Documentation technique

Mobii

Geoffrey Aubert - geoffrey1.aubert@epitech.eu
Nicolas De-Thore - nicolas.de-thore@epitech.eu
Camille Gardet - camille.gardet@epitech.eu
Sebastien Guillerm - sebastien.guillerm@epitech.eu
Nicolas Laille - nicolas.laille@epitech.eu
Pierre Le - pierre.le@epitech.eu
Clement Morissard - clement.morissard@epitech.eu

Aujourd'hui, le marché du smartphone est florissant, le nombre de modèle ne cesse de croître et chaques entreprises utilisent sa propre plateforme pour son produit.

L'utilisateur possédent plusieurs téléphone, ou ayant fait l'acquisition d'un modèle plus récent mais d'une marque différente de son ancien modèle doit donc jongler avec les différentes logiciels proposés par les constructeurs pour gérer les données de ses téléphones.

Mobii propose à ces utilisateurs **un seul outil**. Il pourra ainsi synchroniser des données, effectuer des sauvegardes, transférer sa médiathèque d'un téléphone à un autre plus facilemment.

Mais pas seulement ! L'utilisateur pourra aussi se passer de son téléphone dès lors qu'il est sur son ordinateur. L'application *Mobii* sur ordinateur permettra de recevoir et d'émettre des appels, d'envoyer et de recevoir des SMS via le téléphone.

Des applications similaires à la notre existent déjà, mais elles ne sont pas multi-plateforme ou leurs fonctionnalités sont trop limités, ou bien obsolètes car non mise à jour.

Ce document décrit en détail l'architecture des éléments qui composent le projet *Mobii*. Y sont exposés les réponses aux questions comme **Comment fonctionne ces dits éléments ?**, **Quels sont les interactions entre ces dits éléments ?**

Ainsi, vous découvrirez :

- toutes les caractéristiques des différentes tuiles du projet (techniques, comportementales, fonctionnelles, etc.),
- les différents aspects du projet (logique, processus, données, qualité, etc.),
- une vision de l'implémentation de ces tuiles.

Description du document

Titre	EIP 2014 - Documentation technique
Date	Jeudi 16 Janvier (16/01/2014)
Auteur	Groupe Mobii
Responsable	Nicolas Laille
E-mail	mobii_2014@labeip.epitech.eu
Sujet	Documentation technique du projet Mobii
Version	Finale

Tableau de révision

Date	Auteur	Version	Commentaires
21/03/2013	Group	1.0	Création du document
29/05/2013	Group	3.0	Modification et assemblage du document
15/07/2013	Group	4.0	Modification et assemblage du document
20/10/2013	Group	5.0	Modification et assemblage du document
21/12/2013	Group	6.0	Modification et assemblage du document
16/01/2014	Group	Finale	Modification et assemblage du document

Table des matières

Chapitre 1

Documentation technique du serveur Mobii

1.1 Technologies et environnement de travail

Le serveur est codé en C++ et les librairies utilisées sont :

- Boost 1.52.0 (Asio, Threads, bimap),
- Soci 3.2.0 (pour la couche base de données)

Pour gérer la compilation multiplateforme, cmake est utilisé, de ce fait il n'y a pas d'IDE imposé pour le développement.

1.2 Architecture globale

Le rôle principal de l'application serveur sera de gérer et initialiser les communications entre les différents clients, ainsi que les interactions avec la base de données.

L'applicatif serveur utilise une librairie tierce pour la gestion de la base de données, cette librairie est SOCI, elle supporte plusieurs types de base de données en plus d'être multiplateforme et peut être utilisée avec boost.

Une librairie réseau tierce du nom de boost asio est utilisée pour faire une abstraction réseau de haut niveau, les avantages de cette librairie sont la gestion du multiplateforme mais aussi la flexibilité de cette librairie ainsi qu'une bonne documentation.

Le serveur sera géré exclusivement par le biais d'un client externalisé. Il sera donc développé une application de gestion du serveur, distincte de ce dernier. Ce client ne sera pas développé dans ce document.

Afin de gérer les commandes reçues par la couche réseau le serveur contient une “queue” de messages qui seront exécutés par différentes “threads”.

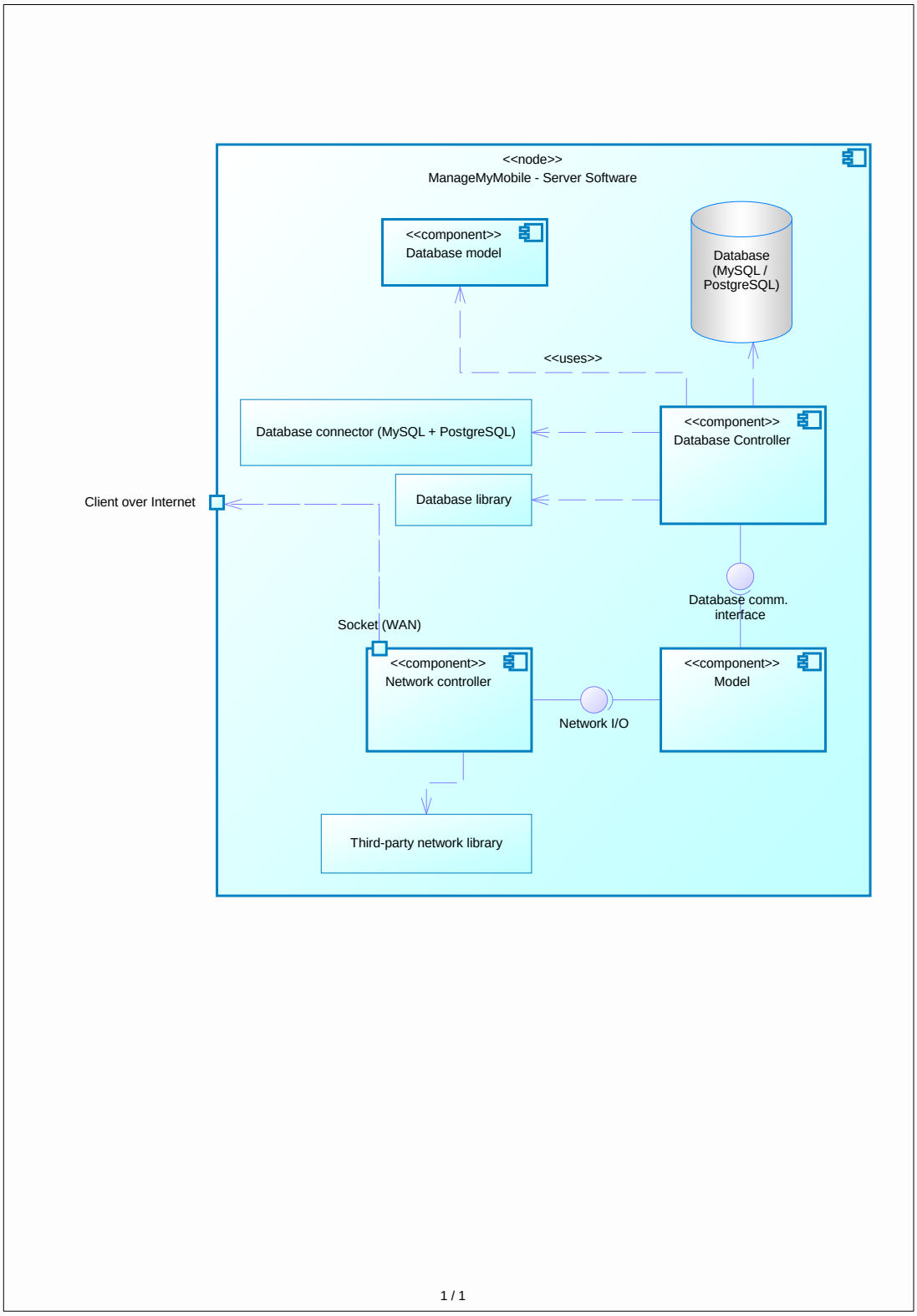


FIGURE 1.1 – Diagramme de composants

1.3 Architecture détaillée

Dans cette partie, l'architecture du serveur sera plus détaillée afin de comprendre les relations entre les classes de l'application ainsi que le rôle de celles-ci.

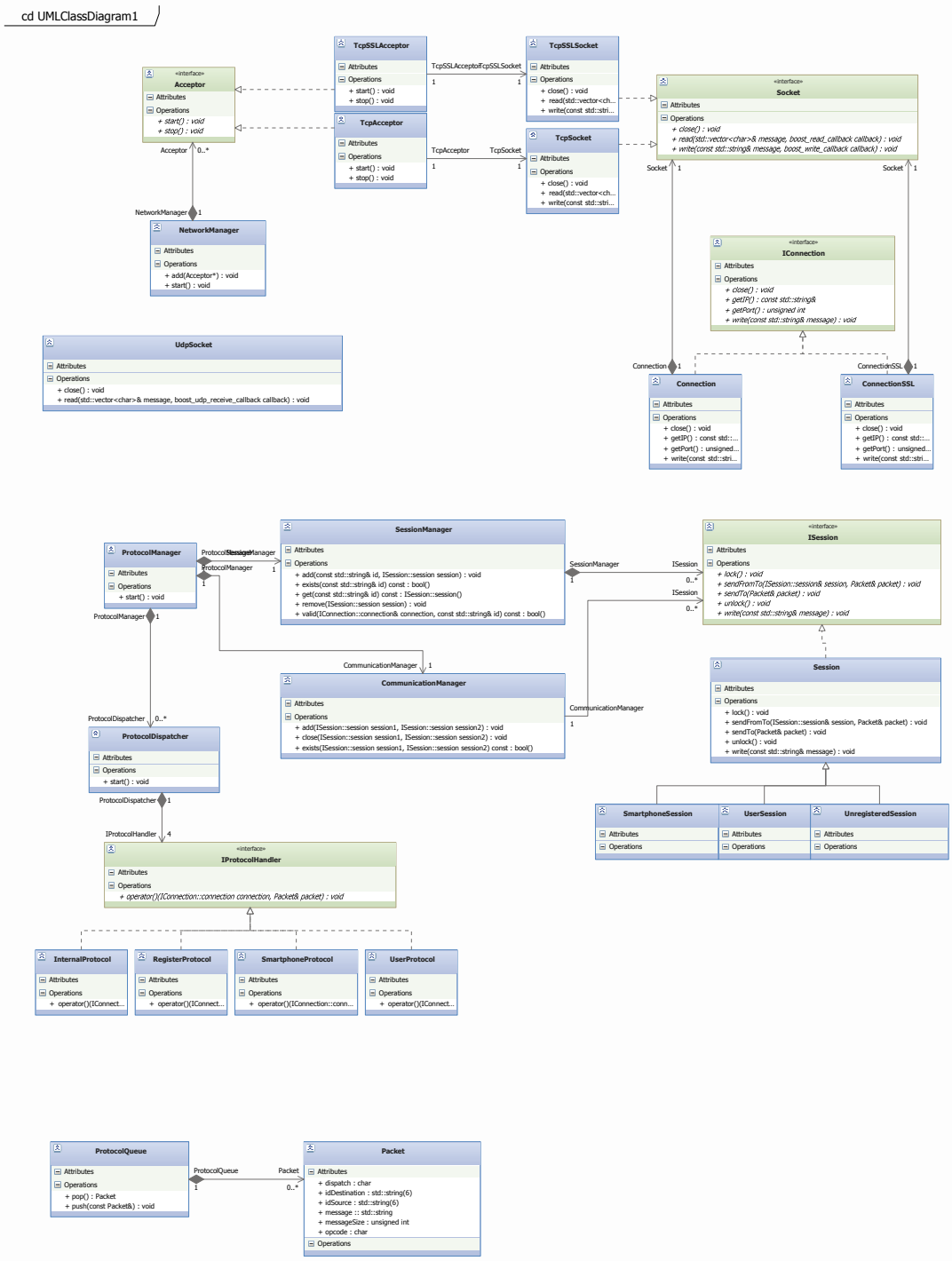


FIGURE 1.2 – Mobii server's class diagram

1.3.1 Architecture de la couche réseau

Une distinction a été faite entre les socket TCP (SSL ou non) et UDP.

Les *Acceptor* permettent d'initialiser les connexions TCP et créent des Connection avec les Socket qui leur sont associées (*TcpSocket* ou *TcpSSLSocket*).

Une *Socket* est l'abstraction des sockets systèmes avec le protocole TCP, elle permet d'écrire et de lire des données en TCP.

Une *Connection* est la représentation d'une connexion sur le serveur, elle est définie par une Socket, une IP et un port. *Connection* gère les messages entrant et les messages sortant. Il place les messages sortant dans une file d'attente et convertis les messages entrant en *Packet* avant des les empiler dans la classe *ProtocolQueue*.

UdpSocket est l'abstraction des sockets systèmes avec le protocole UDP, elle permet d'écrire et de lire des données en UDP.

1.3.2 Architecture de la gestion du protocole

Les messages reçus par la couche réseau sont dépilés dans la classe *ProtocolDispatcher* puis envoyés au *IProtocolHandler* correspondant.

Chaque *IProtocolHandler* (*InternalProtocol*, *RegisterProtocol*, etc.) permet de gérer une partie du protocole.

Dans la partie protocole du serveur, chaque *Connection* est représentée par une Session qui permet de faire la correspondance entre une *Connection* et l'ID de la *Connection*.

Une *Connection* est considérée comme une *UnregisteredSession* tant que la *Connection* n'est pas attribuée à un ID. Tant qu'une *Connection* est une *UnregisteredSession*, celle-ci ne peut pas être utilisé par le reste du protocole.

SessionManager s'occupe de gérer les *Sessions*, il permet de récupérer une *Session* par rapport à son ID ou sa *Connection*, et permet aussi de valider une *Connection* par rapport à un ID. Une *bimap* est utilisée afin de stocker les *Sessions*.

CommunicationManager est la classe permettant de gérer les communications entre *Session*, elle crée des liaisons entre les *Sessions* et permet de vérifier si deux *Sessions* ont le droit de communiquer ensemble.

Chapitre 2

Documentation technique du client Mobii

Ce chapitre vise à présenter les détails de la conception du client desktop de la plateforme Mobii.

Il sera présenté en premier lieu la conception générale de l'application, puis sera ensuite détaillée la conception de chacun des modules principaux qui la constitue, ainsi que leurs interconnexions.

Note — La documentation détaillée de chacune des classes de l'application est disponible sur le site du projet, à l'adresse suivante : http://eip.epitech.eu/2014/managemymobile/dev_doc/desktop/

2.1 Technologies et environnement de travail

Le client desktop Mobii se base sur l'utilisation du framework Qt 5. L'emploi de ce dernier est justifié par le souhait de rendre cette solution multiplateforme et ainsi la proposer au plus grand nombre d'usagers. Il est recommandé d'employer la version la plus récente de ce framework, à savoir la version 5.0.1.

Il est à noter que Mobii utilisait auparavant la version 4 du framework Qt ; il a été décidé au cours du développement de migrer vers la version 5, car cette dernière apportait de nouvelles fonctionnalités jugées suffisamment intéressantes pour justifier une telle mise à niveau.

Le développement de l'application est principalement axé sur Microsoft Visual Studio 2010, mais une solution de compilation QMake est également fournie pour les développeurs souhaitant développer sur une autre plateforme que Windows.

La conception graphique de l'interface utilisateur passe par l'emploi de l'outil Qt Designer, fourni dans le SDK de Qt 4.

2.2 Vue d'ensemble de l'architecture du client Mobii

L'application cliente Mobii est conçue sur la base du patron de conception MVC, ou Modèle-Vue-Contrôleur. Le mode de fonctionnement de Qt, et plus particulièrement sa gestion des interfaces utilisateur, facilite l'utilisation de ce modèle.

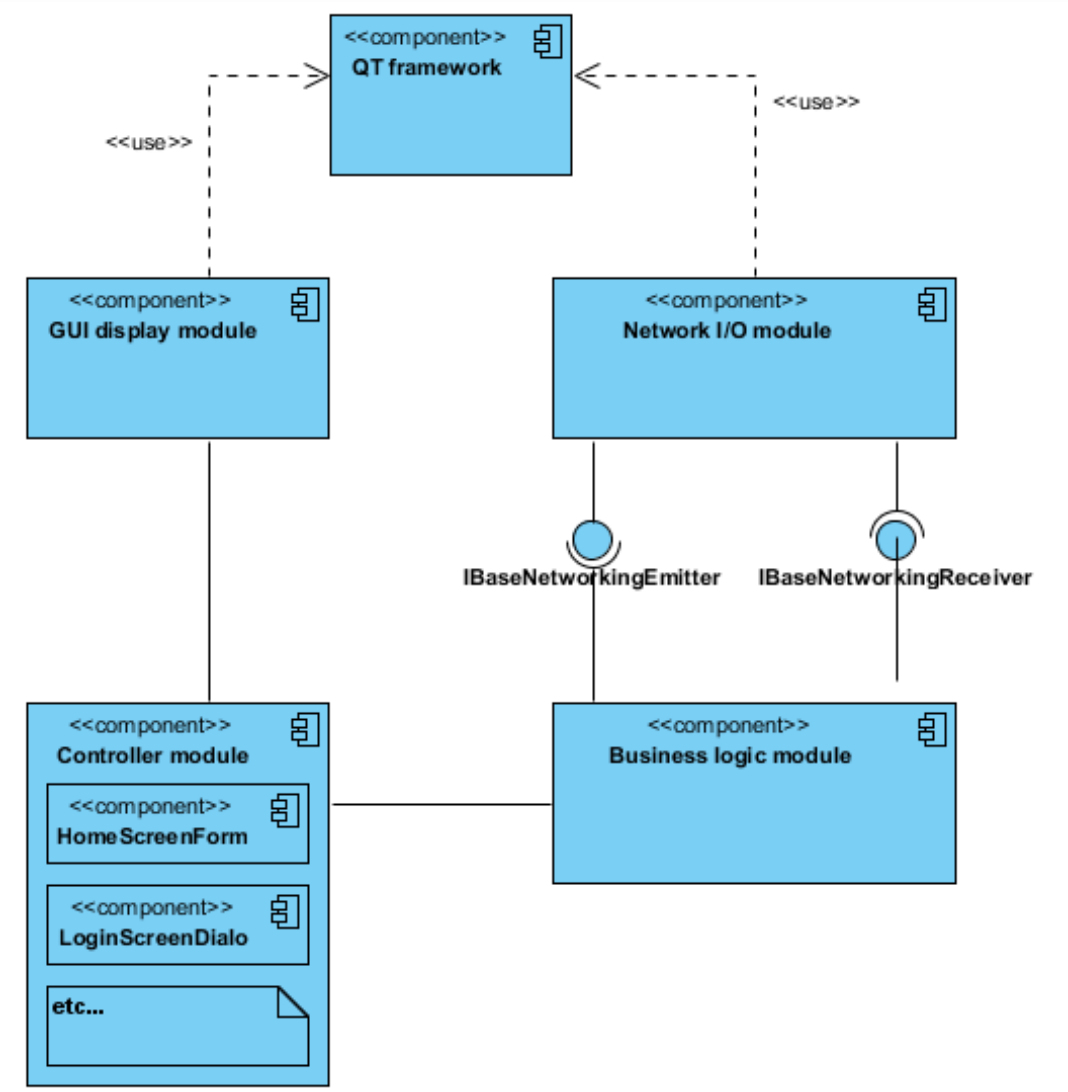


FIGURE 2.1 – Interconnexion des composants de l'application Mobii

2.2.1 Notes sur l'utilisation des bibliothèques réseau Qt

Le module en charge de l'affichage de données à l'écran ainsi que celui gérant les entrées et sorties réseau exploitent les modules Qt adéquats.

A noter que le module réseau, bien qu'étant basé sur les classes réseau Qt, a été conçu de sorte à pouvoir être interchangé pour une technologie différente. Ainsi, le module de logique métier n'interagit pas directement avec le module réseau basé sur Qt, mais plutôt avec les interfaces IBaseNetworkingEmitter et IBaseNetworkingReceiver.

2.2.2 Modules principaux

Le **module d'affichage** (ou *vues*) constitue le pont entre l'utilisateur et les autres modules de l'application. Il permet l'entrée de données utilisateur pour traitement et la représentation graphique de divers évènements internes.

Le **module de logique métier** (ou *modèle*) a pour responsabilité d'assurer l'application du protocole Mobii en traitant les paquets standard Mobii envoyés et reçus du serveur et du client.

Le **module contrôleur** est quant à lui chargé de gérer l'interconnexion entre le module d'affichage et celui de logique métier.

2.3 Architecture détaillée de l'application

Cette section a pour objet de détailler les notions et concepts principaux des divers modules de l'application Mobii.

Le diagramme qui suit résume les interactions entre les différentes classes de l'application.

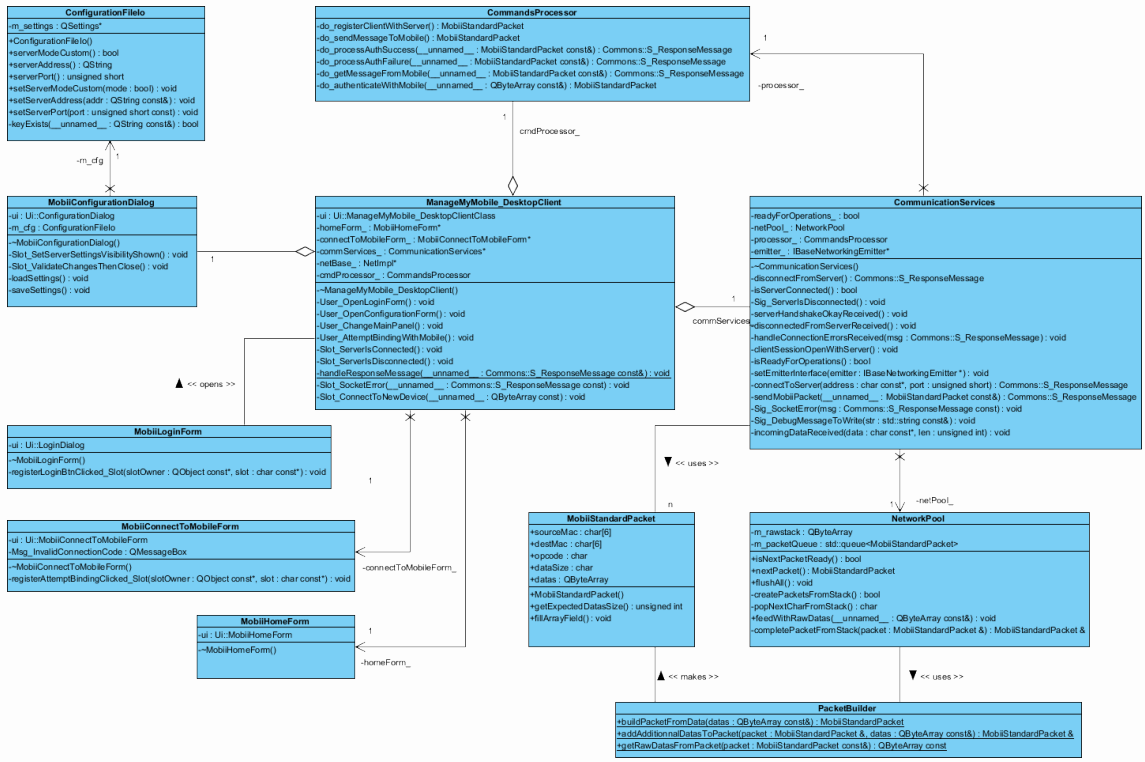


FIGURE 2.2 – Diagramme de classes global de l'application

2.3.1 Système de bus d'évènements global

L'essentiel de l'application est conçu par modules semi-indépendants, rendant l'architecture de l'application client décentralisée.

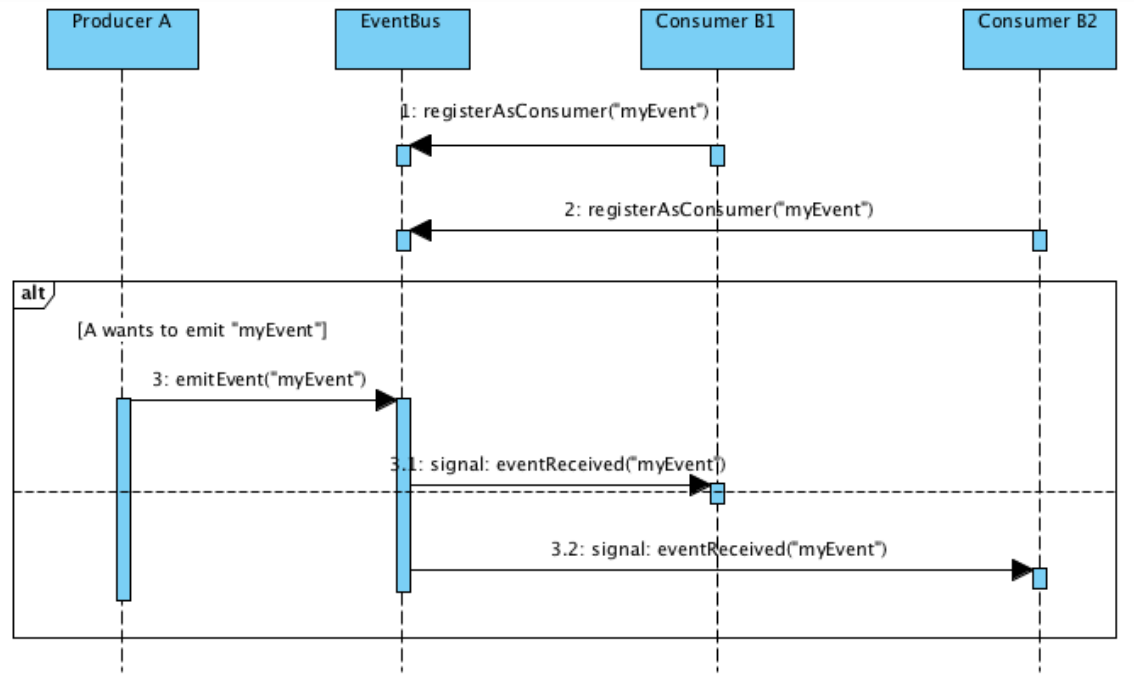


FIGURE 2.3 – Utilisation typique du système de bus d'évènements

Afin de permettre l'intercommunication des différents composants de l'application tout en préservant l'isolation du code, un système de bus d'évènements a été mis en place.

Ce bus d'évènements, opéré par la classe **EventBus**, utilise le patron de conception *Producteur/Consommateur*. Ainsi, un module A peut diffuser un message donné avec des informations arbitraires, tandis qu'un autre module B peut s'inscrire à l'évènement associé à A et ainsi être informé de la transmission d'un nouveau message.

La transmission de ces évènements est basée sur le système de signaux et slots de Qt. La production d'un évènement correspond au déclenchement d'un signal Qt, tandis que la consommation d'un évènement correspond à l'activation d'un slot Qt.

2.3.2 Gestion des panneaux dans l'interface utilisateur

La philosophie de l'interface graphique Mobii est de proposer l'essentiel des fonctionnalités offertes par l'application dans une fenêtre intégrée. Cette intégration est assurée par un système de panneaux (ou *panels*) qui se chargent de façon sélective au sein de la fenêtre principale, de classe **ManageMyMobile_DesktopClient**.

Chacun de ces panneaux se compose à la fois d'une vue au format .UI (fichier Qt Designer) et d'un contrôleur dédié, hérité de **QWidget**. Ce contrôleur local a pour tâche de traiter les entrées utilisateur et de gérer les échanges de données avec le module de logique métier le cas échéant.

De plus, tous les panneaux héritent de l'interface **IEventBusClient**, ce qui leur permet de récupérer ou d'émettre des évènements arbitraires. Les émetteurs et les récepteurs peuvent être des panneaux, mais aussi tout autre module héritant de l'interface **IEventBusClient**.

La classe contrôleur principale **ManageMyMobile_DesktopClient** regroupe l'ensemble

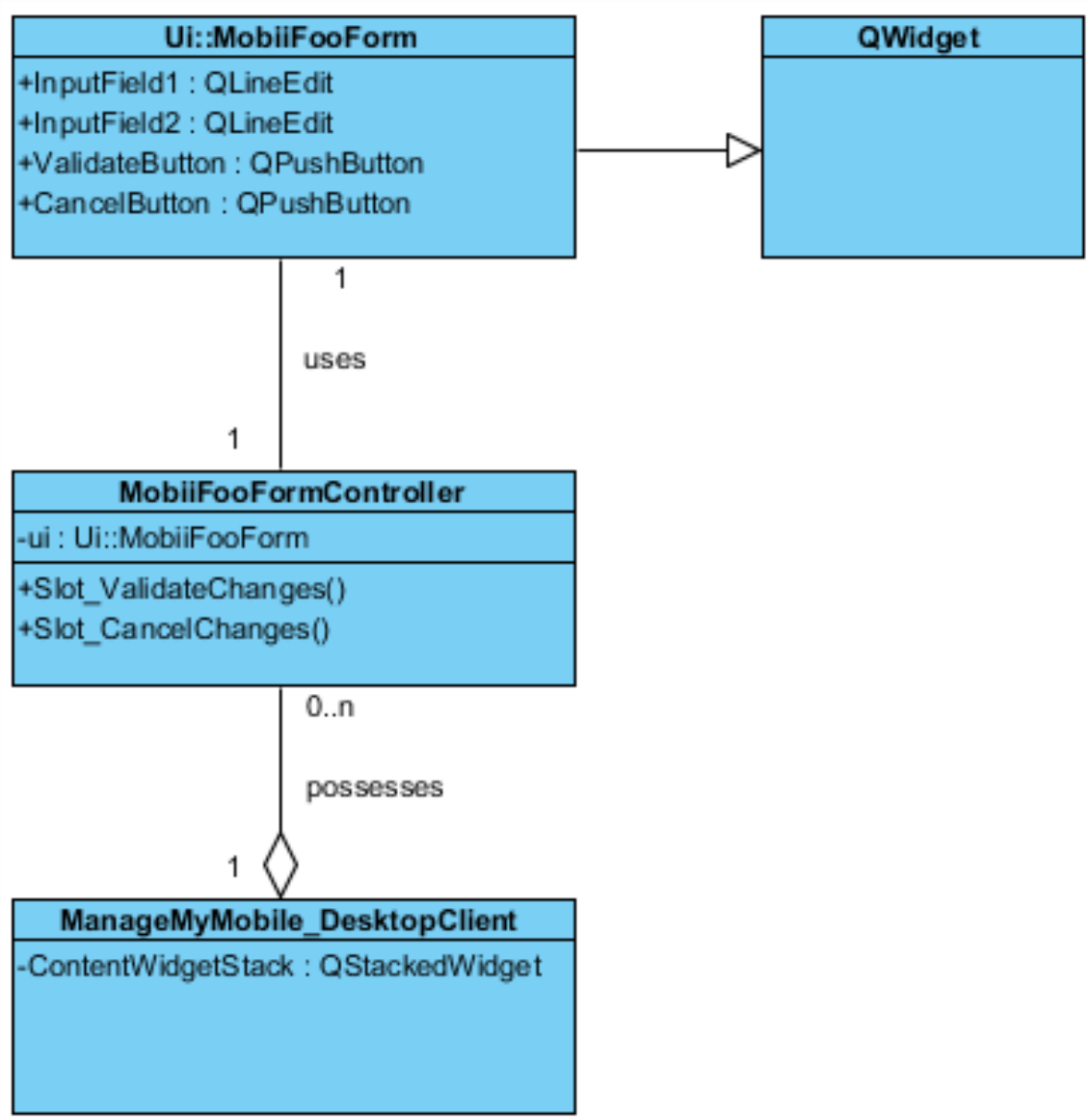


FIGURE 2.4 – Constitution typique d'un panneau

de ces panneaux au sein d'un objet **ContentStackWidget** ; ce dernier permet de charger un panneau dynamiquement par un simple appel de méthode.

L'exemple fourni ci-dessus décrit les classes constituant un panneau typique, ainsi que leurs interconnexions. Ainsi, la classe UI ne communique qu'avec son contrôleur local, qui a un accès exclusif et total aux données membres de l'UI. Il gère les évènements liés aux signaux envoyés par la vue et les traite en fonction.

2.3.3 Architecture réseau et logique métier

2.3.3.1 Entrées/sorties réseau de bas niveau

Les entrées et sorties réseau de bas niveau sont assurées par un module externe au projet principal (*Core Application* sur la figure ci-dessus), **NetImpl**.

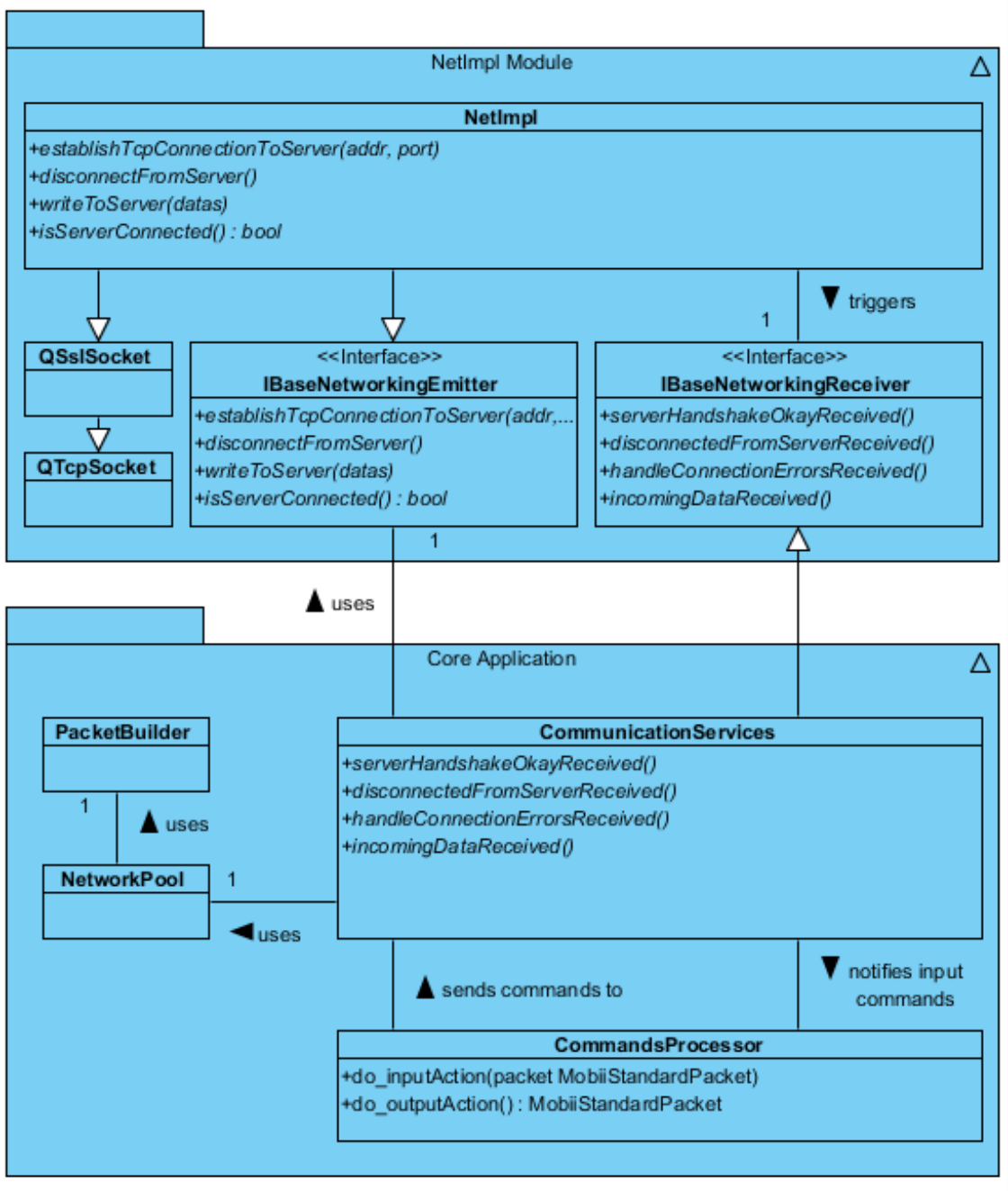


FIGURE 2.5 – Architecture de communication réseau

Les deux modules communiquent par le biais des deux interfaces **IBaseNetworkingEmitter** et **IBaseNetworkingReceiver**.

Les paquets en provenance du réseau sont ensuite reconstruits à l'aide de la classe d'agrégation **NetworkPool**. Tout paquet standard Mobii est construit à l'aide de **PacketBuilder**.

2.3.3.2 Protocole bas niveau

La gestion du protocole "bas niveau" en soit est la responsabilité de la classe **CommunicationServices**, qui est le module central qui gère les évènements liés au protocole. Ce protocole est une

surcouche au TCP/IP.

Il constitue un élément essentiel du module de logique métier et sert de pont entre tout ce qui a trait aux opérations réseau et la prise en charge du protocole.

Ce protocole est l'un des deux utilisés par dans l'écosystème Mobii. Il est dit "bas niveau" car il est est celui qui est le plus proche de la couche réseau du système.

Il s'agit essentiellement d'un protocole de transport, qui est chargé d'implémenter des mécanismes permettant de véhiculer des données utiles d'un élément de la plateforme à un autre.

Ces mécanismes sont les suivants :

- **Expéditeur d'un paquet (code unique)**
- **Destinataire d'un paquet (code unique)**
- **Identification de l'ordre à exécuter du côté du destinataire**
- **Taille attendue de la charge utile**
- **Charge utile du paquet**

2.3.3.3 Protocole d'actions

Le protocole d'actions constitue le second protocole de la plateforme Mobii. Il est conçu pour se greffer dans un paquet Mobii dit "bas niveau" et en constitue la charge utile.

Ce protocole est spécialisé dans le transfert d'informations métier, notamment entre un client et une application mobile Mobii.

Le standard JSON, ou JavaScript Object Notation, a été sélectionné pour constituer ce protocole d'actions. Sa nature flexible permettent notamment de pouvoir changer le format du protocole d'actions en fonction des besoins métiers, indépendamment de l'implémentation du protocole de "bas niveau".

Une autre raison du choix du JSON est qu'il s'agit d'un standard répandu, pour lequel il existe de nombreuses implémentations fiables et ce sur la plupart des plateformes existantes.

Bien que le protocole d'actions soit par nature flexible, toute structure JSON issue du protocole d'actions doit obligatoirement contenir un champ *action*, qui définit la signification métier des données contenues dans le paquet.

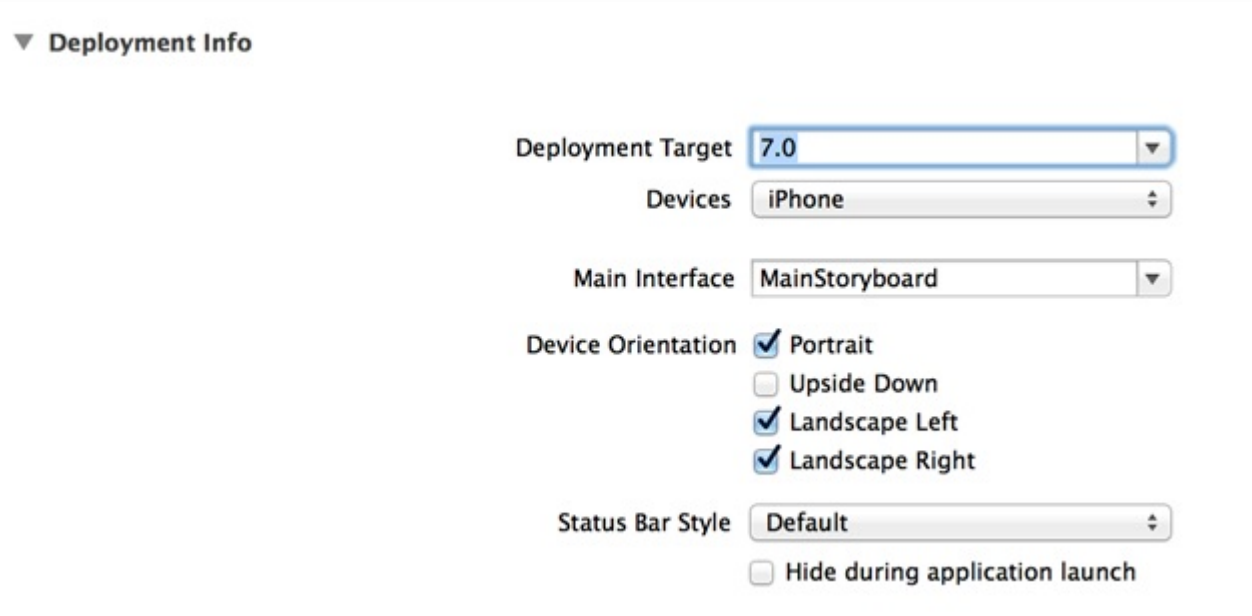
Chapitre 3

Mobii iOS App

3.1 Compilation du projet

3.1.1 XCode

Le projet Xcode Mobii iOS doit être compile avec la dernière version d'XCode (v5.02) et comme SDK de base iOS 7.0.



3.1.2 Dépendances

Le project inclus des librairies externes :

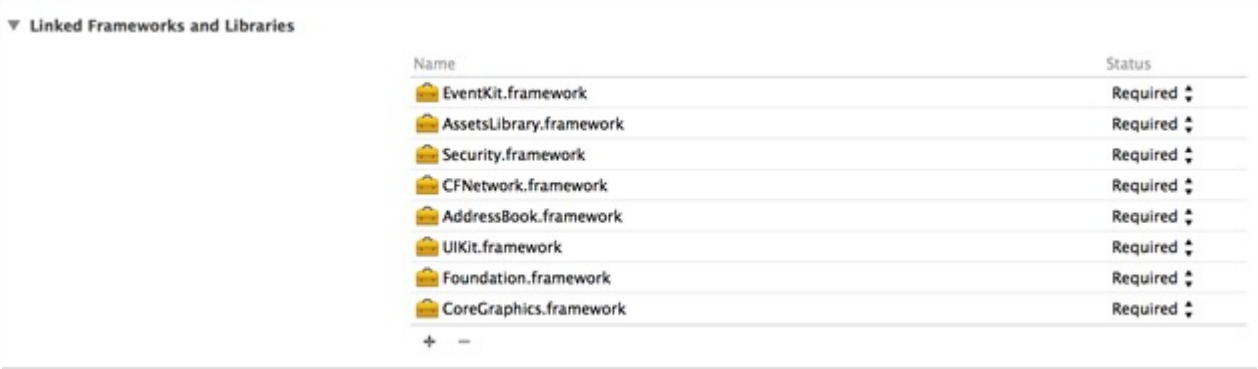
- **Cocoa Async Socket for Mac and iOS**. Cette librarie fournis des outils pour manipuler les socket iOS de facon asynchrone. Deux objets provenant de cette librairie ont été ajoutés au projet : GCDAsyncSocket et GCDAsyncUdpSocket.Plus d'informations sur : http://github.com/robbiehanson/CocoaAsyncSocket

- **Touch JSON**. Cette librarie fournie des outils permettant de manipuler les données JSON. Deux objet de cette librairie ont été inclus dans le projet : CJSONSerializer et CJSONDeserializer.
  Please find more informations on : https://github.com/TouchCode/TouchJSON

Le projet repose également sur des framework Apple non inclus par default :

- EventKit
- AssetsLibrary
- Security
- CFNetwork
- AddressBook

Tous ces framework sont inclus dans le SDK Xcode mais doivent tout de même être linké à lors de la compilation. Pour ce faire, ajoutez ces derniers dans la propriété Linked Frameworks and Libraries du projet.



3.2 Structure de l'application

L'application Mobii iOS à été créée selon un design patter MVC. La majorité des méchanismes de l'application sont contenus dans 5 principaux objet (Model) :

3.2.1 Réseau

L'aspect réseau du projet est pris en charge par l'objet ConnectionManager. Ce dernier est créé avec des paramètres de connexion (IP, port, etc..) et encapsule le type de socket utilisé. Un ensemble de méthodes est fournis pour interagir avec les connections de l'application le plus abstraitement possible.

3.2.2 Gestion de contacts

L'objet ContactManager encapsule le framework AddressBook et fournis un ensemble de méthodes pour récupérer et manipuler la liste de contacts. Le format attendu pour un contact est spécifié dans le protocole d'action.

3.2.3 Gestion du calendrier

L'objet CalendarManager encapsule le framework EKEvent et fournis un ensemble de méthodes pour récupérer et manipuler le calendrier. Le format attendu pour un évenement est spécifié dans le protocole d'action.

3.2.4 Gestion de fichiers

L'objet FileManager encapsule le framework ALAsset et fournis un ensemble de méthodes pour récupérer et manipuler les fichiers (pour l'instant, seulement les photos).

3.2.5 Gestion de packet

L'objet PacketManager prends en charge les paquets recu, les traites et génère des paquet de réponse. Ce dernier contient chacun des objets cités précédemment dans le but de fournir les réponse appropriée aux differentes requêtes. Le format d'un paquet est défini dans le protocole de communication.

3.3 Divers

3.3.1 Bugs connus

3.3.2 To-Do

- Internationalisation,
- Local Notification quand un contact/event est ajouté, modifé ou supprimé,
- Fichier .plist pour faire une correspondance entre le label envoyé par le client et les labels par défaut d'iOS. (ex : Le client envoie une liste de numéros de téléphone, chaque numéro doit être stocké avec le label approprié sur iOS).

Chapitre 4

Mobii Android App

4.1 Technologies et environnement de developpement

L'Application Android est developpee en Java via l'IDE Eclipse (Bundle Android). Les tests de l'application sont effectués sur un émulateur et sur un smartphone Samsung Galaxy S3.

4.2 Architecture globale

L'Application est principalement composée de deux activités et d'un service.

4.3 Architecture détaillée

4.3.1 Activités

L'Activité principale permet la connexion de l'application au serveur par un simple appui sur le bouton de connexion. Elle est également composée d'un menu permettant l'accès au panneau de configuration ("Settings") et au lien vers le site Mobii ("About") et l'arrêt de l'application ("Exit").

Une fois la connexion établie, une seconde activité est lancée. Cette dernière est composée d'un bouton permettant de déconnecter l'application du serveur.

4.3.2 Service de connexion

Un service (ConnectionService) est lancé lors de la connexion au serveur afin de créer une socket et de maintenir la connexion entre l'application et le serveur, même en fond de tâche. La socket permet l'envoi et la réception de flux de données.

4.3.3 Gestion des flux entrants et sortants

Un module composé d'une classe principale (TcpPacket) s'occupe de la création des paquets à envoyer, ainsi que de la réception et du traitement des paquets entrants.

4.4 Description des fonctionnalités

L'Application Android permet :

- La récupération des informations du smartphone, tels la marque, le nom, la version du système d'exploitation,
- La gestion des contacts (récupération, ajout, suppression et mise à jour),
- L'envoi et la réception de sms à partir du client.

Les fonctionnalités suivantes sont en cours de développement :

- La gestion du calendrier (récupération, ajout, suppression et mise à jour),
- L'envoi et la réception de mms,
- La récupération du contenu multimédia, tels les musiques, vidéos, images et autres fichiers.

Des fonctionnalités pourraient également s'avérer intéressantes à ajouter :

- L'envoi et la réception d'appels à partir du client, à condition que la connexion le permette (3g/4g comme Wi-Fi).

4.5 Bibliothèques

Aucune bibliothèque externe n'est utilisée.

Néanmoins, certaines ont fait l'objet de tests :

- Google gson, pour le format des messages composants les paquets,
- Joda-time, pour la gestion de dates/formats des dates.

Une bibliothèque pourrait également faire l'objet de tests afin d'essayer d'améliorer la partie connexion :

- Extasys.

Chapitre 5

Mobii Windows Phone App

5.1 Environnement de développement et technologies utilisées

L'application mobile à été développée avec Visual Studio 2013 sous Windows 8.1 utilisant le kit de développement mobile 8.1. La rétrocompatibilité du projet avec des versions antérieures de ces outils n'est pas garanties.

Les langages utiilisés sont le C++, pour la partie réseau, le C# et le XAML pour le reste de l'application.

5.2 Dépendances externes

Seul la bibliothèques JSON.NET est utilisée. Elle est incluse dans le projet.

À propos de JSON.NET : https://json.codeplex.com/

5.3 Architecture

L'application Mobii Windows Phone utilise le pattern MVVM, ainsi qu'une librairie interne personnalisée pour la partie réseau.

5.3.1 Réseau

Ce module est écrit en C++ est utilise la bibliothèque WinSock afin de garder une certaine liberté d'action sur la connectivité.

5.3.2 Contacts et Calendrier

Le SDK ne permet pas de modifier, créer ou supprimer des contacts/calendriers du téléphone, seulement d'en lire les informations. Cependant, le SDK permet la création d'un "ContactStore", une copie locale des contacts/calendriers où les applications ont tous les droits.