Projet de développement
Gopiwan

Rapport Technique
Master 1 *Web Multimédia Réseau*

David BARRAT
Nicolas BLOYET
Nailya BOGROVA
Guillaume CLAUDIC

24 mai 2015

Table des matières

1 Contexte du projet

Le projet Gopiwan ici présenté est un projet universitaire encadré par M. Réda Kadri.

Le client, M. Frédéric Raimbault, souhaitait qu'à l'issue du projet, il puisse avoir un logiciel permettant de communiquer avec le robot, à distance et obtenir également un affichage de "ce que voit" le robot. L'interface utilisateur n'était pas le point clé de l'application, tant que cette dernière fût exploitable et adaptable.

Pour ce faire, il nous a été fourni :

— Un raspberry Pi (première génération, monocore)
— un kit de développement Gopigo (châssis, carte fille robot, moteurs, visserie...)
— 8 piles rechargeables pour alimenter le robot
— Une tablette nexus tournant sous Android ; le projet étant à la base prévu pour du tout java, nous voulions développer sous Android mais nous avons été amené pour une plus grande flexibilité des clients, à passer par une interface HTML,
— Un chargeur Usb "mural" (sans câble)

Notre client, souhaitait avoir un logiciel intégralement écrit en Java, si possible ; une pile protocolaire nous a donc ainsi été proposée, mais nous sommes venu à la conclusion qu'il faudrait l'adapter, certaines briques de la pile étant incompatibles avec le matériel fourni. On a également dû utiliser des librairies python fonctionnelles, car celles en Java n'étaient pas encore sorties au début du projet.

2 Mise en marche

2.1 Démarrer le GoPiGo

La mise en route du robot contrôlé à distance se passe sans que l'utilisateur n'ait à se préoccuper d'aucun paramétrage. Lors de la livraison du robot, il y aura tout le nécessaire pour que le robot fonctionne "out-of-the-box". En effet, grâce à des scripts de démarrage préalablement installés, le GoPiGo démarrera dès qu'il sera sous tension :

— le wifi en mode hotspot
— le serveur http Jetty

Le Pi sera donc dès lors accessible à l'adresse *192.168.42.1* sur le réseau wi-fi de SSID *GOPIGO* (réseau ouvert).

2.2 Connecter un appareil sur le GoPiGo

Comme précisé précédemment, le GoPiGo déploie à son démarrage un point d'accès Wifi (802.11 b), à ce dernier nous avons attribué l'adresse *192.168.42.1* statique. Pour y connecter un appareil, il suffit de joindre le réseau wifi "GOPIGO", avec une ip elle aussi statique appartenant au réseau *192.168.42.0*

/ 255.255.255.0, le serveur dhcp installé sur le Raspberry Pi étant malheureusement très instable (nous avons à de nombreuses reprises testé différentes méthodes disponibles sur internet pour placer le raspberry pi en hotspot wifi, et en utilisant DHCP, l'attribution des adresses automatique ne fonctionnait plus au redémarrage du Pi).

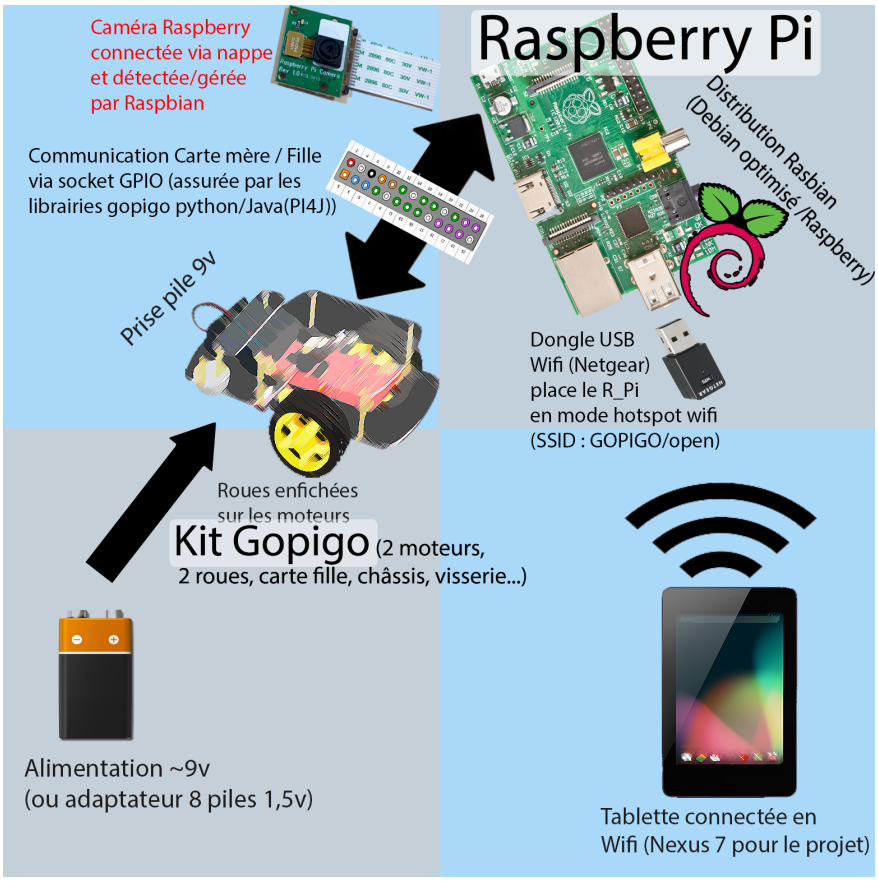
Le GoPiGo est ainsi, ouvert à tout contact par requête HTTP sur le port *8080*, par exemple l'accès à l'interface embarquée, qui se fera dans un navigateur à l'adresse :

http ://192.168.42.1 :8080/interface

2.3 Note concernant le Wi-Fi

En l'état actuel des choses, le wi-fi est relativement instable, c'est à dire que la connexion peut se perdre... Le problème pourrait venir du dongle lui-même ou bien de son antenne.

3 Intéractions matérielles



3.1 Alimentation

L'alimentation du robot peut être effectuée de diverses façons, ce dernier requiert 5V pour fonctionner, et 9V pour faire fonctionner :

— par pile(s) LR6 (couplage série de piles 1.5V via adaptateur)
— par pile 6L6R1 (carrée 9V)
— par USB qui fournit un courant continu de 5V (suffisant à faire fonctionner le Pi, et permet de fournir de l'énergie aux moteurs pour tester leur fonctionnement.

3.2 Kit GoPiGo

Il s'agit d'un kit propulsé par Dexter Ind. *www.dexterindustries.com/GoPiGo/* comprenant le nécessaire pour transformer un Raspberry Pi en un robot fonctionnel. Le kit comprend un châssis basique, des roues en plastique, de la visserie (conçue pour soutenir un Raspberry Pi, la carte gopigo, les roues etc... Pour entrer en intérraction avec le robot, il sera nécessaire de passer par une interface WLAN0 (que je décrirai plus tard) et d'utiliser un logiciel pour convertir les commandes reçues, en ordres pour le robot.

3.3 Raspberry Pi

Le Raspberry Pi fourni est de première génération, mono-coeur, et assez limité par ses ressources, c'est pourquoi nous avons du faire des choix dans les technologies à utiliser pour notre artéfact final. La distribution linux installée est une Raspbian de 2014 (modifiée par Dexter pour pouvoir fonctionner avec la carte Gopigo, code exemple python, scratch fourni, mais tout à fait perfectible, non optimisés...).

3.4 Gopigo carte fille

La carte Gopigo est un PCB embarquant un chipset permettant de contrôler des moteurs, des prises pour les alimenter, et les ordres reçus par le Pi sont transmis par un port GPIO dont je vais expliquer le fonctionnement dans un point suivant.

3.5 Connection GPIO

Pour communiquer avec la carte fille, le raspberry pi et la carte gopigo sont connectés via un port GPIO, afin de transmettre les informations au chipset de la carte qui les convertira en ordres pour les moteurs qui enclencheront les roues.

3.6 Camera Raspberry

La camera du Pi est reliée et alimentée par le biais d'une nappe sur la carte mère du Raspberry. Les données transitent par cette même nappe. Pour activer cette caméra sur le Raspberry, il faut utiliser la commande *raspi-config*.

3.7 Dongle USB

Le dongle fourni était un adaptateur Wifi Netgear utilisant la connectique usb pour s'auto-alimenter. Il correspondra à l'interface WLAN0 sur le Pi, et sera mis en mode hotspot 802.11b grace à la commande hostapd et une configuration de l'interface, cependant, sa stabilité est assez douteuse, nous avons opté pour une adresse IP statique qui nous a permis d'obtenir les meilleures performances et le meilleur fonctionnement du robot.

3.8 Tablette tactile + Connexion Wifi

Nous avons effectué nos tests depuis différents périphériques externes, dont la tablette Asus Nexus tournant sous Android ; nous sommes parvenus à contrôler le robot et à obtenir son affichage par le biais de la connexion Wifi de l'appareil. Une fois que les deux appareils sont synchronisés, l'utilisation du robot se fait de manière transparente sur la tablette.

4 Dépendances à Python

Lors du développement de notre application, nous avons été confrontés à des problèmes techniques : matériels, mais aussi logiciels ; En effet, au début du projet, les librairies Java n'existaient pas encore, on a donc opté pour l'utilisation des librairies python Gopigo déjà existantes, qu'on appelle de deux façons différentes :

— Un démon python pour des communications par socket TCP

— Des appels directs à l'interpréteur python au cours du programme

Dans le premier cas, on lance un démon python à l'écoute sur un port TCP qui se charge de recevoir des appels qu'on lui envoie et se charge de lancer les commandes correspondantes sur le pi. C'est une méthode qui nécessite donc l'exécution continue d'un programme supplémentaire : un démon Python qui se chargera de recevoir tout au long de son fonctionnement des commandes par TCP et de les traduire. L'autre solution permet de s'affranchir de démon, en faisant appel à un interpréteur python à chaque appel, exécutant alors des scripts préalablement placés dans */home/pi/scripts*.

Chaque méthode a ses avantages et ses inconvénients, aussi nous avons choisi de mettre les deux à disposition, à choisir selon l'usage. En effet, dans le cas où le robot ne se déplacerait pas souvent, il est plus économe de faire des appels ponctuels à l'interpréteur sans s'encombrer d'un processus supplémentaire, tandis que si les mouvements sont fréquents, le démon se révèlera plus pertinent.

La question des coûts était primordiale dans ce projet, car le GoPiGo dispose d'une part de peu de ressources, le CPU étant de faible puissance, et devant potentiellement gérer un streaming vidéo, et d'autre part, il s'agit d'un système fonctionnant sur batterie, l'autonomie de l'appareil est donc en jeu.

Il est donc possible de changer de mode de communication en cours même d'utilisation.

5 Interface Client

5.1 Principe de fonctionnement

Les intéractions entre l'interface client sont simples, une méthode send en Javascript qui se charge d'envoyer des requêtes de type GET, sans rechargement de page (avec Ajax, et JQuery) et en exploitant au maximum l'API REST définie dans l'interface du serveur.

```
function send(order)
{
     $.get(adresseDuPi + order);
};
```

Par exemple, si l'on souhaite faire reculer le robot (les ordres sont décrits dans l'interface GoPiGo rest et bien documentés).

```
<input type="button" value="Reculer"
    onClick="send('/gopigo/move/backward')" />
```

5.2 Interface embarquée

Le GoPiGo dispose d'une interface embarquée, qui se trouve être une page html simple, permettant très basiquement d'envoyer les requêtes adéquates au contrôle du GoPiGo via des boutons, et d'afficher son flux vidéo. Le choix d'html s'est fait de part pour son universalité, mais aussi de sa simplicité.

En effet, du fait de l'API que nous avons developpé et dont nous reparlerons ultérieurement, la seule chose que l'interface ait à faire est d'envoyer des requêtes HTTP à l'adresse et au port adéquat. Cette charge est donc très limitée et peut se réaliser très rapidement, notamment au sein d'une page HTML, avec le mécanisme AJAX.

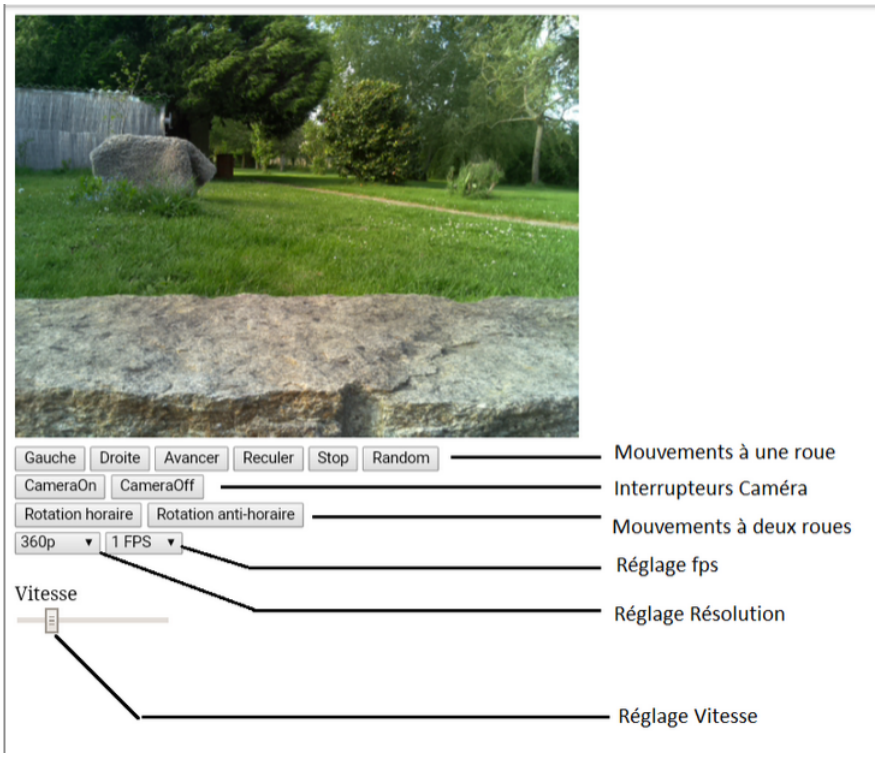
Or, une page html équipée de javascript offre une compatibilité très large de matériel : tout appareil disposant d'un navigateur est ainsi en mesure d'accéder à cette interface et de pouvoir contrôler le robot, que ce soit un téléphone de n'importe quelle marque, un ordinateur traditionnel, ou autre...

Il n'y a donc de fait aucune contrainte de plate-forme.

Cette interface embarquée est accessible à l'adresse :

http ://192.168.42.1 :8080/interface

5.3 Mode d'emploi de l'interface embarquée


Gauche
Droite
Avancer
Reculer
Stop
Random
CameraOn
CameraOff
Rotation horaire
Rotation anti-horaire
360p
1 FPS
Vitesse
Mouvements à une roue
Interrupteurs Caméra
Mouvements à deux roues
Réglage fps
Réglage Résolution
Réglage Vitesse

5.3.1 Barre d'adresse

Comme énoncé précedemment, pour accéder à l'interface Gopigo, il est nécessaire de saisir l'adresse de l'interface wlan0 (par défaut) du raspberry pi. Ici, l'adresse du pi est 192.168.42.1 et le serveur tourne sur le port 8080. L'adresse d'accès à l'interface est 192.168.42.1 :8080/interface

5.3.2 Vue caméra

Lorsque l'utilisateur charge l'interface (et que tout s'est bien passé sur le pi), il aperçoit l'affichage reçu par la caméra du robot.

5.3.3 Commandes robot/ camera

La troisième partie de l'interface concerne les commandes utilisateur,

5.4 Directions

— Gauche, permet de faire tourner la roue gauche du robot, qui aura pour incidence de le faire tourner vers la gauche.
— Droite, permet de faire tourner la roue droite du robot, qui aura pour incidence de le faire tourner vers la droite.
— Avancer, permet de faire tourner les deux roues du robot vers l'avant, qui aura pour incidence de le faire avancer.
— Reculer, permet de faire tourner les deux roues du robot vers l'arrière, qui aura pour incidence de le faire reculer.
— Stop, permet d'arrêter la rotation de la ou des roues, si elles sont en activité.

5.5 Camera

— CameraOn, permet d'activer la caméra du robot.
— CameraOff, permet de désactiver la caméra du robot.

5.6 Demi-tours

— Rotation horaire, permet de faire faire au robot un demi-tour en sens dextrogyre (sens des aiguilles d'une montre).
— Rotation anti-horaire, permet de faire faire au robot un demi-tour en sens lévogyre (sens inverse des aiguilles d'une montre).

5.7 Paramètres caméra

— Résolution, permet de modifier la résolution des images envoyées par le robot (entre 240p à Full HD, 360p conseillée).

— Frames par seconde, permet de modifier le nombre d'image(s) envoyée(s) par seconde, par le robot, et étant données les contraintes matérielles, les valeurs envisageables oscillent entre 1FPS et 10FPS.

5.8 Vitesse du robot

Ici, nous avons opté pour un slider afin de choisir la vitesse du robot, plutôt que de laisser à l'utilisateur la possibilité de saisir des valeurs de vitesse incohérentes. On peut choisir entre une vitesse allant de 0 à 255 cm.s-1

5.9 Développement d'une interface grâce à l'API

En dehors de cette interface embarquée très simple, il est possible de réaliser très simplement d'autres interfaces grâce à l'API REST mise a disposition. En effet, à partir du moment où un programme peut envoyer une requête HTTP, il peut se servir de cette API, et n'importe qui ayant lu la documentation fournie peut donc développer sa propre interface, sous la forme voulue (Page HTML, application mobile, logiciel desktop "classique", etc...).

La documentation relative à cette API est bien entendu fournie.

6 Programme serveur

Le programme serveur fourni est en fait un intergiciel, faisant l'interface entre des appels provenant d'interfaces clientes, et la librairie GoPiGo Python.

De façon très schématique, il met à disposition les ressources proposées par le GoPiGo selon le style d'architecture REST, et traduit donc les requêtes HTTP des clients en appels à l'API GoPiGo Python.

6.1 Fonctionnement

Nous avons utilisé deux librairies externes pour le developpement du serveur :

— *Jetty*, projet de la fondation Eclipse

— *Jersey*, implémentation de la spécification *JAX-RS* par Oracle

Jetty est un serveur HTTP et moteur de servlet, qui nous sert donc à créer un serveur HTTP à l'écoute sur le port *8080*, et dont certaines classes constitueront des servlet, qui en l'occurrence délivreront le résultat de l'éxécution des commandes.

Ces servlet seront annotés grâce à *Jersey*, qui nous permettront de définir les verbes HTTP adéquats, les *paths* relatifs à chaque fonction, mais aussi le type de données produites.

Pour information, les fonctions de mouvement délivrent une réponse en texte brut, tandis que l'interface est délivrée en html, et les frames vidéo en bitmap.

Les annotations Jersey permettent donc d'obtenir un code combinant lisibilité et élégance et adaptable aux souhaits du développeur.

Ci-dessous un exemple commenté d'utilisation de telles annotations :

```
@Path("/gopigo")          // on definit le path d'acces au servlet
public class MoveService {
 // ...
    @GET       // cette fonction répondra au verbe GET...
    @Path("/stop")    // ... utilisé a l'adresse /gopigo/stop...
    @Produces(MediaType.TEXT_PLAIN) // ... et rendra sa réponse sous
                                    // forme de texte brut
    public String stop() throws IOException {

        Globals.gopigo.stop();       // appel au matériel
        return "Stopped !\n" ;       // réponse rendue
    }
```

6.2 Architecture

Le programme a été fait de façon la plus modulaire possible. Les classes ont été regroupées par package suivant leur rôle.

6.2.1 Packages

Le programme serveur est fourni sous forme d'un jar éxécutable, organisé comme suit :

```
src/
├── server/ ..... Sources serveur
│   ├── Globals.java
│   ├── HttpServer.java
│   ├── Launcher.java
│   ├── daemons/ ..... Processus d'arrière plan
│   │   ├── DaemonPython.java
│   │   └── DaemonStreaming.java
│   ├── gopigo/ ..... Interface API / matériel
│   │   ├── GoPiGo.java
│   │   ├── GoPiGoMoves.java
│   │   └── GoPiGoStreaming.java
│   ├── python/ ..... Scripts python
│   │   └── macros.py
│   └── rest/ ..... Services web (API)
│       ├── HTMLService.java
│       ├── MoveService.java
│       ├── SettingsService.java
│       └── StreamService.java
└── client/ ..... Sources client
    └── index.html
```

Chaque package a donc une fonctionnalité bien précise, de façon à rendre chaque partie la plus indépendante possible.

Le package *dameons* contient tous les processus d'arrière-plan nécéssaires au bon fonctionnement du GoPiGo. Ces classes sont typiquement instanciées par *server.gopigo.GoPiGo*. Il se pourrait qu'à terme, le nombre de classes composant ce package soit amené à diminuer.

Cette classe *GoPiGo*, qui joue le rôle d'interface entre les appels de l'API et le matériel, est en quelque sorte le point central du programme, le point d'échange des autres packages. L'utilisation de cette classe se justifie du fait que le mode de communication avec le matériel peut évoluer au cours du temps, suivant les librairies utilisées pour communiquer à la carte GoPiGo.

Le package python contient tous les scripts nécésssaires au fonctionnement du GoPiGo, et son contenu doit être copié au préalable dans */home/pi/scripts*, cette copie est effectuée par la personne souhaitant mettre en route le programme que nous avons développé durant ce projet.

Enfin, le package *rest* contient toutes les classes faisant office de servlet, et contitue donc le point d'entrée des requêtes client. Il faut noter que ces *Web Services* sont eux aussi indépendants de la librairie utilisée pour communiquer avec le matériel, et que de ce fait, ils ne font que faire suivre les fonctions de l'API à la fonction correspondante de *server.gopigo.GoPiGo*.

6.2.2 Schéma d'architecture

Ci-dessous, la pile protocolaire que nous laissons pour cette fin d'année, mais qui sera sûrement amenée à évoluer, au moins pour la brique concernant les librairies python qui peut être supprimée.

à gauche, nous avons la partie Jersey / Jetty. C'est le plus haut niveau du côté serveur.

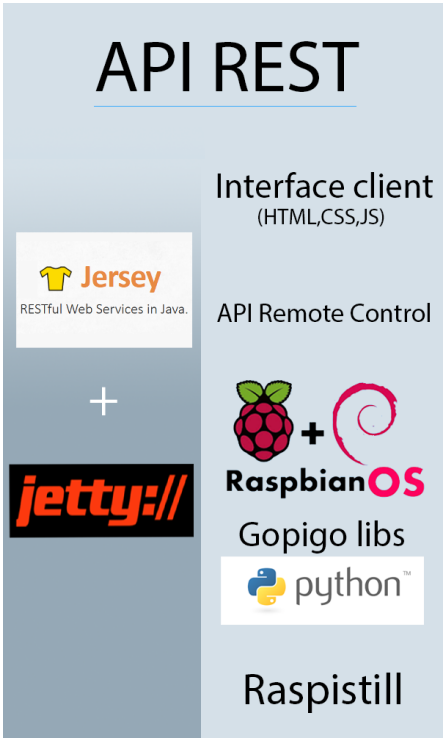
Ensuite, à droite l'interface client, générée par Jetty à partir d'une simple feuille html avec un style basique, en vue d'évolution, extrèmement adaptable, de plus il n'y a pas de dépendance, pour un code extrêmement léger côté client.

Cette interface fera office de mandataire entre les commandes envoyées par le client et le serveur Jetty, par le biais de simples requêtes GET sur des Paths définis dans les classes du package nommé 'rest'.

Je rappelle encore une fois, que le serveur Jetty (à base de Java) tourne sur la JVM d'une distribution basée sur Debian, appelée Raspbian.

Nous avons utilisé les librairies python car celles développées pour Java n'étaient et ne sont toujours pas encore suffisamment mûres à l'heure où j'écris ce rapport, mais cela commence à prendre forme ; ceci dit, le projet Dexter Gopigo étant maintenu par des développeurs visiblement indépendants, il risque de falloir mettre les mains dans le cambouis si une migration vers "du-tout-Java" est prévue pour une date ultérieure proche, même si une bonne partie du travail est déjà faite.

Et pour finir, il me semblait important de citer la commande Raspistill qui remplace très avantageusement FFMpeg, car elle permet de produire un flux "vidéo" à partir de clichés pris par la caméra en "rafale", et de fait, diminuer considérablement la charge sur le processus du Raspberry Pi.



7 Evolution possible

Comme énoncé dans ce rapport, le programme est le plus modulaire possible, et est de fait, facilement adaptable à librairie de communication GoPiGo pour Java, disponible depuis peu. Nous avons fait au mieux, pour respecter les protocoles souhaités par le client, en avons substitués certains au profit de technologies mieux adaptées (moins gourmandes dans le cas présent), afin d'obtenir du code léger et maintenable.

Il suffira en fait en très grande partie à remplacer les appels en python de la classe *server.gopigo.GoPiGo* par des appels Java correspondants. La partie démon Python pourra disparaître, car les librairies Java permettront de communiquer directement par le port GPIO avec la carte fille Gopigo. Cet intergiciel est donc facilement adaptable, sans pour autant devoir changer la partie des web services, ce qui serait dommageable si des interfaces basées sur notre API voient le jour.