Documentation technique

Defuze.me – EIP 2012

Informations

Nom du projet	Defuze.me
Type de document	Documentation technique
Date	19/01/2012
Version	1.8
Mots-clés	Architecture – Fonctionnement – Technologies – Diagrammes – Bugs - Interactions
Auteurs	Adrien Jarthon – jartho_d Arnaud Sellier – sellie_a Alexandre Moore – moore_a Jocelyn De La Rosa – de-la-_o Athéna Calmettes – calmet_b Luc Pérès – peres_a François Gaillard - gailla_f

Rédaction et modifications

Version	Date	Nom	Description
1.0	09/03/2011	Adrien Jarthon Arnaud Sellier Alexandre Moore Jocelyn De La Rosa Athéna Calmettes Luc Pérès François Gaillard	◆ Première version
1.1	13/04/2011	Adrien Jarthon Jocelyn De La Rosa	◆ Détails sur la base serveur ◆ Mise à jour API mobile ◆ Détails sur la partie Audio ◆ Qt Mobility - Multimedia
1.2	27/04/2011	Adrien Jarthon	◆ Détails API web service
1.3	07/05/2011	Adrien Jarthon	◆ Mise à jour des diagrammes
1.4	08/05/2011	Adrien Jarthon	◆ Uniformisation des formats

1.5	17/05/2011	Athéna Calmettes	◆ Ajout table des illustrations
1.6	08/06/2011	Jocelyn De La Rosa	◆ Résumé du document
1.7	10/06/2011	Athéna Calmettes	◆ Ajout des mots-clés
1.8	19/01/2012	Adrien Jarthon	◆ Formatage

Table des matières

1 - Résumé du document

Ce document est la documentation technique officielle de la suite applicative defuze.me. Il est divisé en quatre parties :

- La documentation technique du client : l'application bureau ;
- La documentation technique du serveur: site internet et APIs ;
- La documentation technique de l'application mobile fonctionnant sur Android et iOS ;
- Les bugs connus au sein de la suite logicielle.

2 - Rappel sur le fonctionnement de l'application

2.1 - Description du logiciel

Notre EIP a pour but la création d'un logiciel de diffusion de radio destiné aux professionnels : defuze.me.

Plus que la diffusion de musique, il permet aux stations de radio de gérer un maximum d'éléments de leur quotidien, notamment les contrats publicitaires, les jingles, la récupération et l'exportation de flux de données.

Le logiciel a une interface moderne et ergonomique, permettant de gérer efficacement et simplement la diffusion tout en proposant de nombreuses manipulations avancées. Il est utilisable aussi bien sur des surfaces classiques que tactiles, et ce quel que soit le nombre d'écrans à disposition.

Enfin, un fort accent est mis sur l'interaction avec un service web conçu par nos soins, permettant aux radios d'interagir facilement avec leurs auditeurs, au travers d'Internet.

2.2 - Décomposition du projet

Notre projet se décompose en différentes parties :

- Le logiciel client, qui est une application bureau fonctionnant sur Windows, Linux et MacOS, permettant la gestion du son, des pistes audio, de la bibliothèque musicale et du planning ;
- Une application fonctionnant sur les tablettes tactiles équipées d'iOS ou d'Android, dialoguant avec le logiciel client et implémentant les fonctions de base, telles que la gestion de la bibliothèque et l'organisation du planning ;
- Un service web et un site web, qui sont hébergés sur un serveur applicatif distant et qui communiquent en permanence avec le logiciel client, pour assurer de multiples fonctions comme le contrôle de la licence du logiciel et la gestion du planning à distance.

Dans la suite de ce document, chacune de ces différentes parties sera développée.

2.3 - Architecture globale

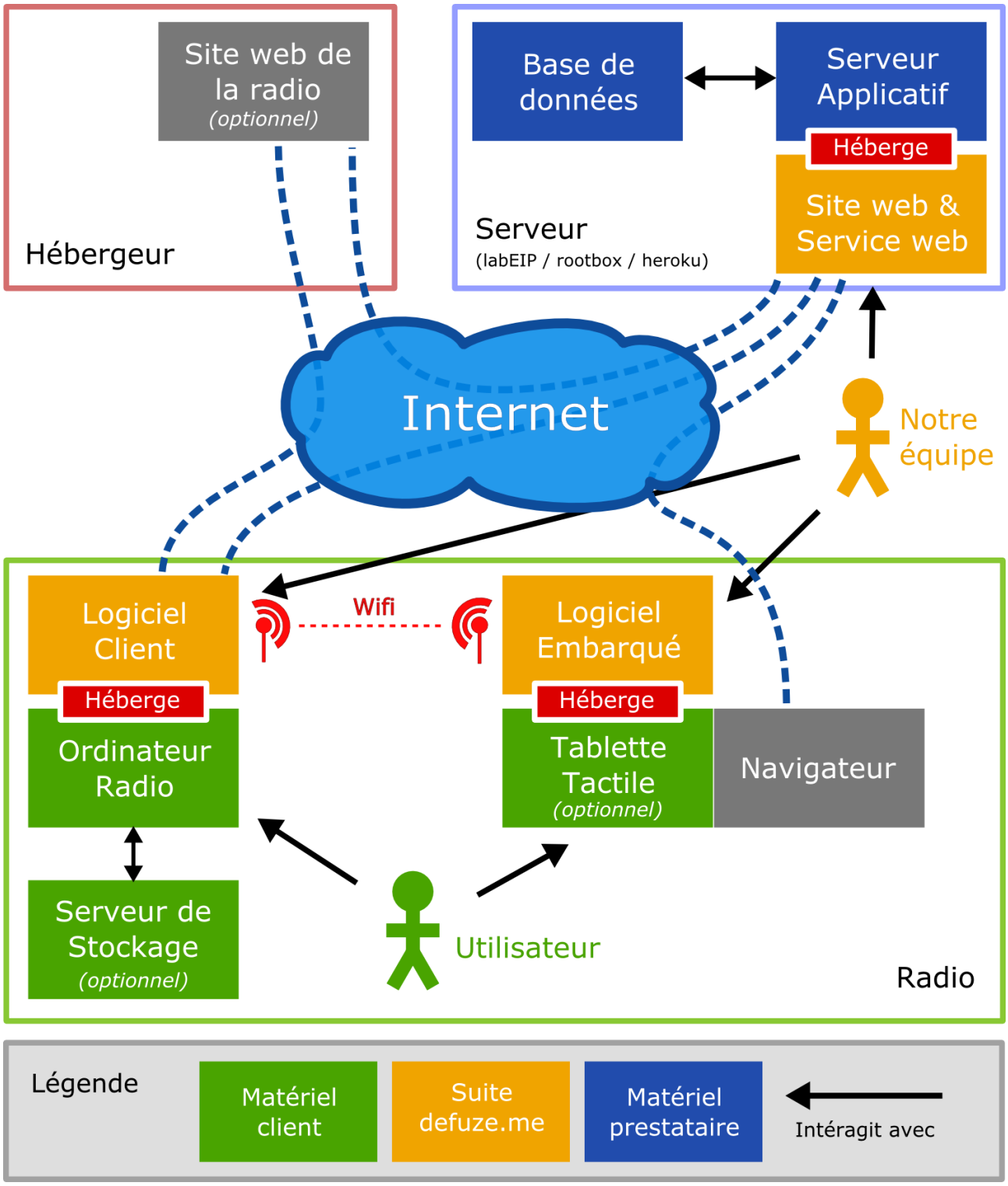

Illustration 1 : Architecture globale

3 - Client

3.1 - Architecture

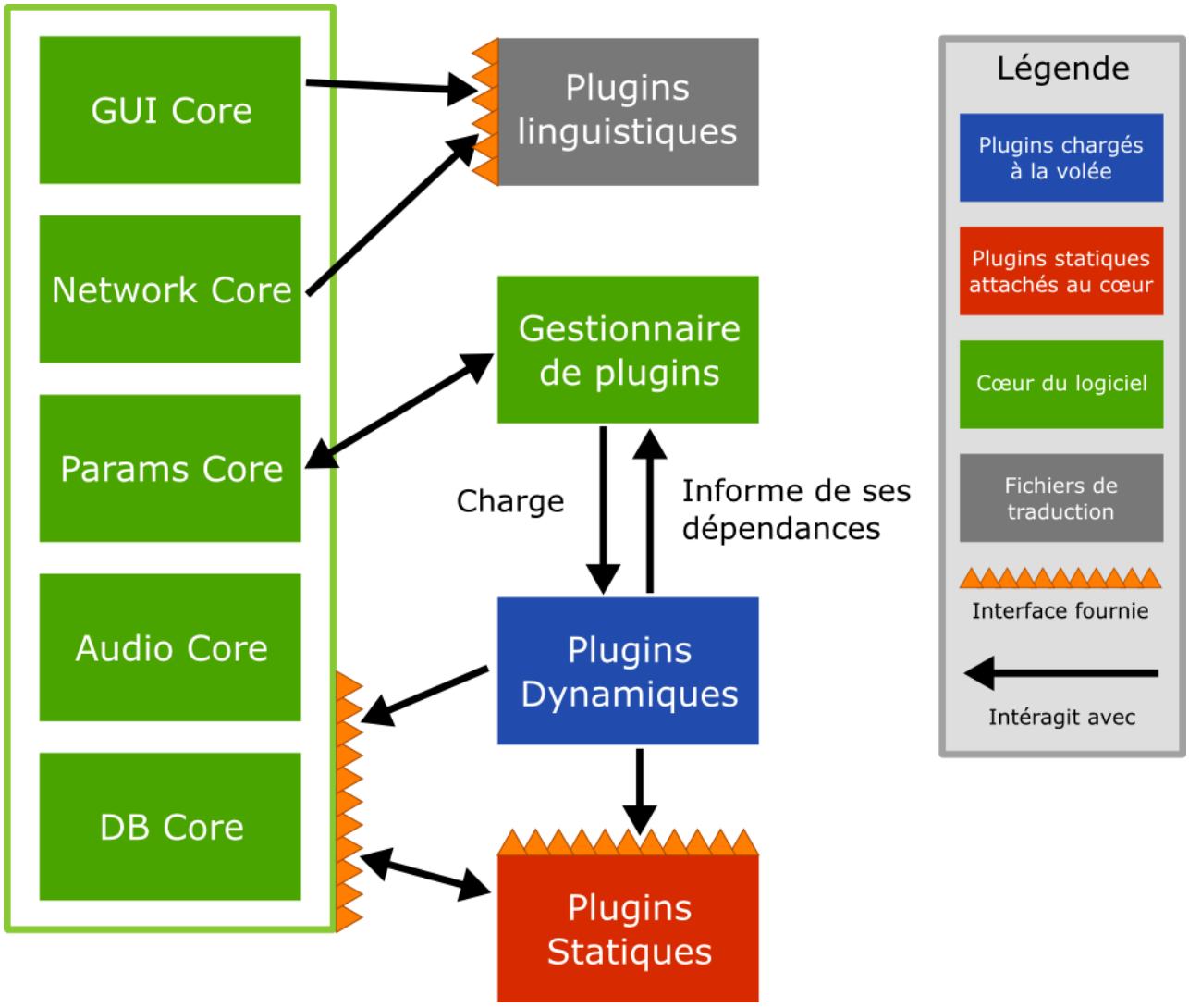

Illustration 2 : Client - Architecture

Le cœur du logiciel, divisé en plusieurs modules, ne propose pas directement de fonctionnalités à l'utilisateur, mais fournit des services aux plugins.

Certains de ces plugins sont statiques. Ils fournissent à l'utilisateur les fonctionnalités essentielles du logiciel (la file de lecture ou les lecteurs audio par exemple). Ils sont indissociables du cœur du logiciel.

Au contraire, les plugins dynamiques fournissent des fonctionnalités supplémentaires qui n'intéresseront peut-être pas tous les utilisateurs. Ils peuvent être (dé)chargés à la volée suivant les besoins, à l'aide du gestionnaire de plugins.

Enfin, les plugins linguistiques sont des fichiers binaires contenant les traductions des éléments textuels.

3.2 - Technologies utilisées

Le logiciel client est codé en C++ et utilise le framework Qt de Nokia.

Qt fournit tous les éléments nécessaires à la réalisation de notre application, et apporte plusieurs éléments essentiels :

- La portabilité : un code Qt compile indifféremment sous les trois systèmes d'exploitation ciblés, à savoir Windows, Linux et Mac OS X.
- Soutenu par Nokia, Qt est utilisé dans des projets professionnels de grande envergure (tels KDE ou Meego). Il est en développement perpétuel et possède une communauté active. Cela nous assure la viabilité à long terme de ce framework.
- Nokia et Qt fournissent depuis peu de nouveaux modules Qt ciblés sur une utilisation mobile et tactile. La version allégée du logiciel client utilise donc des technologies très récentes.

Parmi les nombreux éléments du framework Qt, certains sont particulièrement importants dans l'architecture mise en place :

- Qt Plugins

  Nous utilisons l'API bas niveau de Qt permettant d'ajouter des fonctionnalités aux applications. Ces plugins sont soit statiques, soit dynamiques (sous forme de bibliothèques dynamiques SO ou DLL). Les plugins interagissent avec l'application via une interface C++.

  Les Qt Plugins permettent ainsi à l'architecture d'être modulaire.

- Qt Linguist

  Qt Linguist est un outil incorporé au framework Qt qui permet de créer facilement une application multilingue. Il permet en effet de séparer le texte affiché du code (via l'utilisation de clés). La traduction peut alors être effectuée séparément grâce à l'outil graphique fourni, puis rendu disponible via des fichiers binaires .qm. Ces binaires peuvent être chargés à la volée par l'application Qt.

  De plus, Qt peut détecter la langue du système de l'utilisateur et charger automatiquement le binaire .qm le plus adapté.

- Qt Mobility - Multimedia

  Qt Mobility – Multimedia est un add-on à Qt fournissant une interface de programmation permettant la lecture et l'enregistrement audio, la gestion de contenu multimédia (listes de lectures) et donnant un accès bas niveau aux flux audio, permettant ainsi leur modification pour appliquer des effets et des filtres.

Les technologies hors-Qt utilisées sont relatives à la base de données.

- SQLite (via QtSQL)

  SQLite a été choisi comme SGDB du logiciel client pour sa simplicité. En effet, cette application ne sollicite qu'assez peu la base, et ne requiert pas d'opération complexe.

  L'utilisation d'un SGDB plus complexe (utilisant par exemple une architecture client/serveur) n'est pas nécessaire.

  L'application accède à la base de données via le module QtSQL du framework Qt, qui fournit une interface objet simple.

3.3 – Diagramme de classes

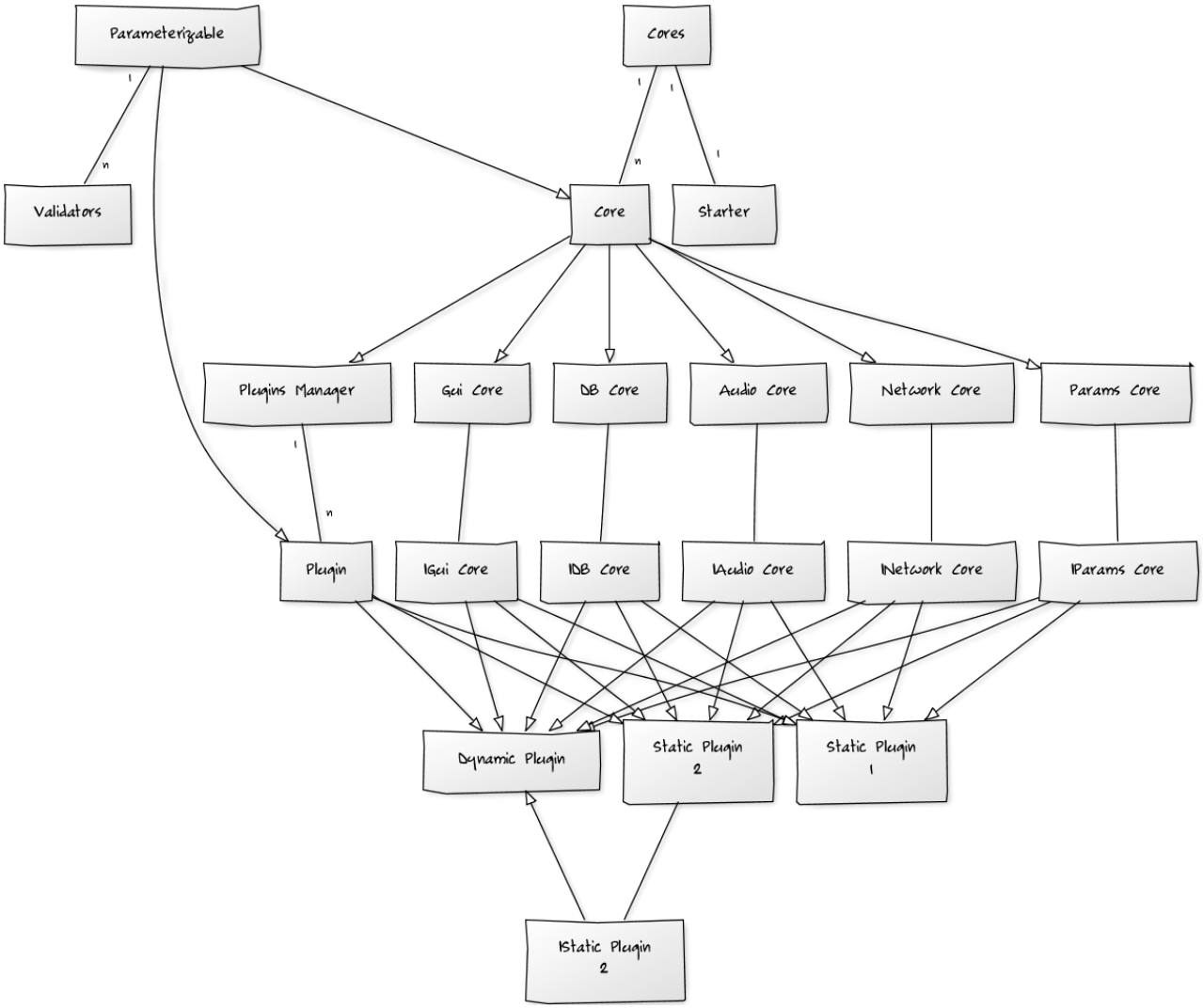

Illustration 3 : Client - Diagramme de classes

- Les `Core` fournissent les fonctionnalités de base du logiciel via une interface (IGuiCore pour le GuiCore par exemple). Chaque `Core` ou plugin est libre d'implémenter une ou plusieurs de ces interfaces pour exploiter les fonctionnalités du ou des `Core`.
- De la même manière, les plugins statiques fournissent également une interface pouvant être implémentée par les autres plugins.

3.4 - Modèle de données

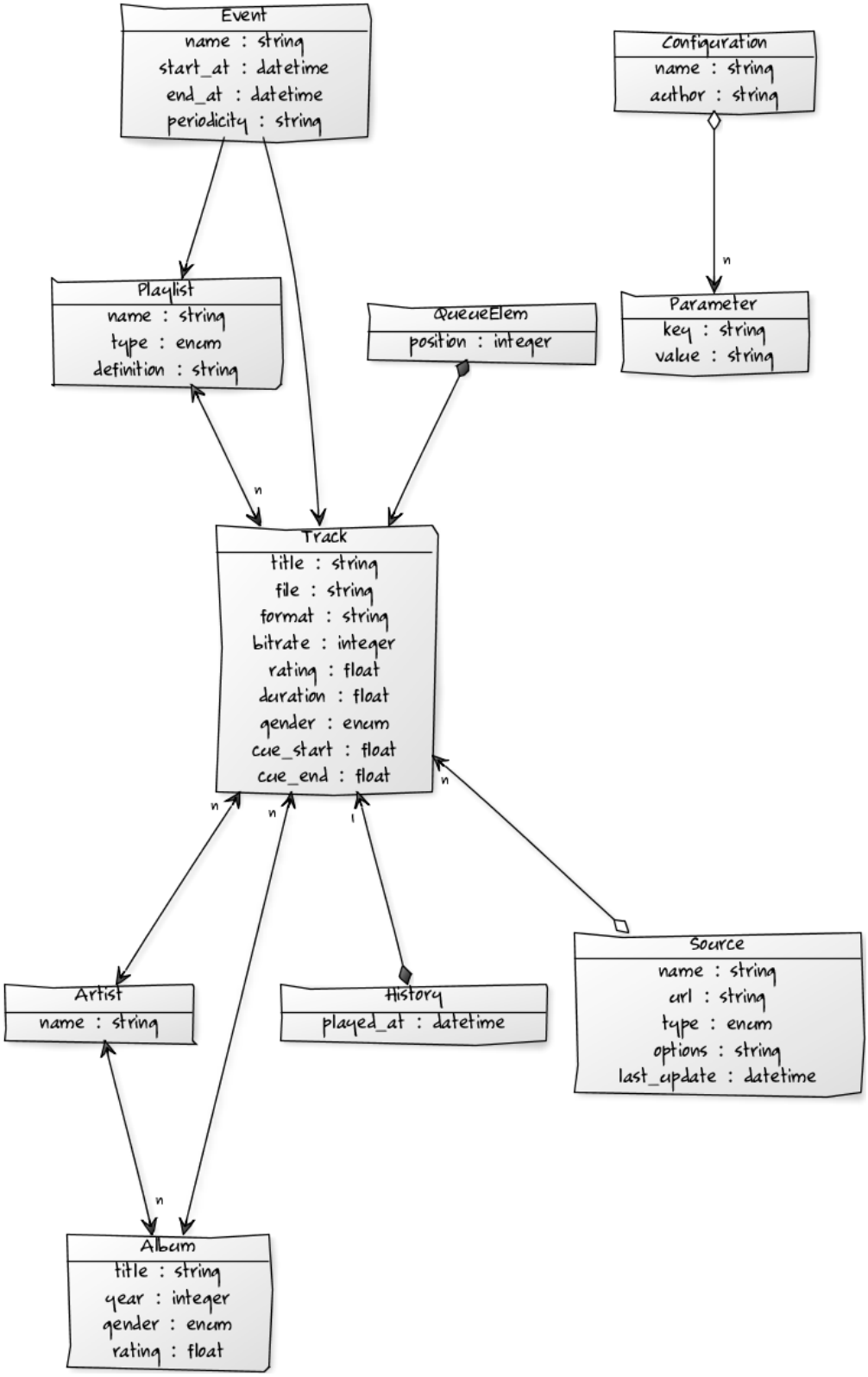

Illustration 4 : Base de données - Diagramme de classes

3.5 - Particularités de l'application allégée

L'application allégée, directement connectée au logiciel client, propose à l'utilisateur des contrôles de base. Elle se comporte de la même manière que les plugins statiques. Cependant, elle n'interagit pas directement avec les CORES, mais seulement avec le TABLET API PLUGIN, qui fait office de passerelle entre les CORES et l'application allégée (voir le diagramme de l'application mobile).

L'application allégée utilise une technologie particulière du framework Qt : QML.

Le Qt Modeling Language est une technologie Qt récente qui permet de créer facilement (via une syntaxe Javascript) une interface graphique adaptée aux écrans tactiles. Cela permet de séparer facilement l'interface graphique du reste du code, idéal pour l'application allégée qui utilise en grande partie le code des plugins statiques, mais pas leur interface.

4 - Serveur

4.1 - Architecture

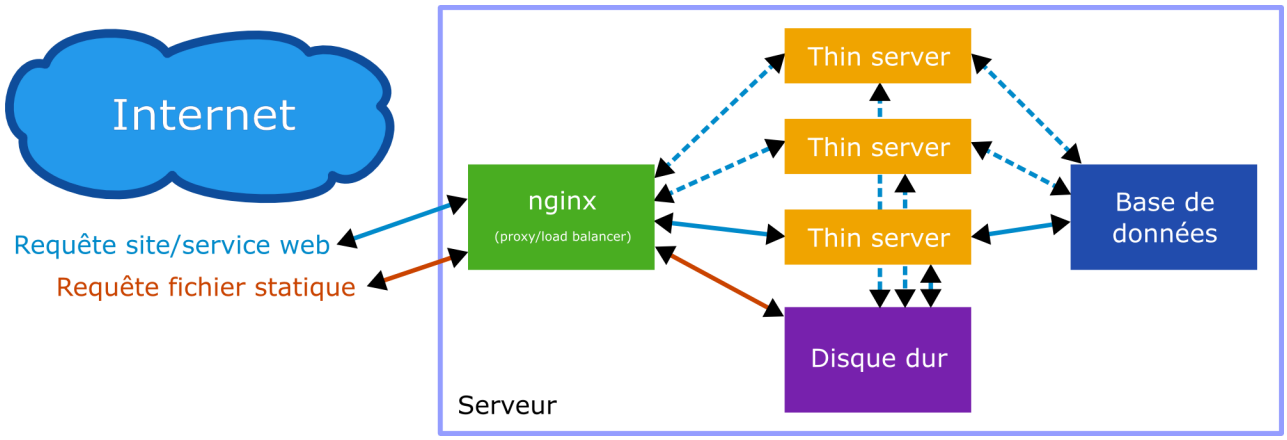

Illustration 5 : Serveur - Architecture

Le serveur fournit les fonctionnalités suivantes :

- Le site public de l'application ;
- L'interface d'administration en ligne ;
- Le service web qui communique avec le logiciel client.

4.2 - Technologies utilisées

Le serveur est développé en langage Ruby grâce au framework Ruby on Rails. Il repose sur les composants existants suivants :

- Ruby (1.9+) ;
- Ruby on Rails (3.0+) ;
- Thin server (1.2+), le serveur web qui exécute notre application ;
- nginx (0.7+), proxy et load balancer ;
- PostgreSQL, serveur de base de données.

Nous avons choisi le framework Ruby on Rails car c'est une technologie récente et agile, qui permet de développer des application web de façon rapide et maintenable. De plus, c'est une technologie très suivie et mise à jour régulièrement.

Thin est l'un des serveurs compatibles avec Ruby on Rails les plus légers et rapides actuellement, il convient parfaitement à nos besoins.

Enfin nous avons choisi PostgreSQL comme serveur de base de données car il est très performant et maintenu, mais Ruby on Rails supportant la quasi-totalité des serveurs de bases de données actuels, il nous sera très facile d'en changer si nécessaire.

4.3 - Modèle de données

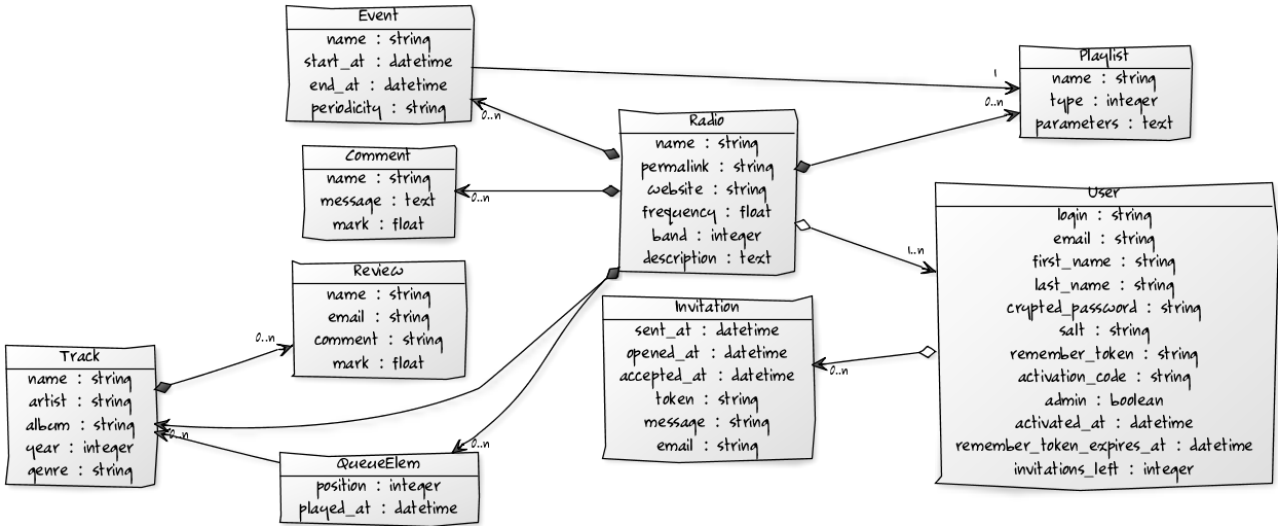

Illustration 6 : Serveur - Diagramme de classes

La base de données serveur permet de stocker certaines informations du logiciel de radio (telles que la file de lecture, les listes de lecure ou les événements), afin de pouvoir les afficher sur le site et permettre à l'utilisateur de modifier à distance ces informations, qui seront mises à jour au niveau du serveur, avant de les synchroniser avec le logiciel client.

Cette base de données contiendra également les comptes utilisateurs ainsi que leurs radios, pour permettre leur authentification et le futur contrôle des licences.

4.4 - Interactions extérieures

Le serveur (ici en violet) interagit avec les autres composantes de notre projet via le réseau local ou distant grâce aux protocoles suivants (voir schéma ci-contre).

La sécurité a été adaptée aux contraintes de chaque situation.

Le serveur radio (ici en bleu foncé) est le seul dont nous ne contrôlons pas la partie logicielle. Il s'agit d'un éventuel serveur web appartenant à la radio, auquel nous offrons la possibilité de récupérer les informations de diffusion du logiciel client.

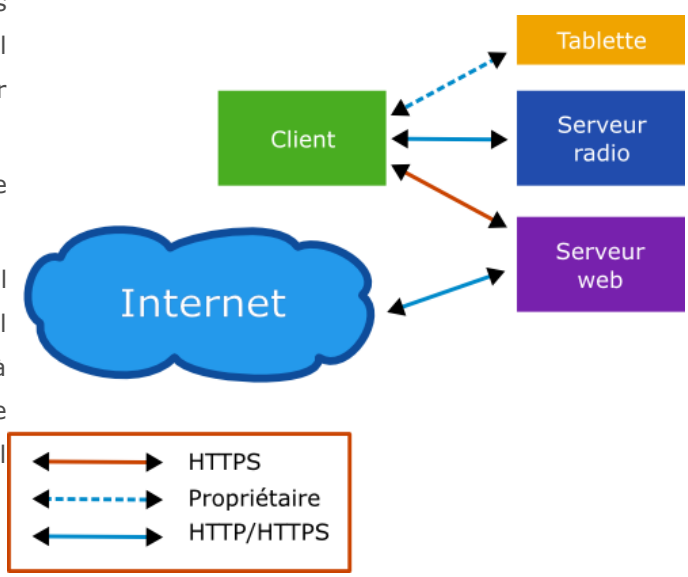

Illustration 7 : Serveur - Protocoles utilisés

4.5 – API Client

L'API client se situe entre le logiciel client et le service web. Sa fonction principale est de transmettre les informations de la radio sur le site web, de telle sorte que l'utilisateur puisse les modifier à distance. Cette API est privée et uniquement accessible par un client authentifié.

C'est une API HTML RESTfull pouvant utiliser indifféremment XML ou Json comme format de données.

Voici une liste rapide des actions possibles, avec leur URL RESTfull, les verbes HTTP utilisables et les formats servis.

URL	Verbes	Formats servis
/radios/nom-de-la-radio	GET POST	XML, JSON (Informations sur la radio)
/radios/nom-de-la-radio/playing	GET POST	XML, JSON (File de lecture)
/radios/nom-de-la-radio/history	POST	XML, JSON (Historique de lecture)
/radios/nom-de-la-radio/stats	GET POST	XML, JSON (Statistiques de la radio), PNG (Graphiques)
/radios/nom-de-la-radio/comments	GET	XML, JSON (Commentaires sur la radio)
/radios/nom-de-la-radio/tracks/3/rate	GET	XML, JSON (Notes sur une piste)
/radios/nom-de-la-radio/planning	GET POST	XML, JSON (Planification)
/radios/nom-de-la-radio/license	GET POST	XML, JSON (Contrôle de la licence logicielle)

4.6 – API Publique

L'API publique permet au site web de la radio de récupérer des informations en live directement depuis notre service web, ce qui est beaucoup plus simple que de les récupérer depuis le logiciel.

C'est également une API HTML RESTfull très simple d'utilisation.

L'accès à ses fonctions peut être publique ou restreinte selon les préférences de l'administrateur de la radio.

URL	Verbes	Formats servis
/radios/nom-de-la-radio	GET	XML, JSON, HTML (Informations sur la radio), PNG (QR code de la radio)

/radios/nom-de-la-radio/playing	GET	XML, JSON, HTML (File de lecture)
/radios/nom-de-la-radio/history	POST	XML, JSON, HTML (Historique de lecture)
/radios/nom-de-la-radio/stats	GET	XML, JSON, HTML (Statistiques de la radio), PNG (Graphiques)
/radios/nom-de-la-radio/comments	GET POST	XML, JSON, HTML (Voir/ajouter un commentaire)
/radios/nom-de-la-radio/tracks/3/rate	GET POST	XML, JSON, HTML (Voir/ajouter une note), PNG (Représentation de la note)

5 - Application Mobile

5.1 - Architecture

L'application mobile procure à son utilisateur la possibilité d'effectuer diverses actions directement sur l'application client, telles que :

- Lecture ;
- Pause ;
- Stop ;
- Stop en fin ;
- PushToTalk ;
- Crossfader ;
- Réorganisation de la liste de lecture.

5.2 - Technologies utilisées

Le développement se fait dans un premier temps pour les périphériques Android. L'IDE utilisé sera Eclipse, en complément au SDK Android fourni par Google. Le développement en JAVA permet d'exploiter au maximum les possibilités offertes par Android. Quelques tests ont été effectués avec Titanium dans sa version 1.4, qui à l'heure actuelle, semble poser quelques problèmes avec les connexions réseau.

5.3 - Interactions

L'application est, dans la plupart des cas, embarquée sur du matériel de type tablette tactile. Le matériel doit, dans tous les cas, disposer d'une connexion lui permettant de communiquer avec l'application cliente. L'interaction se fait à l'aide d'une connexion TCP. Les données sont réceptionnées côté client par le Network Core, qui les transmet au Plugin API mobile.

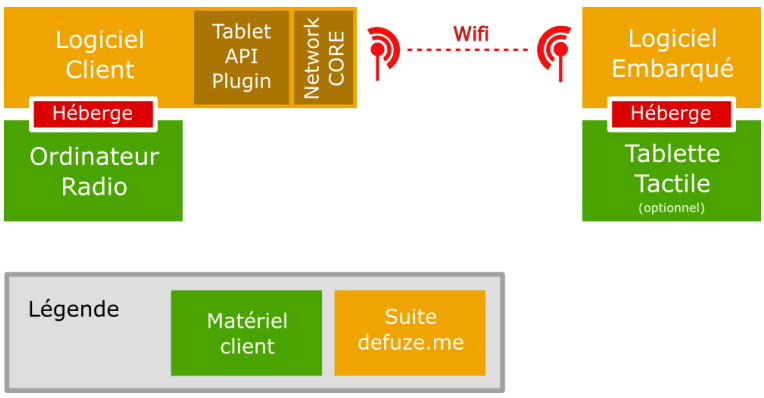

Illustration 8 : Application mobile - Interactions

5.4 – API

Voir Annexe B - API Mobile.

6 - Bugs connus

6.1 – Site web

1. Traduction anglaise manquante sur un message d'invitation ;
2. Le bouton « Rejoindre une radio existante » est, pour l'instant, sans effet.

Annexes

A - Table des illustrations

B - API mobile

```
Defuze.me                          Mobile API                          2010-2012


Revision    Date          Author            Description
-----------------------------------------------------------
v0.1        2011-01-22    Adrien Jarthon    Architecture & Events
v0.2        2011-03-31    Adrien Jarthon    Authentication + TCP format update


The mobile API allows the Defuze.me mobile application to communicate with the
client software. The communication is etablished through the TCP protocol, and
the data formatting using JSON encoding.

1. Message architecure
----------------------

The communication between the mobile application and the client software will be
2-way and event-oriented. Each pair can send messages to the other, multiple
messages can be sent at once. One message describe one event. One or more
messages CAN reply to another one if an answer is needed, but not necessary
right after the request, tons of messages can be send between a request and its
answers.

1.1 Message format

Each message will be represented by a JSON hash containing:
* the type of event
* the UID of the message (choosed by sender)
* the UID of the message we reply to (optional)
```

```
* any additional event-dependent data

Example of message:
{
      event:      'newQueueTrack',
      uid:  345,
      data: {
            track:      {
                  title:      'Dezzered',
                  artist:     'Daft punk',
                  album:      'Tron legacy OST',
                  year: '2010',
                  id:         4987
            },
            position:   12
      }
}

Every key and event name will be camelCased.

This way of asynchronous and non-blocking communication handling (using reply
uid) allows us to keep the app very reactive. For example if the mobile app
trigger a huge library search, nothing will block and the user is still able to
use the live controls and the playing queue while the search is being processed
by the client.

1.2 TCP format

The JSON flow will be formatted this way:

{message1}\0{message2}\0{message3}\0

It's a list of messages (json hash) separated by a null char ('\0'). When
receiving data, the client will have to put the different TCP packets together
until they form a valid json hash. The trailing '\0' at the end of each message
will help the receiver split the data flow. Every binary data will be encoded
using JSON string format or base64 to avoid the null char inside of our content.

2. Events
---------

2.1 Live control

Live control events can be sent by the client to update the mobile app display
of by the mobile app to control the client software.

Available events:
* play
* pause
* stop
* next
* talk
* endTalk

All this events have no additionnal parameters. If sent by the mobile app, the
client MUST answer to the mobile app request with one of the following event:
* ok
* noChanges (if the control was already in the desired state)
```

```
* error (SHOULD contain additional error informations)
```

For the mobile app to know what are these answers for, the client MUST specify the request event id in the 'replyTo' field of the message hash.

Example:

```
(mobile -> client)
{
      event:            'play',
      uid:        42
}

(client -> mobile)
{
      event:            'ok',
      uid:        375,
      replyTo:    42
}
```

2.2 Authentication

Once connected to the client, the mobile app will have to authenticate itself, and wait for acceptance from client side. When launched for the first time, the mobile application must generate a random token and store it for further use.

Then will be a base64 string of length 16.
ex: d2KI1ONxc8123bJF

On connection, the client will send an initial event requesting the application's identification token:

```
{
      event:      'authenticationRequest',
      uid:  32,
}
```

The mobile application must then answer like this:

```
{
  event: 'authentication',
  uid:  765,
  replyTo: 32,
  data: {
    token: 'd2KI1ONxc8123bJF',
    appVersion: '0.1',
    deviceName: 'Samsung S100 Galaxy Tab'
  }
}
```

The client will then check the informations and ask the user if necessary.
If the app is accepted, the mobile app will receive:

```
{
  event: 'authenticated',
  uid: 92,
  replyTo: 765,
}
```

Now every regular event described in this document can be received and sent.
If the authentication fail, the client will send an event like this:

```
{
  event: 'authenticationFailed',
  uid: 93,
  replyTo: 765,
  data: {
    message: "Sorry, your version is too old, please upgrade"
  }
}
```

And the connection will instantly be closed by the client.

2.3 Playing queue

The client software will continously stream to the mobile app every modification to the playing queue using one of the following event:
* popQueueTrack
* newQueueTrack
* removeQueueTrack
* moveQueueTrack

Each "track" in the queue will have a position relative to the currently playing track. The playing track will have the position 0, the next will have the position 1 and the last -1. When the current track fade out to the next track, at the end of the fade (when the first track is unloaded) the second track become the first and optain the position 0, every track position in the queue is shifted by -1.

When this does arrive (at each track change) the client will send a "popQueueTrack" event to the mobile app. The message will look like this:

```
{
      event:      'popQueueTrack',
      uid:  42,
}
```

When a new track is present in the queue (added on client side), the client will send a "newQueueTrack" message looking like this:

```
{
      event:      'newQueueTrack',
      uid:  345,
      data: {
            track:      {
                  title:      'Dezzered',
                  artist:     'Daft punk',
                  album:      'Tron legacy OST',
                  year: '2010',
                  id:         4987
            },
            position:   12
      }
}
```

In that case, the mobile app will have to insert the track at the given position

and shift any later track (position >= 12) by +1. If the given position is negative (history), the older tracks are shifted by -1.

When a track is remove from client side, the client will send the following message:

```
{
      event:      'removeQueueTrack',
      uid:  345,
      data: {
            track:      {
                  title:      'Dezzered',
                  artist:     'Daft punk',
                  album:      'Tron legacy OST',
                  year: '2010',
                  id:         4987
            },
            position:   12
      }
}
```

In this case, the mobile app will have to shift the later tracks (> 12) positions by -1. The "moveQueueTrack" event is a combination of the 2 previous events. It is intended to move a track inside the queue and will look like this:

```
{
      event:      'moveQueueTrack',
      uid:  345,
      data: {
            track:      {
                  title:      'Dezzered',
                  artist:     'Daft punk',
                  album:      'Tron legacy OST',
                  year: '2010',
                  id:         4987
            },
            position:         12
            oldPosition:      42
      }
}
```

Of course, the mobile app will have to shift the ids accordingly.

When a queue change occur on the mobile app side, the mobile application must send one of the previous events (except popQueueTrack) and the client will reply with on of the available answer event (ok, noChanges or error).

2.4 Music library