. Le schéma III.13 représente les différents répertoires des sandbox.

Malheureusement aucune solution d'accès au fichier n'a été trouvée pour le moment. En attendant de trouver une solution, nous avons copié le fichier dans un répertoire accessible pour effectuer les différents tests de lecture, et la fonction permettant la lecture des SMS renvoie des données "statiques" pour valider le fonctionnement de l'application.

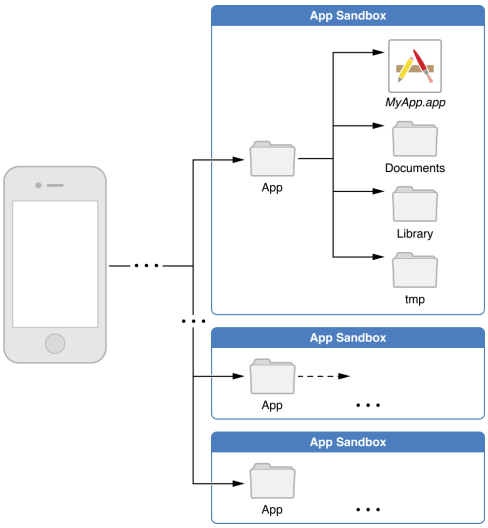

FIGURE III.13 – Sandbox sous iOS
Source : http://developer.apple.com

III.3.4 Les messages XMPP

Pour communiquer avec le site internet, le client XMPP de l'application a été développé grâce à *XMPPFramework*[6]. Il s'agit d'un framework libre et open-source développé et maintenu par principalement Robbie Hanson. XMPPFramework est développé en Objective-C destiné aux applications OS X ou iOS.

Le client XMPP fonctionne comme celui sous Android. Lors de la réception d'un SMS l'application va formater le message sous le format JSON puis envoyer un message XMPP au propre compte Google. Lors de la réception de messages XMPP l'application va récupérer les données stockées sous le format XMPP puis va envoyer le SMS.

Dans l'application mobile, nous n'utiliserons qu'un seul client XMPP et nous voulions qu'il persiste durant toute la vie de l'application (jusqu'à ce que l'utilisateur tue le processus). Le patron de conception (design pattern) *singleton* a donc été utilisé dans le projet iOS. En effet singleton permet de s'assurer de n'avoir qu'une seule instance qui persiste en mémoire, et qui de plus est accessible dans toute l'application. Le schéma III.14 représente le diagramme de classe du singleton.

6. Source : https://github.com/robbiehanson/XMPPFramework

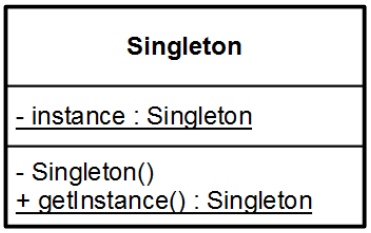

FIGURE III.14 – Pattern singleton : diagramme de classe

IV Résultats - Discussion

Nous avons pu terminer le projet dans les temps seulement, même si celui-ci est aujourd'hui fonctionnel, il persiste encore certains problèmes. Nous avons toujours essayé de les résoudre de la manière la plus efficiente qui soit. Cependant, il nous est arrivé de bloquer sur des parties pour lesquelles aucune solution propre n'était réalisable. Nous avons alors du appliquer des manipulations qui empêcheront ce projet d'être un projet considéré comme stable dans un environnement grand public.

IV.1 Protocole de transfert

Le choix de prendre GTalk comme moyen de communication n'était pas sans conséquences. En effet, il nous est apparu impossible d'avoir une gestion fine de GTalk.

IV.1.1 Réception et traitement

Lors de l'envoi d'un message XMPP d'un compte A vers un compte B, toutes les interfaces de connexion de B recevront le message. Alors l'envoi d'un message XMPP d'un compte A vers ce même compte A, le message sera aussitôt reçu par A.

Dans notre cas, le fait d'envoyer un message XMPP sur notre propre compte revient à recevoir et traiter ce même message que l'on vient d'envoyer. Bien heureusement, nos applications filtreront les messages qui ne leurs sont pas destinés.

Comme le montre l'image IV.1, une boucle se crée. Dans les faits, elle ne causera jamais de problème, néanmoins c'est un comportement qui n'était pas désiré et qu'on aurait souhaité éviter.

IV.1.2 "Broadcast"

Les messages XMPP ne se limitent pas aux quelques clients présents dans notre projet : site web et applications mobiles. Tous les messages que nous envoyons sont aussi redirigés vers les outils utilisant GTalk.

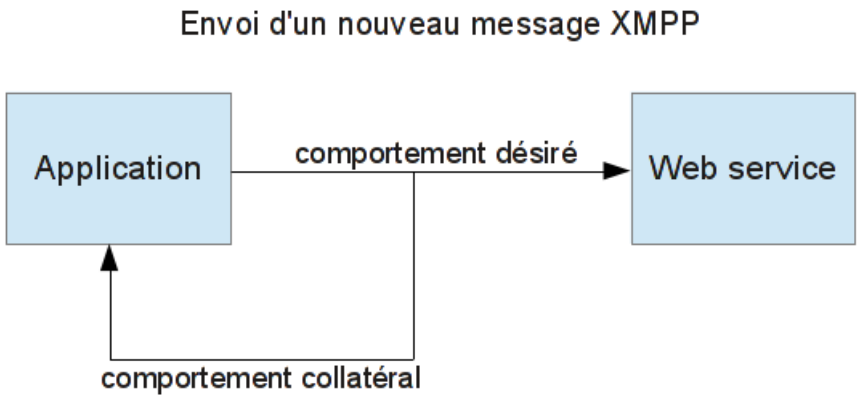

FIGURE IV.1 – Création d'une boucle lors de l'envoi d'un message XMPP

En pratique, cela impliquait de recevoir des messages au format JSON sur plusieurs lieux indésirables.

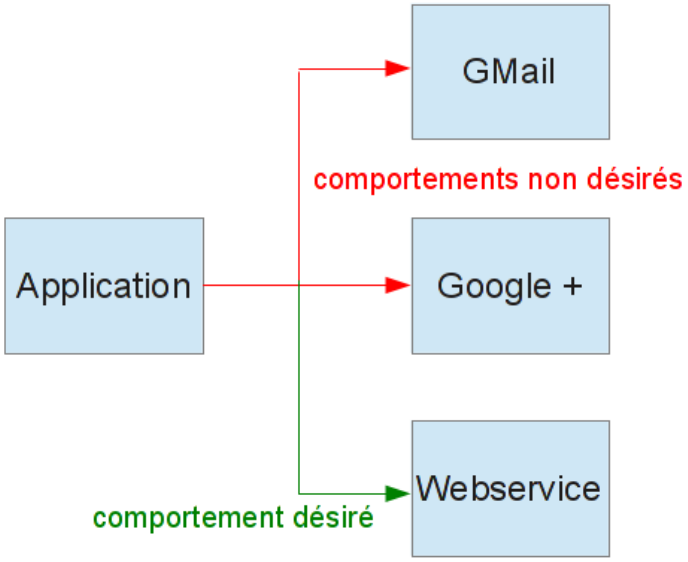

FIGURE IV.2 – Destinataires lors d'un envoi d'un message XMPP

Comme le montre le schéma IV.2, lors de l'envoi d'un message XMPP originellement destiné uniquement à une application ou au web service, c'est tout l'environnement qui reçoit et affiche le nouveau message. Le résultat est montré dans l'image IV.3, un message XMPP au format JSON.

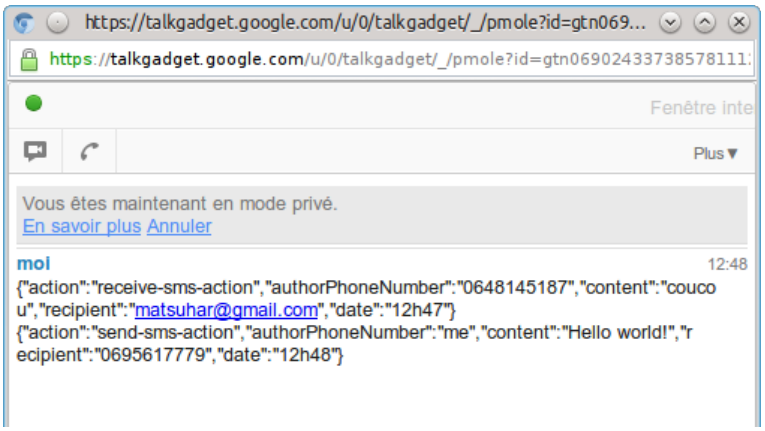

FIGURE IV.3 – Message XMPP au format JSON reçu avec GTalk

IV.2 Site web

Concernant le site web, les résultats obtenus sont probants. Néanmoins, plusieurs fonctionnalités auxquelles nous avions pensé manquent.

IV.2.1 Gestion des contacts

IV.2.1.1 Cahier des charges

Notre cahier des charges requérait une gestion simple et transparente des contacts pour l'utilisateur.

Le téléchargement des contacts du compte Google couplé avec l'affichage de la liste des noms de ces contacts, permet à l'utilisateur d'envoyer des SMS très simplement sans avoir à spécifier le numéro de téléphone.

IV.2.1.2 Problème

Comme expliqué dans la partie III.1.3.1 les contacts présents sur les smartphones Android sont synchronisés avec le même compte Gmail. Sous iOS cette fonction existe mais n'est pas associé à Google : Apple utilise iCloud qui permet aux utilisateurs une synchronisation automatique des contacts sur tous les appareils Apple associés à l'adresse mail. Ainsi par défaut il n'est pas possible d'accéder aux contacts de l'iPhone depuis le site web.

Contrairement à Google qui effectue ses authentifications avec OAuth, Apple ne propose pas d'API permettant l'authentification sur iCloud. La solution trouvée a donc été de synchroniser les contacts de l'iPhone avec un compte Google, au détriment de la synchronisation d'iCloud.

IV.2.1.3 Amélioration possibles

Filtres

Dans notre site web nous avons filtré la liste des contacts à ceux présents dans notre carnet d'adresses et en nous limitant aux numéros de téléphone portable (06.xx.xx.xx.xx ou 07.xx.xx.xx.xx). Or il est possible de recevoir des SMS venant de numéro "courts" (8.xx.xx), de numéros de téléphone fixe, ou bien encore de contacts non présents dans nos contacts.

Notre version actuelle ne permet pas de résoudre ce problème. Une solution serait d'ajouter un filtre sur la page web et d'enregistrer tous les contacts ainsi que ceux non présents dans Google.

Anciens SMS

Lorsque l'on se déconnecte du site web toutes les conversations sont supprimées du serveur en raison de la politique de sécurité que nous avons voulu garder.

Nous avons pensé apporter une amélioration à notre site web dans la prochaine version, qui aurait pour objectif de télécharger les anciens messages échangés avec un contact. Ces SMS transiteraient aussi par le protocole XMPP, soit un par un, soit sous forme de liste selon les performances et la taille maximale des messages XMPP.

IV.2.2 Le cache

La *session* est une courte période pendant laquelle le serveur web communique avec le client (l'utilisateur). Durant cette période une quantité de mémoire est allouée au client pour y stocker différentes informations, que l'on appelle "variable de session".

Dans le framework Play que nous avons utilisé la notion de "session" est quasiment inexistante en raison de sa nature "RESTful", c'est à dire qui ne conserve pas de données liées aux clients durant la session. Play propose l'utilisation de *caches*, mais ceux-ci sont accessibles depuis chaque utilisateur et possèdent une durée de vie limitée.

Ainsi nous avons été obligés de contourner le problème en utilisant des méthodes "peu orthodoxes" ce qui rend l'application peu maintenable. De plus,cela ne facilite pas l'ajout de fonctionnalités. L'utilisation d'un framework plus adapté aurait été un choix plus judicieux.

IV.3 Applications mobiles

Concernant tout d'abord les applications mobiles, certaines contraintes imposées par les systèmes d'exploitations nous ont empêchés de construire une application qui marche sur n'importe quel téléphone.

Au vu des techniques employées, il nous parait peu probable de pouvoir publier aucune des deux applications sur les "markets" officiels de Apple et Google.

IV.3.1 Android

Un autre souci que nous avons rencontré sur la plateforme Android est la limite de SMS. Pour empêcher les applications de spammer à la place d'un utilisateur, Android a mis en place une limite d'envoi de SMS pour une application. Toute application ne peut envoyer plus de 100 SMS par jour. Une fois la limite atteinte, l'application est bridée. Chaque nouveau SMS envoyé doit alors faire l'objet d'une confirmation par l'utilisateur ce qui rend le projet inutile car nous perdons l'avantage de pouvoir se passer du téléphone portable.

Malgré nos recherches sur le sujet, il s'est avéré impossible de modifier cette limite en passant par des solutions réglementaires. La limite est définie dans la base de données système du système d'exploitation qu'on ne peut modifier sans avoir les droits Root.

Les droits Root correspondent simplement à avoir le droit de tout modifier sur son système avec les risques que cela comporte. Il faut savoir que pour obtenir ces droits, il faut rooter son téléphone. Bien que rooter son téléphone soit légal en soit, il comporte certains risques. Une mauvaise manipulation peut engendrer des dommages sur le téléphone. De plus, l'action en elle même n'est pas à la porté des débutants.

Ces principales raisons font que notre application ne fonctionnera correctement que sur les téléphones possédant les droits root. Cela implique que l'utilisateur devra être capable de manipuler son téléphone.

Pour pouvoir outrepasser ces contraintes, il est nécessaire de modifier le comportement par défaut du système. Nous avons utilisé un éditeur de bases de données SQLite avec lequel nous avons modifié une clé particulière. Il s'agit de la clé sms_outgoing_check_max_count que nous avons positionnée à un nombre négatif afin de ne plus avoir de limite d'envoi de message.

Cela modifié, nous sommes alors en mesure d'envoyer autant de messages que désirés.

IV.3.2 iOS

IV.3.2.1 Généralité

L'utilisation des frameworks privés rendra impossible la publication de notre solution sur l'Apple Store. Ceci pourra réduire fortement le nombre d'utilisateurs car ceux-ci devront installer l'application manuellement.

Compte tenu des problèmes rencontrés et des solutions utilisées, il est clair que l'application iOS nécessite de nombreuses améliorations. La lecture des SMS, imposant de lire un fichier hors

de la Sandbox comme expliqué dans la partie III.3.3.2, sera le problème principal à résoudre dans la prochaine version de l'application iOS.

IV.3.2.2 Les SMS

Lorsque l'on envoie un SMS manuellement depuis un iPhone, le message s'ajoute dans la conversation ce qui permet d'avoir un historique et un suivi des messages échangés avec la personne. Par contre, lorsque les SMS sont envoyés depuis l'application, ceux-ci ne s'ajoutent pas dans la conversation avec le destinataire.

La solution qui pourrait être utilisée serait d'ajouter "manuellement" le message dans la base de données des SMS de l'iPhone depuis l'application juste avant son envoi. Cette solution n'a pas pu être testée pour l'instant car le fichier de la base de données est (pour l'instant) inaccessible comme expliqué dans la partie III.3.3.2.

De plus, lors de l'envoi des SMS, aucune confirmation d'envoi n'est possible. Cela pourrait poser certains problèmes lorsque le SMS à envoyer est important. La solution pourrait être recherchée dans les prochaines versions de notre solution.

IV.4 Perspectives d'évolutions

Malgré son fonctionnement, nous considérons le projet comme non mature et nous aurions souhaité y apporter de nouvelles fonctionnalités pour le rendre véritablement fonctionnel.

IV.4.1 Fonctionnement dans le cloud

Notre web service fonctionne à condition d'avoir un serveur. Nous l'avons exclusivement fait fonctionner en local sur nos machines personnelles et nous n'avons donc pas pu vérifier son fonctionnement en condition réelle à grande échelle. Pour cela, nous aurions souhaité déployer notre service dans le cloud. Cela signifie simplement que le projet aurait un support matériel qui ne nous appartient pas mais que nous louons.

Heroku[1] permet de louer un emplacement virtuel dans lequel nous aurions pu y placer notre service. Ce système est très intéressant principalement pour une raison : il permet ainsi de déléguer la gestion du ou des serveurs à une autre entreprise. Un peu de la même manière que pour le protocole OAuth, une tache précise est alors gérée par une entité compétente. Plus aucun besoin alors de s'occuper de la maintenance des serveurs ou de gérer les charges des serveurs pour avoir un load balancing idéal.

1. Site web : http://www.heroku.com/

Pour nous c'était aussi l'occasion d'utiliser un service de gestion d'application dans le cloud. Pour des contraintes principalement budgétaires, nous avons choisi de ne pas déployer notre application dans le cloud.

IV.4.2 Déploiement dans les markets

De même, nous pensons que, une fois le projet mature, il serait intéressant de mettre nos applications Android et iOS dans leurs markets respectif. Les markets sont les plateformes de téléchargement officiel de iOS et Android. Déployer notre travail sur ces markets est avant tout un souci de compréhension de ces plateformes. Cela permet aussi de gagner en visibilité même si n'est pas l'objectif premier de ce projet.

Malheureusement, il nous parait impossible aujourd'hui de publier sur les markets. Les raisons sont les moyens mis en œuvre pour développer nos applications. Sur iOS comme sur Android nous utilisons des solutions non présentes sur les frameworks officiels pour réaliser certaines fonctionnalités. Rendre notre travail public sur un market ne serait donc sûrement pas possible ou bien inutilisable pour une grande majorité des utilisateurs.

IV.4.3 Couverture de test

C'est clairement le maillon faible de notre travail. Nous avons privilégié le développement rapide pour atteindre nos objectifs primaires, les fonctionnalités. Plutôt que de se concentrer sur une tâche et de garantir son fonctionnement, nous avons préféré développer le plus possibles de fonctionnalités et nous nous sommes occupés de beaucoup de tâches secondaires. La couverture de test du projet est donc nulle. Couvrir un projet signifie garantir que celui-ci fonctionne à l'aide d'un jeu de tests. Cette partie d'un projet professionnel est souvent conséquente et nous avons fait le choix de ne pas travailler dessus. Il s'agit, à nos yeux, d'une erreur et nous pensons adopter un développement orienté *Tests Driven Development* sur nos prochains projets. Les méthodes TDD reposent sur une approche ou les tests de fonctionnement seront écris avant les fonctionnalités. Cela permet de maintenir un projet fonctionnel.

IV.4.4 Documentation

Nous n'avons produit aucune documentation avec notre projet. Tout est sous forme de commentaires directement écrit dans le code.

Écrire une documentation rendrait le projet plus ouvert. C'est aussi une condition importante quant à la délivrance d'un projet.

De plus cela permettrait de renseigner les utilisateurs curieux sur la nature du projet et son fonctionnement pour lever les doutes concernant la sécurité.

Nous avons néanmoins délibérément abandonné la conception d'une documentation pour des raisons de temps.

IV.4.5 Gestion des MMS

Actuellement, le projet ne gère que les SMS. Nous avons envisagé de rendre possible l'utilisation de MMS. L'un des avantages principal de l'ordinateur par rapport au téléphone portable et la facilité avec laquelle on peut par exemple récupérer une image et à l'aide d'un glisser déposer, l'insérer dans un mail par exemple. Cette fonctionnalité serait très intéressante dans un projet comme celui là.

Conclusion

Ce projet avait pour objectif de trouver et développer une solution permettant aux utilisateurs d'envoyer des SMS depuis un ordinateur. De plus notre cahier des charges imposait une gestion fine des contacts pour obtenir une solution simple et pratique.

Notre première solution remplie entièrement le cahier des charges en proposant une première solution fonctionnelle à l'utilisateur. Du côté des applications mobiles qui font office d'intermédiaire entre le site web et le correspondant, le bilan est mitigé. L'application mobile Android est terminée et maintenable grâce à son architecture, alors que l'application iOS est bloquée pour des raisons de sécurité. Néanmoins, aucune des deux applications n'est aujourd'hui potentiellement publiable sur les markets officiels de Apple et Google. Les techniques employées pour leur fonctionnement les rendent incompatibles avec les politiques mises en place par les responsables des systèmes d'exploitations. Le site web, quant à lui, propose une première version fonctionnelle pour l'utilisateur. L'utilisateur pourra alors se connecter sur le site web avec ses identifiants Google, verra ses contacts s'afficher et pourra discuter avec chacun d'entre eux à condition d'avoir installé et activé l'application mobile iOS/Android sur son téléphone portable. Le produit est donc entièrement opérationnel et rempli les conditions de départs correctement.

De nombreuses améliorations devront être effectuées pour permettre de proposer notre solution au grand public. Néanmoins, avant tout éventuel développement de nouvelles fonctionnalités, il apparaît primordial de travailler sur la couverture de test. Celle-ci n'ayant pas fait l'objet d'un travail de notre part, l'évolution du projet pourrait souffrir de cette lacune et perdre en qualité.

L'ajout de fonctionnalités dans notre projet permettra d'offrir aux utilisateurs une solution plus complète qui satisfera les besoins de tous. Il apparaît alors intéressant d'envisager d'avoir une première version iOS entièrement fonctionnelle.

De même, nous pensons qu'il serait utile de permettre l'envoi et la réception de MMS ainsi que la possibilité de communiquer vers plusieurs destinataires en même temps. Il serait aussi utile de proposer une solution sécurisée de sauvegarde des conversations afin que l'utilisateur puisse, lors de la connexion vers le site, retrouver ses précédents messages.

Webographie

[iMessage] The iPhone wiki « iMessage ». 2012. http://theiphonewiki.com/wiki/IMessage

[CoreTelephony] The iPhone wiki « CoreTelephony ». 2012. http://theiphonewiki.com/wiki/CoreTelephony