Dossier Technique

Ekawa

ekow

Machine à café automatique connectée

LECOMTE Jean-Luc

BTS SNIR LaSalle Avignon 2021

Sommaire

1. Introduction

Ekawa est une machine à café connectée et pilotable par une application mobile Android.

La cafetière Ekawa est équipée d'un magasin rotatif et motorisé de capsules (8 emplacements pouvant contenir 4 capsules chacun).

L'application mobile Ekawa permettra :

- de lancer un café personnalisé
- de connaître en temps réel les différentes informations sur la machine

2. Présentation générale

2.1. Expression du besoin

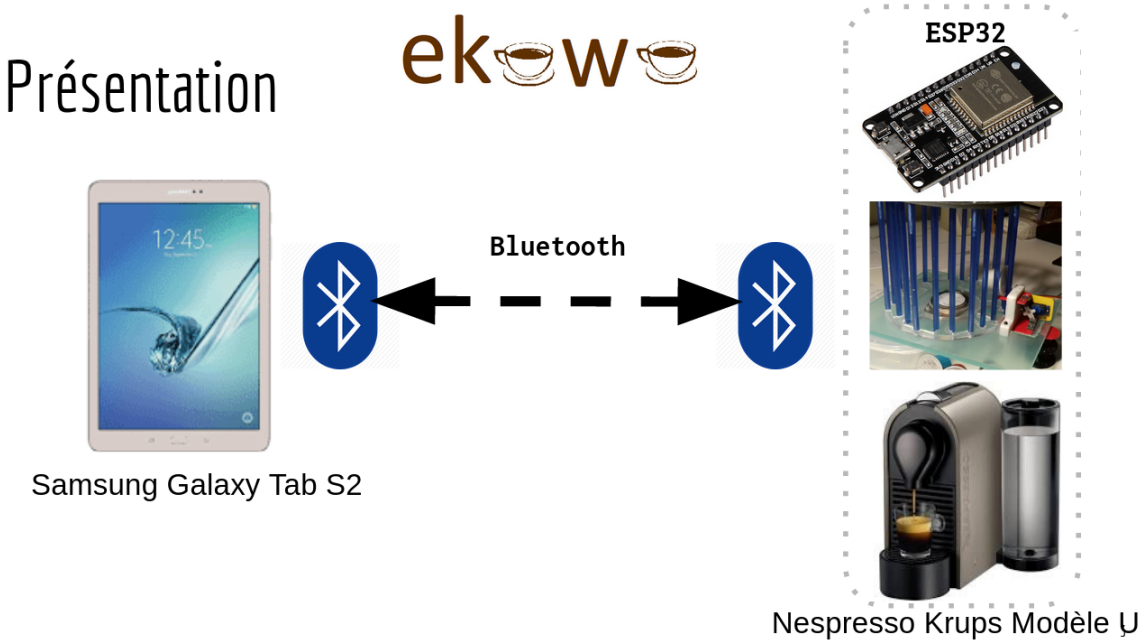
Le principe de base est simple. La connectivité du système doit permettre de piloter sa machine à café à distance. La machine à café connectée doit offrir bien plus qu'une simple fonction « télécommande ».

Les missions du système sont :

- Lancer la préparation d'un café avec la possibilité de choisir le type de capsule ("saveur et intensité") et le type de boisson (Ristretto, Espresso et Lungo);
- Alerter l'utilisateur (Niveau d'eau trop bas, Présence/Absence capsule, Présence/Absence "tasse", Machine en cours d'utilisation, Bac à capsules plein, Dureté / Qualité de l'eau en option, ...);
- Suivre l'état de vie de la machine à café (statistiques sur le nombre de boissons réalisées, cycle d'entretien, dureté / qualité de l'eau en option, …);
- Informer l'utilisateur (statistiques, analyse "santé" à partir du nb cafés/jour, taux de caféine recommandé, ...);
- Paramétrer les préférences de l'utilisateur (type de capsule et de boisson préférée);
- Programmer la préparation d'un café (horaire, délai) en option ;
- Piloter la machine à distance à partir d'une application mobile.

2.2. Architecture du système

Présentation général



Samsung Galaxy Tab S2

Nespresso Krups Modèle U

Présentation du matériel

Samsug Galaxy Tab S2

Sortie le 21 juillet 2015, la Galaxy Tab S2 de 9,7 pouces dispose d'un écran à dalle Super Amoled de 24,6 cm de diagonale au format 4:3 et à la définition de 2048 x 1536 px. Elle embarque également une puce mobile maison octo-cœur, Exynos 5433 (quadri-cœur ARM Cortex-A57 cadencé à 1,9 GHz + quadri-cœur ARM Cortex-A53 à 1,3 GHz), une mémoire vive de 3 Go, une capacité de stockage de 32 ou 64 Go extensible via l'ajout d'une carte microSD, un capteur photo-vidéo de 8 Mpx à l'arrière et un autre de 2,1 Mpx à l'avant, un lecteur d'empreinte digitale et une batterie de 5870 mAh pour alimenter le tout en énergie. Wi-Fi a/b/g/n/ac et Bluetooth 4.0 constituent le gros morceau de la connectivité sans fil. Il n'y a pas de GPS dans cette Tab S2.

L'interface utilisateur est assurée par Android 5.0.2 Lollipop (API level : 21), totalement recouvert du système TouchWiz propre et si cher à la marque coréenne.

La tablette Samsung Galaxy Tab S2 pouces était commercialisée à partir de 499 € à sa sortie.

Nespresso Krups Modèle U

Sorti en 2012, le modèle Krups Nespresso U est tout en finesse et derrière sa forme en hauteur (28,5 cm), se cache une cafetière électrique milieu de gamme que l’on peut moduler facilement.

L’avantage des machines Nespresso, c’est que d’une machine à l’autre il y a des constantes, on évite ainsi les surprises. Mais chaque modèle possède ses particularités. La krups yy1301 nespresso “u” crème pur (c’est son nom complet) peut être résumée par deux adjectifs : modulable et intelligente.

ESP32

ESP32 est une série de microcontrôleurs de type système sur une puce (SoC) d'Espressif Systems, basé sur l'architecture Xtensa LX6 de Tensilica (en), intégrant la gestion du Wi-Fi et du Bluetooth (jusqu'à LE 5.0 et 5.11) en mode double, et un DSP. C'est une évolution d'ESP8266.

Le magasin

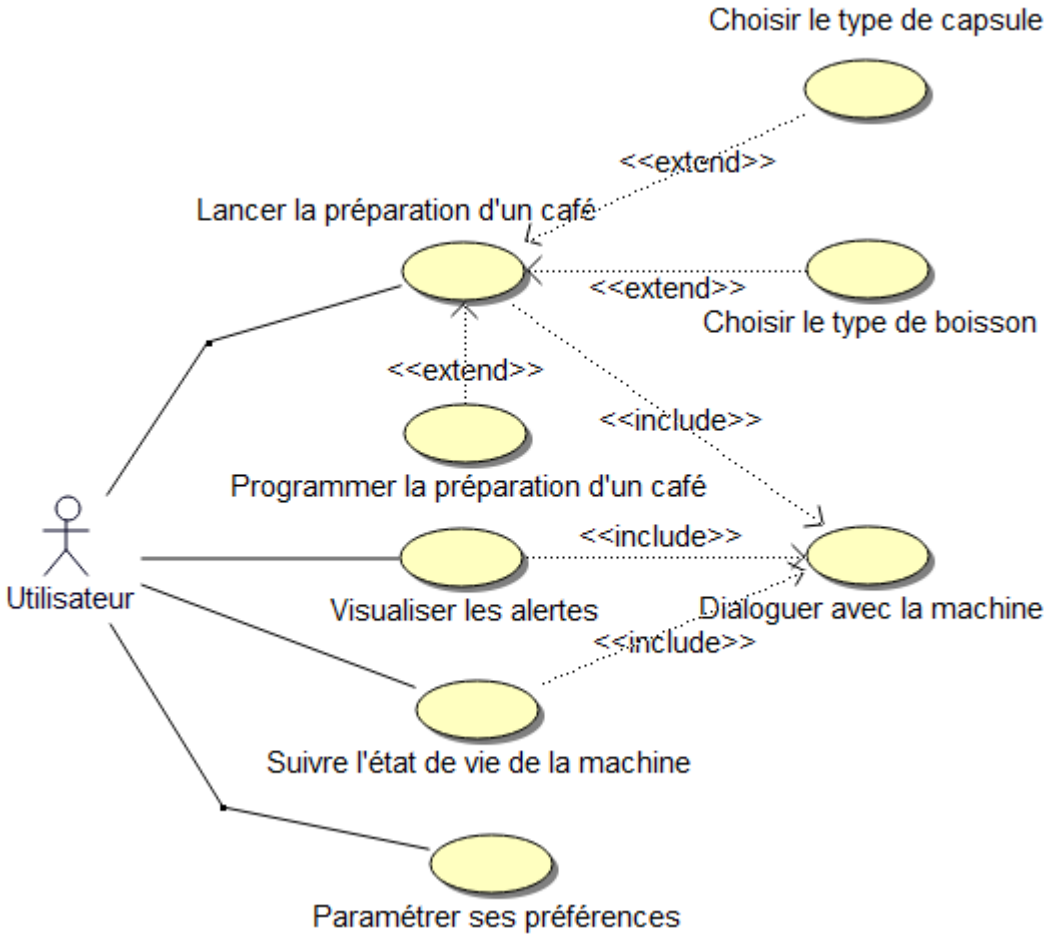
Le magasin est entièrement réalisé par les enseignants de LaSalle Avignon.
Il est composé :

- de 8 colonnes pouvant contenir 4 capsules chaqu'une;
- d’un moteur central à courant continue qui permet la rotation du magasin;
- d’un servo-moteur qui permet de faire tomber la capsule dans la machine;
- de 8 capteurs de présence en infra-rouge pour la détection des capsules.

2.3. Cas d’utilisation

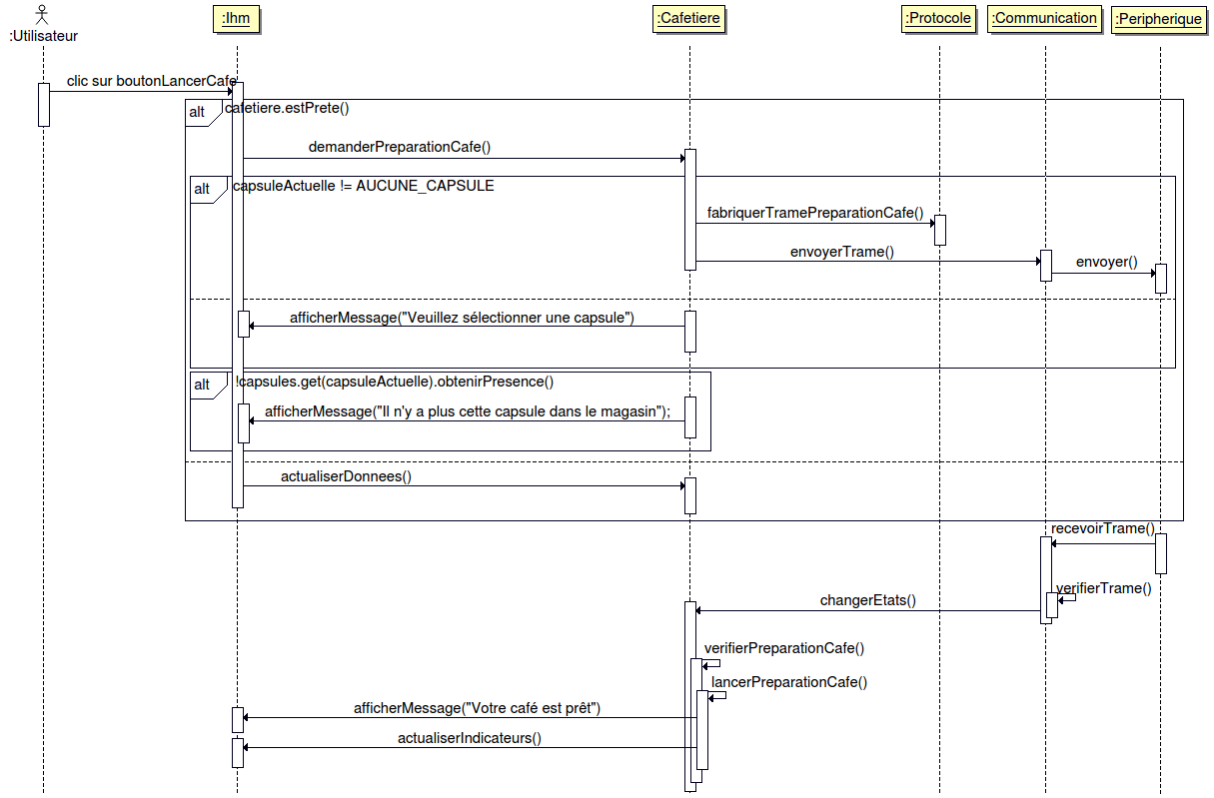
Il s’agit d’un système de contrôle à distance automatisé qui permettra :

- De lancer la préparation d’un café depuis votre smartphone android
- De vous avertir dans les cas suivants :
    - Lorsque votre téléphone n’est pas connecté avec la machine
    - Lorsque votre tasse n’est pas bien placé sur la machine
    - Lorsque le bac de la machine est plein
- De vous montrer le niveau d’eau restant dans votre machine
- De suivre l'état de la machine
- D’enregistrer vos préférences
- De programmer des préparations de cafés



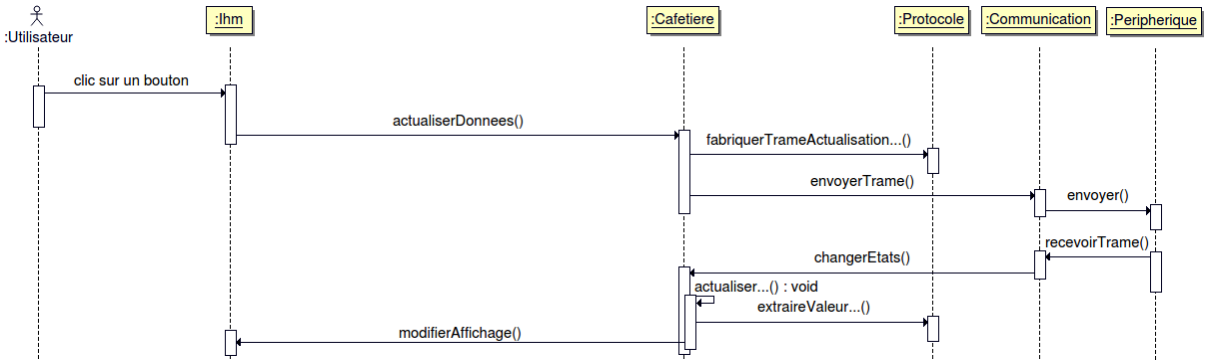
2.4. Fonctionnement général

Scénario : Lancer la préparation d'un café



Lorsque l'utilisateur appuie sur le bouton "boutonLancerCafe", le programme commence par vérifier que la cafetière est prête (qu'une capsule soit sélectionnée, que la capsule sélectionnée soit disponible, que la cafetière ne soit pas déjà en train de préparer un café, etc...). Si les conditions sont bien respectées, le programme fabrique la trame en fonction de la capsule et la boisson sélectionnée. Cette trame est ensuite envoyée à la cafetière en Bluetooth qui répondra par une réponse positive ou négative. Pour finir, le programme affichera un message en fonction de la réponse de la cafetière.

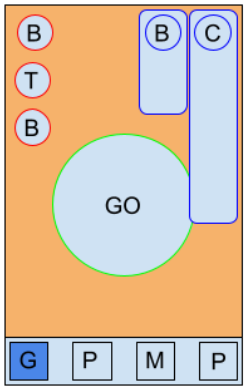
Scénario : Actualiser les données



Lorsque l'utilisateur appuie sur bouton (ou de manière périodique) le programme fabrique une trame en fonction du type de données à actualiser. Cette trame est ensuite envoyée à la cafetière en Bluetooth qui répondra par de nouvelles données que le programme va extraire et stocker avant de les afficher.

2.5. Maquette de l'IHM

Voici la maquette conçu en début de projet :



Page Générale

Indicateurs de :
- Connection Bluetooth
- Présence de tasse
- Trop plein du bac
- Informations générales

Indicateur du niveau d'eau

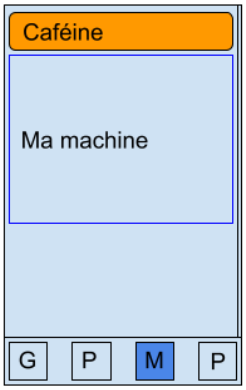
Listes déroulantes pour sélectionner le type de café et le type de boisson

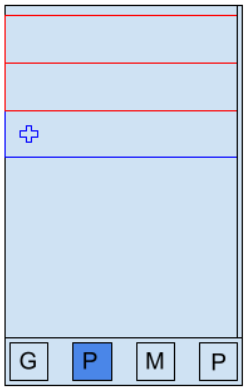
Bouton pour lancer la préparation d'un café

Page Machine

Informations sur la caféine :
- Nombre de café recommander par jour

Informations relative à la machine :
- Nombre de café effectuer
- Type de capsule
- etc...



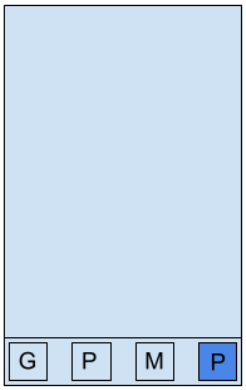


Page Programmation

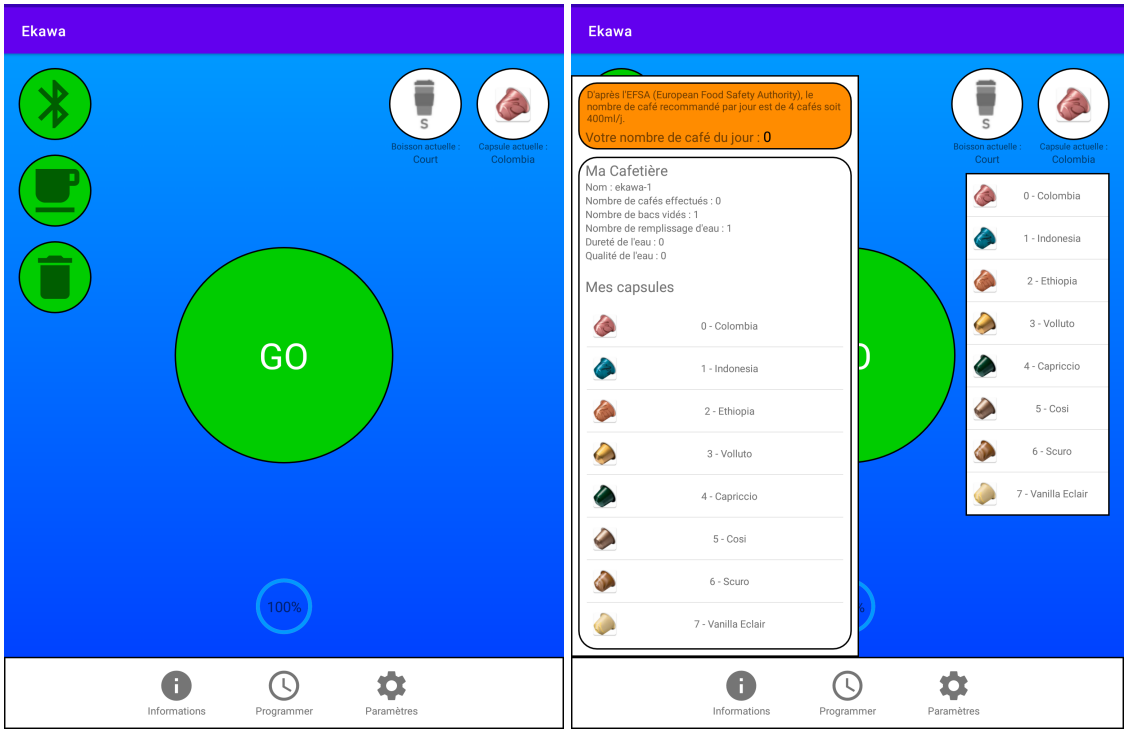
Liste avec détails des café programmer :
- Café(s) déjà programmé(s)
- Bouton pour ajouter une nouvelle programmation de café

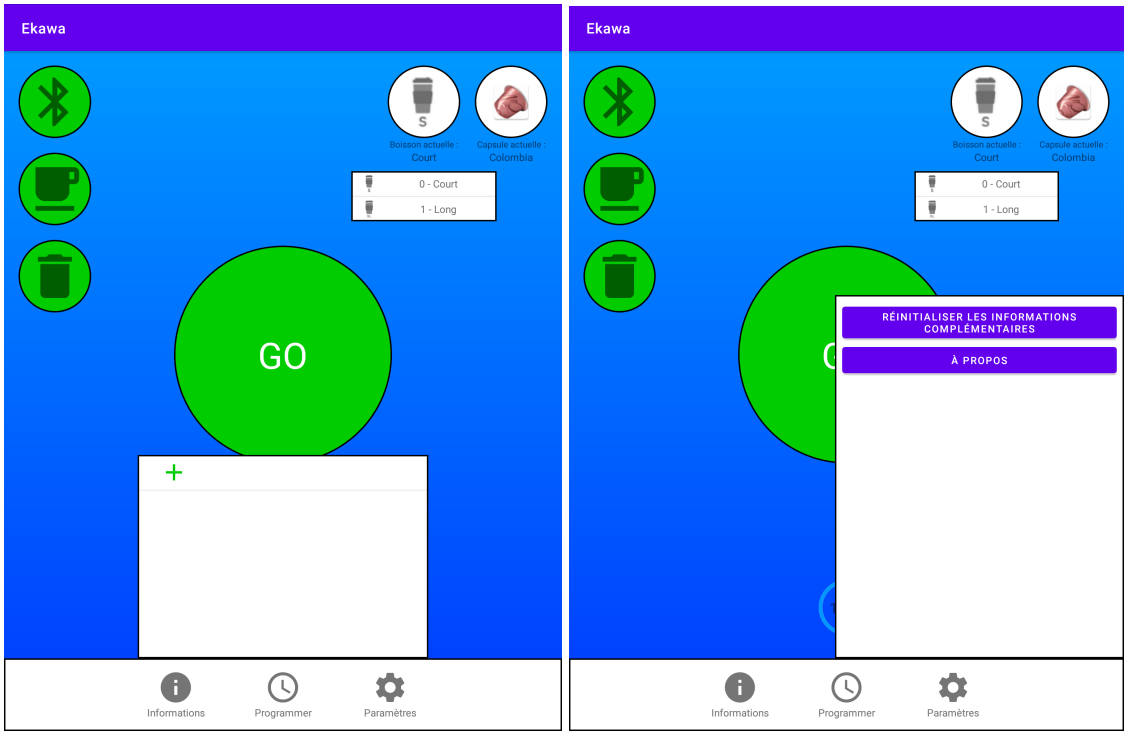
Page Paramètre

En cours de finalisation ...



Voici le résultat final :





Le design final de l'application se rapproche beaucoup de la maquette originelle.

Les changements :

- Toutes les options de l'application sont présentes sur une seule activité, contrairement à la maquette qui était censée être utilisable sur plusieurs activités.

2.6. Identification des tâches

- Lancer la préparation d'un café avec la possibilité de choisir le type de capsule et de boisson ;
- Visualiser les alertes (Niveau d'eau trop bas, Présence/Absence capsule, Présence/Absence "tasse", Machine en cours d'utilisation, Bac à capsules plein) ;
- Suivre l'état de vie de la machine à café (statistiques sur le nombre de boissons réalisées, cycle d'entretien) ;
- Paramétrer les préférences de l'utilisateur (type de capsule et de boisson préféré) ;
- Communiquer avec la machine via une liaison sans fil
- Programmer la préparation d'un café

2.7. Planification des tâches

Taches	Priorité	Itération
Créer le squelette de l'IHM	Haute	1
- Lancer la préparation de son café	Haute	2
- Choisir le type de capsule	Haute	2
- Programmer la préparation d'un café	Basse	3
Afficher des informations relative à la machine :	Moyenne	2
- Nombre de capsules restantes	Moyenne	2
- Présence de tasse	Moyenne	2
- Quantité d'eau restante	Moyenne	2
- Etat de la machine	Moyenne	2
- Statistiques sur le nombre de cafés réalisés	Moyenne	2
- Cycle d'entretien	Moyenne	2
- Dureté et qualité de l'eau	Moyenne	2
Afficher des informations relatives au café :	Basse	3
- Nombre de cafés max par jour	Basse	3
- Informations sur chaque capsule et son contenu	Basse	3
- Enregistrer des préférence utilisateurs :	Basse	3
- Types de capsules préférées	Basse	3
- Jours et Heures préférés pour lancer la préparation d'un café	Basse	3
Effectuer une connexion sans-fil avec la machine	Haute	1,2
- Détection	Haute	1
- Envoyer une trame	Haute	2
- Recevoir une trame	Haute	2
- Effectuer une connexion automatique avec la machine	Basse	3

Remarques :

- Nombre de cafés recommandé par EFSA (*European Food Safety Authority*) de 400m/j soit 4 cafés/j

2.8. Informations

Auteur

LECOMTE Jean-Luc <lecomte.jeanluc.pro@gmail.com>

Date

2021

Version

1.0

Dépôt du code source

https://svn.riouxsvn.com/ekawa-2021

3. Présentation personnelle

3.1. Rappel du besoin initial

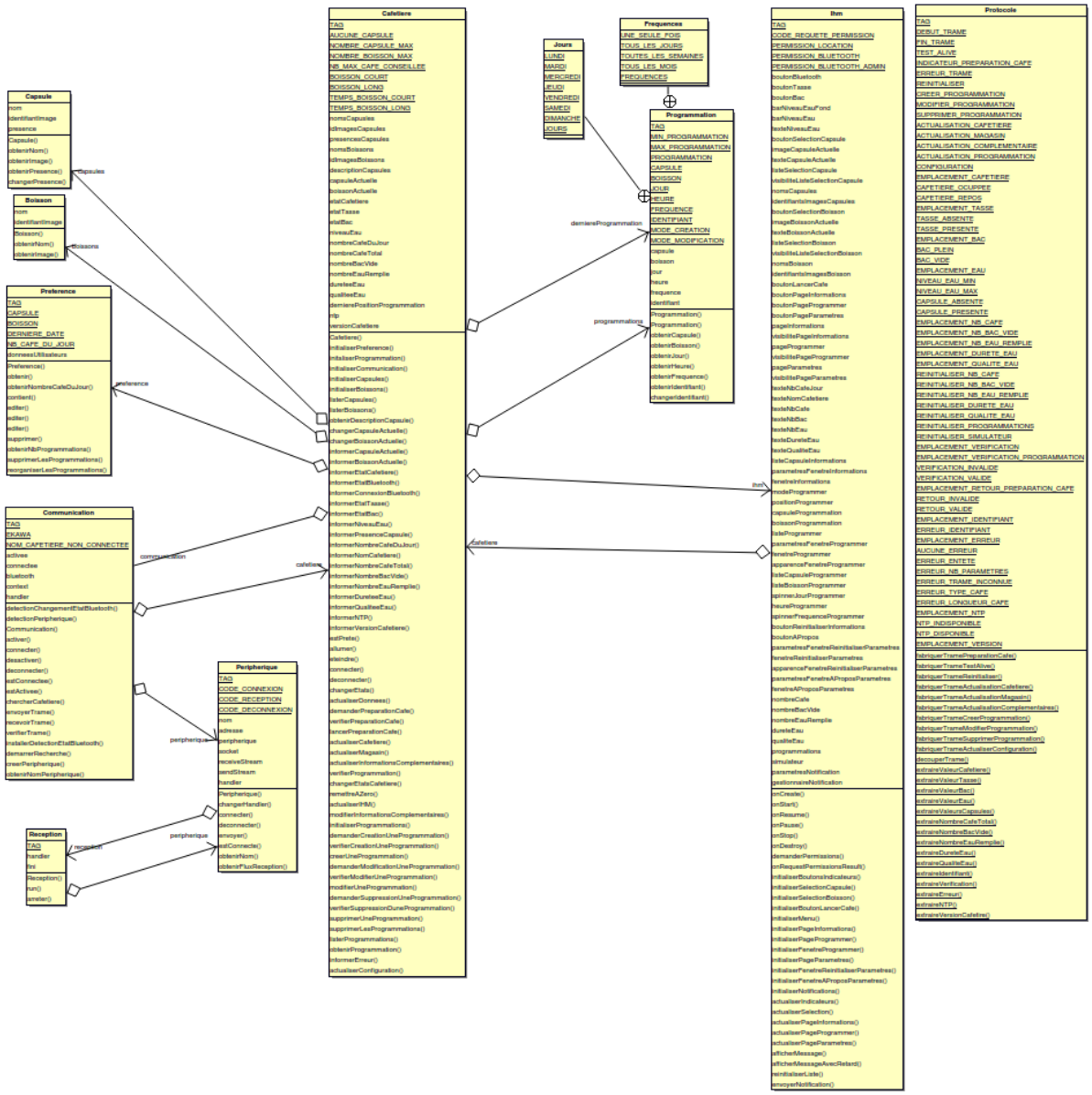
Le principe de base est simple. La connectivité du système doit permettre de piloter sa machine à café à distance. La machine à café connectée doit offrir bien plus qu'une simple fonction « télécommande ».

3.2. Organisation

Pour l'organisation du projet et la communication entre tous les membres du groupe, nous avons utilisé :

- Subversion qui est un logiciel libre de gestion de versions hébergé sur le site RiouxSVN pour l'ensemble du code source du projet.
- L'espace de stockage commun Google Drive pour tous les documents ressources.
- Gantt pour avoir une vue de la planification du projet.
- Bouml pour la partie diagramme du projet.

3.3. Diagramme de classes

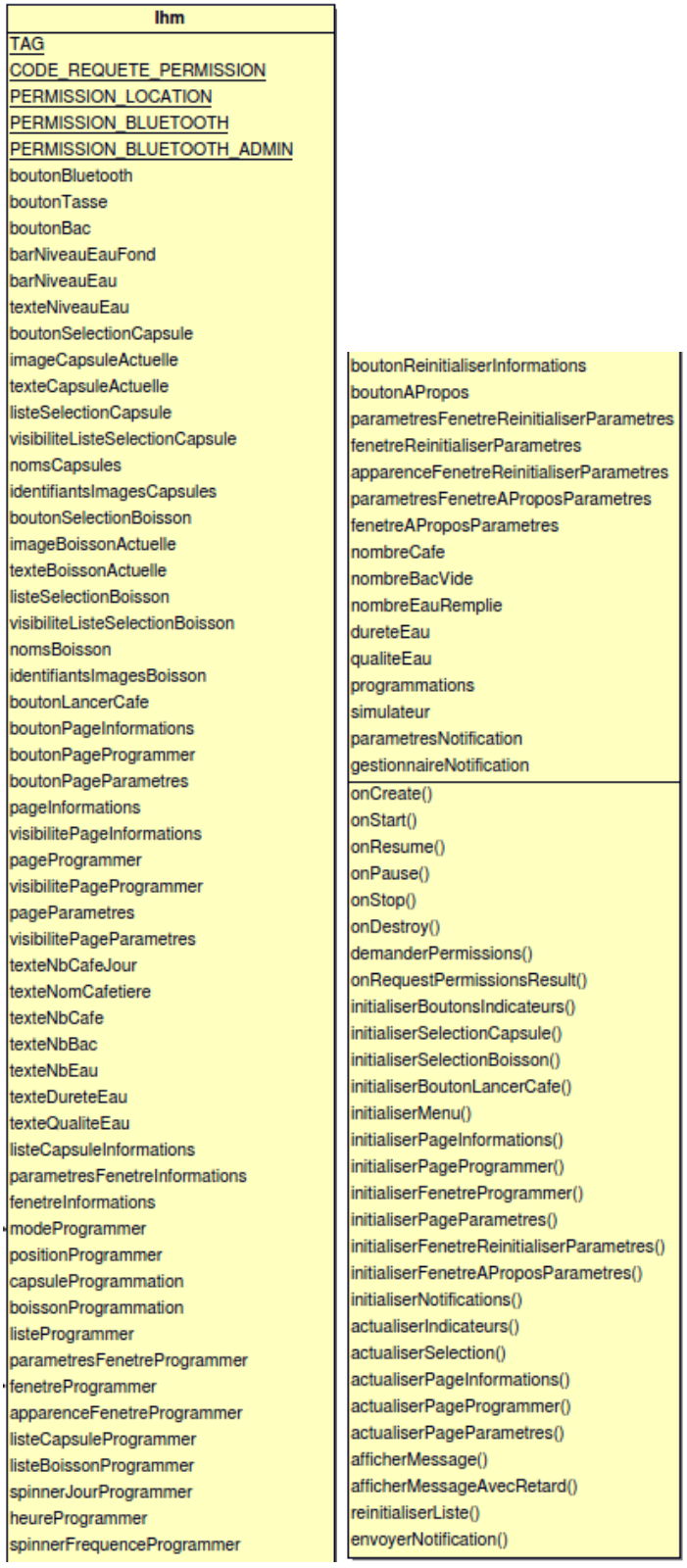


Ce diagramme de classe est disponible dans la documentation du projet.

3.4. Informations sur les classes

Ihm

La classe **`Ihm`** s'occupe de gérer toutes les fonctions d'affichage sur l'interface de l'application.



Attributs

Les attributs commençant par “visibilité” sont des booléen qui permettent au programme de retenir si une fenêtre est ouverte ou non.
Cela me permet d’afficher ou non les fenêtres en fonction des demandes de l’utilisateur.

```
view.setVisibility(View.VISIBLE);
view.setVisibility(View.INVISIBLE);
```

Méthodes

Les méthodes commençant par “actualiser” sont des fonctions qui permettent d’actualiser les indicateurs en fonction des trames reçues.

```
textview.setText(cafetiere.informer...());
view.setBackgroundResource(R.drawable. ...);
imageview.setImageResource(R.drawable. ...);
```

La méthode “afficherMessage(String texte)” permet d’afficher un message dans le Toast d’android.

Un Toast est considéré comme un message d'information, d'avertissement. Ce message est dit transitoire car il ne nécessite aucune intervention de la part de l'utilisateur et ne prend même pas le focus : dans le cas où l'utilisateur serait en train d'effectuer une saisie dans un champ, la saisie continuera dans ce même champ durant tout le temps de l'affichage du message. Enfin, le message disparaît de lui-même.

```
public void afficherMessage(String texte)
{
   Log.d(TAG, "afficherMessage()");
   Toast.makeText(getApplicationContext(), texte,
Toast.LENGTH_LONG).show();
}
```

La méthode “afficherMessageAvecRetard(String texte, int temps)” permet, lui aussi, d’afficher un message dans le Toast d’android, mais avec un retard indiqué par l’argument “temps” (en millisecondes).
Pour pouvoir obtenir un délai sans bloquer toute l'application, la méthode créer un thread qui s'endort en fonction de “temps” avec la méthode Thead::sleep(long millis). Ensuite le thread appelle le thread principale avec la méthode “runOnUiThread()” en lui indiquant le nouveau message à afficher.

```
public void afficherMessageAvecRetard(String texte, int temps)
{
   Log.d(TAG, "afficherMessageAvecRetard()");
   new Thread()
   {
       @Override public void run()
       {
           try
           {
```

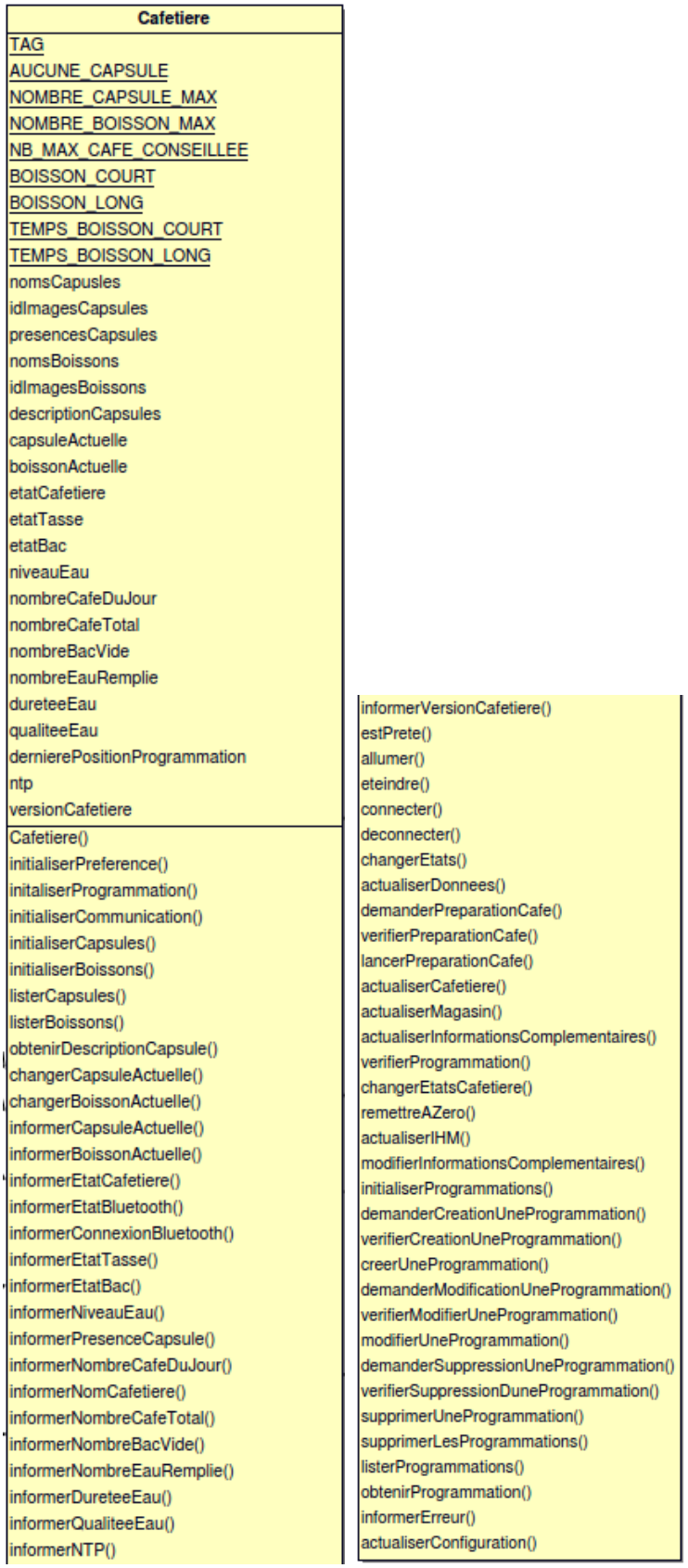
```
                sleep(temps);
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        Toast.makeText(getApplicationContext(),
texte, Toast.LENGTH_LONG).show();
                    }
                });
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }.start();
}
```

La méthode “envoyerNotification(Programmation programmation)” permet d’envoyer une notification en fonction des données de la “programmation” en argument.

```
public void envoyerNotification(Programmation programmation)
{
    Log.d(TAG, "envoyerNotification()");
    parametresNotification.setContentTitle("Votre programmation de
" + programmation.obtenirHeure() + " est en cour de
préparation.");

parametresNotification.setContentText(nomsCapsules[programmation.o
btenirCapsule()] + " - " +
nomsBoisson[programmation.obtenirBoisson()]);
    gestionnaireNotification.notify(0,
parametresNotification.build());
}
```

Cafetiere

La classe **`Cafetiere`** est la classe principale de l'application en assurant les liaisons entre les classes et l'IHM.



Attributs

Les attributs permettent de définir les états de la capsule comme : la capsule actuelle, la boisson actuelle ou même la présence de la tasse, l'état du bac ou le niveau d'eau

Méthodes

Les méthodes commençant par "informer", "lister" ou "obtenir" sont des assesseurs qui permettent d'obtenir les informations de la cafetière.

```
public int informer...()
{
   Log.d(TAG, "informer...()");
   return ...;
}
```

Les méthodes "allumer()", "eteindre()", "connecter()", "deconnecter()" permettent de modifier l'état du Bluetooth.

La méthode "changerEtats(String trame)" permet de mettre à jour les données relatives à la cafetière en fonction de la trame reçue.

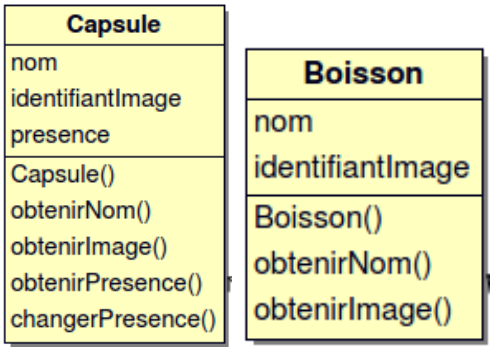
```
public void changerEtats(String trame)
{
   Log.d(TAG, "changerEtats()");
   trame = trame.replace(Protocole.DEBUT_TRAME, "");

   if(trame != null)
   {
       if (trame.startsWith(Protocole.TEST_ALIVE))
           actualiserDonnees();
       else if
(trame.startsWith(Protocole.INDICATEUR_PREPARATION_CAFE))
           verifierPreparationCafe(trame);
       else if
(trame.startsWith(Protocole.ACTUALISATION_CAFETIERE))
           actualiserCafetiere(trame);
       else if
(trame.startsWith(Protocole.ACTUALISATION_MAGASIN))
           actualiserMagasin(trame);
       else if
(trame.startsWith(Protocole.ACTUALISATION_COMPLEMENTAIRE) ||
trame.startsWith(Protocole.REINITIALISER))
           actualiserInformationsComplementaires(trame);
       else if
(trame.startsWith(Protocole.ACTUALISATION_PROGRAMMATION))
       {
           if (verifierProgrammation(trame))
               lancerPreparationCafe();
       }
       else if (trame.startsWith(Protocole.CREER_PROGRAMMATION))
           creerUneProgrammation(trame);
       else if
```

```
(trame.startsWith(Protocole.SUPPRIMER_PROGRAMMATION))
            verifierSuppressionDuneProgrammation(trame);
        else if
(trame.startsWith(Protocole.MODIFIER_PROGRAMMATION))
            verifierModifierUneProgrammation(trame);
        else if (trame.startsWith(Protocole.ERREUR_TRAME))
            informerErreur(trame);
        else if (trame.startsWith(Protocole.CONFIGURATION))
            actualiserConfiguration(trame);
    }
}
```

Les méthodes contenant le mot “Programmation” permettent de créer, modifier ou supprimer des programmations (voir classe Programmation).

Capsule et Boisson

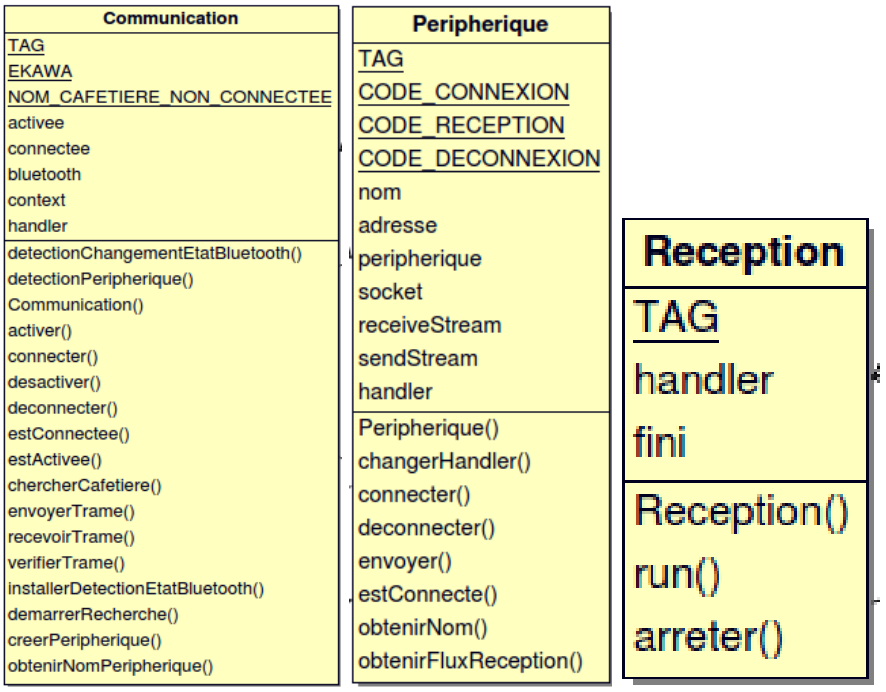
La classe **`Capsule`** est une structure de données relatives aux capsules présentes dans le magasin.
La classe **`Boisson`** est une structure de données relatives aux boissons effectuables par la machine.



Ces classes sont des structures, ce qui signifie qu'elles n'influent pas sur les autres classes. Elles permettent le stockage de données de manière organisée.

Communication, Peripherique et Receveur

Les classes **`Communication`**, **`Peripherique`** et **`Receveur`** s'occupent de la transmission sans-fil avec la machine.



Attributs

L'attribut “socket” est un objet de la classe BluetoothSocket.
Un socket est une interface logicielle avec les services du système d’exploitation, grâce à laquelle un développeur exploitera facilement et de manière uniforme les services d’un protocole réseau.
L'attribut “receiveStream” (objet de la classe InputStream) permet au socket de recevoir des informations.
L'attribut “sendStream” (objet de la classe OutputStream) permet au socket d’envoyer des informations.

```
socket =
peripherique.createRfcommSocketToServiceRecord(UUID.fromString("0
0001101-0000-1000-8000-00805F9B34FB"));
receiveStream = socket.getInputStream();
sendStream = socket.getOutputStream();
```

L'attribut “handler” ou “gestionnaire” en français, permet une liaison entre les threads des classes “Peripherique”, “Reception” et le thread principal (ici représenté par la classe Communication.
Lorsque le programme initialise le handler, il réécrit la méthode Handler::handleMessage(Message msg). Cela lui permet de pouvoir appeler une ou plusieurs méthodes différentes en fonction du code reçu.

```
private final Handler handler = new Handler()
{
   public void handleMessage(Message msg)
   {
       super.handleMessage(msg);
       switch (msg.what)
       {
           case Peripherique.CODE_CONNEXION:
               connectee = true;
               cafetiere.actualiserIHM();

peripherique.envoyer(Protocole.fabriquerTrameTestAlive());
               break;
           case Peripherique.CODE_RECEPTION:
               cafetiere.changerEtats(recevoirTrame((String)
msg.obj));
               break;
           case Peripherique.CODE_DECONNEXION:
               connectee = false;
               cafetiere.actualiserIHM();
               break;
       }
   }
};
```

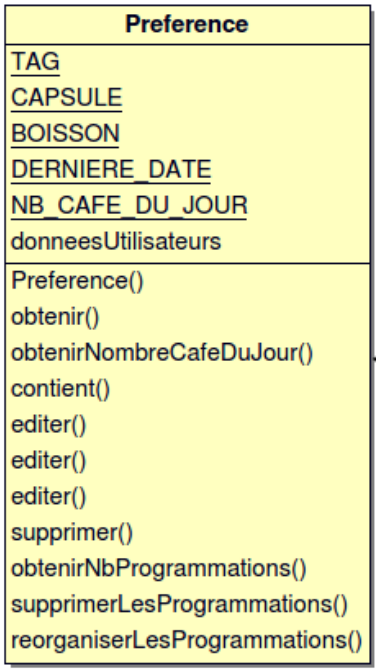
Pour envoyer un message, et donc appeler la méthode Handler::handleMessage(Message msg), le programme utilise la méthode Handler::sendMessage(Message msg).

Un objet de la classe Message est créé afin de contenir les informations (code du handler, trame reçue, …).
L'attribut publique Message::what permet de stocker le code du handler.
L'attribut publique Message::obj permet de stocker la trame reçue.

```
Message msg = Message.obtain();
msg.what = Peripherique.CODE_RECEPTION;
msg.obj = trame;
handler.sendMessage(msg);
```

Preference

La classe **`Preference`** est la classe qui gère les préférences utilisateurs.



Attributs

L'attribut “dooneesUtilisateurs” est un objet de la classe SharedPreferences.
Les “SharedPreferences” ou “préférences partagées” en français, permettent aux activités et aux applications de conserver les préférences, sous la forme de paires clé-valeur similaires à une carte qui persisteront même lorsque l'utilisateur ferme l'application.
“dooneesUtilisateurs” est en mode privé (défini par l’argument “ihm.MODE_PRIVATE” dans la méthode AppCompatActivity::getPreferences()), ce qui signifie que les préférences de l’application seront disponibles uniquement par l’application Ekawa.

```
donneesUtilisateurs = ihm.getPreferences(ihm.MODE_PRIVATE);
```

Méthodes

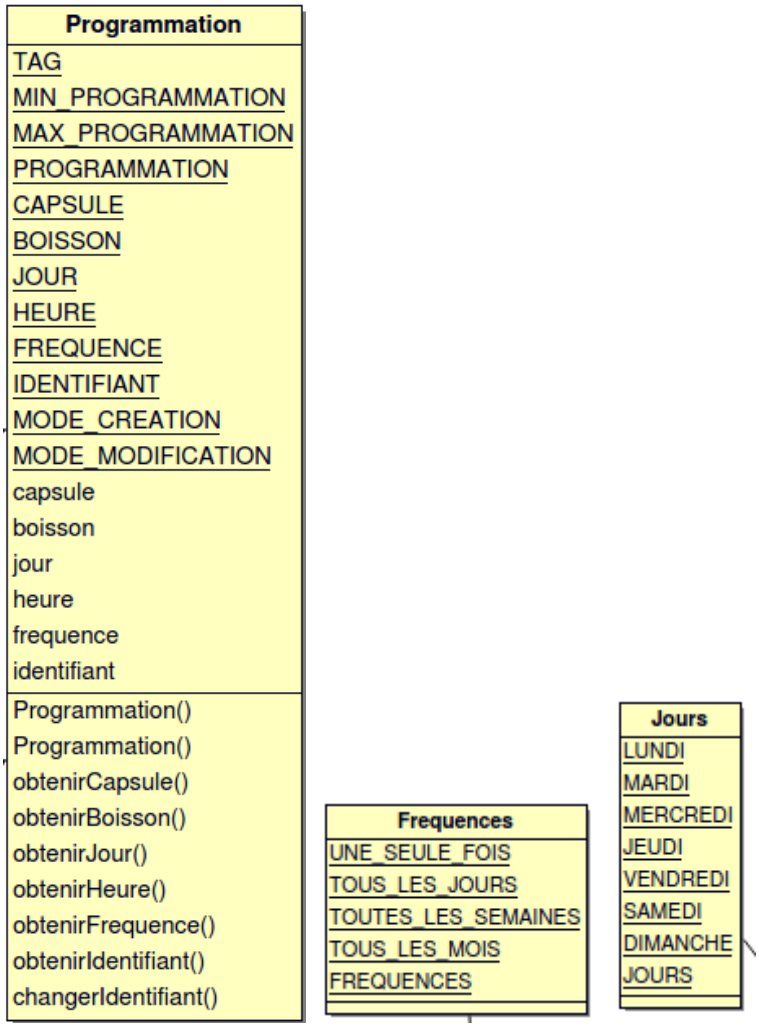
Les méthodes “editer()” permettent d’ajouter ou de modifier des données dans les préférences de l’application. Il y a plusieurs méthode car SharedPreferences propose plusieurs méthodes d’ajout/modification de données en fonction du type (ex : SharedPreferences::Editor::putString() ; SharedPreferences::Editor::putInt() ; …).

```
public void editer(String nomPreference, ... valeur)
{
   donneesUtilisateurs.edit().put...(nomPreference,
valeur).apply();
}
```

La classe Preference peut donc prendre en charge les types suivants : String, int, boolean

Programmation

La classe **`Programmation`** est la classe qui gère les programmations de cafés.



Ces classes sont des structures, ce qui signifie qu'elles n'influent pas sur les autres classes. Elles permettent le stockage de données de manière organisée.

Attributs

La classe Programmation possède 2 classes : "Frequences" et "Jours".
La classe Frequences permet de définir la fréquence : le nom de chaque fréquence et son identifiant.

```
public static class Frequences
{
   public final static int UNE_SEULE_FOIS = 0;          //!< La
valeur associée à la fréquence : Une seule fois
   public final static int TOUS_LES_JOURS = 1;          //!< La
valeur associée à la fréquence : Tous les jours
   public final static int TOUTES_LES_SEMAINES = 2;     //!< La
valeur associée à la fréquence : Tous les semaines
   public final static int TOUS_LES_MOIS = 3;           //!< La
valeur associée à la fréquence : Tous les mois

   public final static String[] FREQUENCES =
```

```
    {
        "Une seule fois",
        "Tous les jours",
        "Toutes les semaines",
        "Tous les mois"
    };  //!< Le nom de chaques fréquences
}
```

La classe Jours permet de définir le jour de la semaine : le nom de chaque jour et son identifiant.

```
public static class Jours
{
    public final static int LUNDI = 0;      //!< La valeur
associée au jour : Lundi
    public final static int MARDI = 1;      //!< La valeur
associée au jour : Mardi
    public final static int MERCREDI = 2;   //!< La valeur
associée au jour : Mercredi
    public final static int JEUDI = 3;      //!< La valeur
associée au jour : Jeudi
    public final static int VENDREDI = 4;   //!< La valeur
associée au jour : Vendredi
    public final static int SAMEDI = 5;     //!< La valeur
associée au jour : Samedi
    public final static int DIMANCHE = 6;   //!< La valeur
associée au jour : Dimanche

    public final static String[] JOURS =
    {
        "Lundi",
        "Mardi",
        "Mercredi",
        "Jeudi",
        "Vendredi",
        "Samedi",
        "Dimanche"
    };  //!< Le nom de chaques jour
}
```

Protocole

La classe **`Protocole`** est la classe qui permet le respect du protocole de communication.

La classe “Protocole” est une classe de service.
Une classe de service est une classe désignée comme constante, c'est-à-dire que l’intégralité de ces attributs et de ces méthodes doivent être constantes. Une classe de service ne peut être définie dans une autre classe.

Attributs

Les attributs constants commençant par “EMPLACEMENT” permettent au programme de savoir à quel emplacement se situe chaque donnée dans toutes les trames.
Pour pouvoir obtenir un tableau avec une donnée dans chaque case, le programme utilise la méthode String::split(String).

```
trame.split(";");
```

Les autres attributs constants sont des valeurs possibles pour chaque donnée (voir documentation).

```
public static final String CAPSULE_ABSENTE = "0";
//!< La capsule est absente
public static final String CAPSULE_PRESENTE = "1";
//!< La capsule est présente
```

```
if(valeurs[i].equals(CAPSULE_ABSENTE))
   etatsCapsules[i] = false;
else if(valeurs[i].equals(CAPSULE_PRESENTE))
   etatsCapsules[i] = true;
```

Méthodes

Les méthodes commençant par “fabriquerTrame” permettent au programme de construire une trame à envoyer.
Un trame est composée de 3 partie :

- les délimiteurs (début : **$ekawa** / séparateur : **;** / fin : **\r\n**)
- les identifiants (S1, S2, C, P, ...)
- les arguments (non obligatoire) passer en arguments de la méthode

```
public static String fabriquerTramePreparationCafe(int
boissonActuelle, int capsuleActuelle)
{
   String trame = Protocole.DEBUT_TRAME;
   trame += Protocole.INDICATEUR_PREPARATION_CAFE + ";";
   trame += boissonActuelle + ";";
   trame += capsuleActuelle;
   trame += Protocole.FIN_TRAME;

   Log.d(TAG, "Fabriquer trame = " + trame);

   return trame;
}
```

Voir 3.5. Protocole de communication.

3.4. Outils de développement

Les outils logiciels :

Désignation	Caractéristiques
Atelier de génie logiciel	**Bouml** v7.11
Logiciel de gestion de version	**Subversion** (RiouxSVN)
Environnement de développement	**Android Studio** 4.1.3

3.4.1. UML

UML

Le Langage de Modélisation Unifié, de l'anglais Unified Modeling Language (UML), est un langage de modélisation graphique à base de pictogrammes conçu comme une méthode normalisée de visualisation dans les domaines du développement logiciel et en conception orientée objet.

BoUml

BOUML est une boîte à outils UML 2 gratuite comprenant un modeleur vous permettant de spécifier et de générer du code en C ++, Java, Idl, Php, Python et MySQL.

Depuis la version 7.0, BOUML est à nouveau un logiciel libre.

BOUML fonctionne sous Windows, Linux et MacOS X.

BOUML est très rapide et ne nécessite pas beaucoup de mémoire pour gérer plusieurs milliers de classes, voir benchmark.

3.4.2. Subversion

Subversion

Subversion (en abrégé svn) est un logiciel de gestion de versions, distribué sous licence Apache. Il a été conçu pour remplacer CVS. Ses auteurs s'appuient volontairement sur les mêmes concepts (notamment sur le principe du dépôt centralisé et unique) et considèrent que le modèle de CVS est bon, seule son implémentation est perfectible.

Subversion fonctionne donc sur le mode client-serveur, avec :

un serveur informatique centralisé et unique où se situent :
les fichiers constituant la référence (le « dépôt » ou « référentiel », ou « repository » en anglais),
un logiciel serveur Subversion tournant en « tâche de fond » ;
des postes clients sur lesquels se trouvent :
les fichiers recopiés depuis le serveur, éventuellement modifiés localement depuis,
un logiciel client, sous forme d'exécutable standalone (ex. : SmartSVN) ou de plug-in (ex. : TortoiseSVN, Eclipse Subversive) permettant la synchronisation, manuelle et/ou automatisée, entre chaque client et le serveur de référence
Le projet a été lancé en février 2000 par CollabNet, avec l'embauche par Jim Blandy de Karl Fogel, qui travaillait déjà sur un nouveau logiciel gestionnaire de version.

Le 14 février 2010, SVN est devenu officiellement un projet de la fondation Apache, prenant le nom d'Apache Subversion.

RiouxSVN

RiouxSVN est un service d'hébergement Subversion totalement gratuit et privé.

3.4.3. Android

Android

Android est un système d'exploitation mobile basé sur le noyau Linux et développé actuellement par Google.

Android est aussi défini comme étant une pile de logiciels organisée en cinq couches distinctes :
• un système d’exploitation (comprenant un noyau Linux avec les pilotes) ;
• des bibliothèques logicielles telles que WebKit, OpenGL, SQLite, ... ;
• un environnement d’exécution et des bibliothèques permettant d’exécuter des programmes prévus pour la plate-forme Java ;
• un kit de développement d’applications (SDK) ;
• des d’applications standards (un environnement de bureau, carnet d’adresses, application téléphone, ...).

Remarque : À chaque version du système est associé un niveau d’API (Exemple : Android 4.4 KITKAT → API 19).

Android Studio

Android Studio est un environnement de développement intégré (EDI) pour développer des applications Android. Il est basé sur IntelliJ.

Android Studio permet principalement d'éditer les fichiers Java (ou Kotlin) et les fichiers de configuration d'une application Android.

Environnement de Développement Intégré (EDI)

Un environnement de développement intégré (EDI ou IDE pour Integrated Development Environment) est un ensemble d'outils intégrés dans un même logiciel et qui permettent de développer des programmes.

Android Studio propose de nombreux outils pour gérer le développement d'applications et permet de visualiser la mise en page des écrans sur des écrans de résolutions variées simultanément.
Remarque : Android Studio n'est pourtant pas un outil indispensable pour développer des applications Android. Seul le kit de développement Android est nécessaire.

Kit de développement (SDK)

Un kit de développement logiciel (SDK = Software Development Kit ou devkit) est un ensemble d'outils logiciels destinés aux développeurs, facilitant le développement d'un logiciel sur une plateforme donnée (par exemple, iOS, Android, GNU/Linux, OS X, Microsoft Windows).

L'Android SDK est un ensemble complet d'outils de développement (pour Linux, MAC OS ou Windows). Il inclut un débogueur, des bibliothèques logicielles, un émulateur basé sur QEMU, de la documentation, des exemples de code et même des tutoriaux.

ADB

ADB (Android Debug Bridge) est un outil inclus dans le package Android SDK. Il se compose d'un programme client et d'un programme serveur communiquant entre eux et qui permet :

- la copie de fichier ;
- l'accès à la console Android ;
- la sauvegarde de la mémoire ROM ;
- l'installation de logiciel.

API

C'est une interface de programmation d'application (API = Application Programming Interface) qui est un ensemble normalisé de classes et/ou de fonctions utilisables par un logiciel. L'API (et donc ses services) est offerte par une bibliothèque logicielle.

Une bibliothèque logicielle est une collection de fonctions prêtes à être utilisées par des programmes.

Framework

Littéralement un “cadre de développement". Le framework Android fournit une API ET une manière (un “cadre”) de l’utiliser (notamment la notion d’Activité propre à Android).

Le framework Android permet :

- de définir des interfaces utilisateurs graphiques (en XML)
- de créer des activités (composant de base d’une application Activity)
- de fournir et partager des données (composant Content Providers)
- d’interagir avec le système (composants Intents, Broadcast Receivers, Services, ...)

Android permet aussi d’accéder aux fonctionnalités avancées du smartphone / tablette :

- Stockage (fichiers, base de données, ...)
- Communication Réseau (Wifi, Bluetooth, 3G/4G, ...)
- Multimédia (audio, photo, caméra)
- GPS
- Téléphonie (appels, SMS)

Java

En résumé, la plate-forme Java comprend :

- un compilateur Java (javac), fourni par le JDK (Java Development Kit), convertit les codes source Java en bytecode Java (un langage intermédiaire).
- une JVM (java), fournie par le JRE (Java Runtime Environment), comprend un compilateur JIT (Just In Time) qui convertit à la volée le bytecode intermédiaire en un code natif pour la machine.

Gradle

La fabrication d'une application Android nécessite maintenant Gradle.
L’outil gradle est le moteur de production qui permet de construire (build) le projet Android.
Son usage est "transparent" dans l’IDE Android Studio.

3.5. Protocole de communication

3.5.1. Présentation

Contenu : caractères ASCII standard

Délimiteurs		
Début	Champ	Fin
$ekawa	**;**	**\r\n**

Type de trames	
Indicateur	Informations
A	Test de communication
C	Lancer la préparation d'un café
E	Trame contenant un code d'erreur
P	Programmer la préparation d'un café
M	Modifier programmation de la préparation d'un café
S	Supprimer une programmation de café
R	Réinitialiser les paramètres machine
?	Obtenir des informations sur la configuration
S1	Actualiser les données de la cafetière (état cafetière, tasse, bac, niveau eau)
S2	Actualiser les données du magasin (présence capsule)
S3	Actualiser les données complémentaires (cafés totaux, nombre de bacs vidés, nombre de remplissage)
S4	Actualiser les données de programmation

Trame de test de communication **'A'**	
Indicateur	Informations
t	Char : lettre de test (A)
r	Char : reponse (A)

Lancer la préparation d'un café **'C'**	
Indicateur	Informations
b	Booléen : type de boisson (0 = court / 1 = long)
c	Entier : type de café (0 à 7 en fonction de la colonne du magasin)
r	*Booléen : réponse (0 = impossible / 1 = possible)*

Trame d'erreur **'E'**	
Indicateur	Informations

r	*Booléen : réponse (0 = ok / x = code d'erreur)* *1 ERREUR_ENTETE* *2 ERREUR_NB_PARAMETRES* *3 ERREUR_TRAME_INCONNUE* *4 ERREUR_TYPE_CAFE* *5 ERREUR_LONGUEUR_CAFE*

Programmer la préparation d'un café (en option pour les EC) 'P'	
Indicateur	Informations
c	Entier : type de café (0 à 7 en fonction de la colonne du magasin)
b	Booléen : type de boisson (0 = court / 1 = long)
j	Entier : jour de la semaine pour la programmation (0 = Lundi / 1 = Mardi / 2 = Mercredi / 3 = Jeudi / 4 = Vendredi / 5 = Samedi / 6 = Dimanche)
h	String : heure de la programmation (exemple : "16h15") ("h" étant le séparateur)
f	Entier : fréquence de programmation (0 = Une seule fois / 1 = Tous les jours / 2 = Toutes les semaines / 3 = Tous les mois)
r	*Booléen : réponse (0 = impossible / x = identifiant de la programmation à partir de 1)*
Remarques : le nombre de programmations est limité à 4. Le simulateur nécessite une connexion WiFi (serveur NTP obligatoire) pour réaliser une programmation de café. Le simulateur ne gère pas la fréquence 3 (Tous les mois).	

Modifier programmation de la préparation d'un café (en option pour les EC) 'M'	
Indicateur	Informations
x	Entier : identifiant de la programmation
c	Entier : type de café (0 à 7 en fonction de la colonne du magasin)
b	Booléen : type de boisson (0 = court / 1 = long)
j	Entier : jour de la semaine pour la programmation (0 = Lundi / 1 = Mardi / 2 = Mercredi / 3 = Jeudi / 4 = Vendredi / 5 = Samedi / 6 = Dimanche)
h	String : heure de la programmation (exemple : "16h15") ("h" étant le séparateur)
f	Entier : fréquence de programmation (0 = Une seule fois / 1 = Tous les jours / 2 = Toutes les semaines / 3 = Tous les mois)
r	*Booléen : réponse (0 = impossible / x = l'identifiant de la programmation modifiée)*

Supprimer la programmation d'un café `'S'`	
Indicateur	Informations
x	Entier : identifiant de la programmation (toujours > 0)
r	*Booléen : réponse (0 = impossible / x = l'identifiant de la programmation supprimée)*

Réinitialiser les paramètres machine `'R'`	
Indicateur	Informations
c	Booléen : nombre total de cafés effectués par la machine (0 = non / 1 = oui)
b	Booléen : nombre total de bacs vidés (0 = non / 1 = oui)
e	Booléen : nombre total de remplissage d'eau (0 = non / 1 = oui)
d	Integer : dureté de l'eau (-1 = ne pas réinitialiser)
[q]	Integer : qualité de l'eau (-1 = ne pas réinitialiser) **[en option]**
[p]	Booléen : l'ensemble des programmations (0 = non / 1 = oui) **[en option]**
[s]	Booléen : force les états simulés (0 = non / 1 = oui) **[en option]** **- un remplissage de l'eau** **- un vidage du bac à capsules** **- un remplissage du magasin**
r	*Booléen : réponse (0 = impossible / 1 = possible)* *Remarque : la trame `R` génère l'envoie des trames d'états S1, S2 et S3*

Obtenir des informations sur la configuration `'?'`	
Indicateur	Informations
n	*Booléen : NTP actif (0 = oui / 1 = non)*
r	*Integer : nombre de colonnes du magasin*
c	*Integer : nombre de capsules par rangée*
b	*Integer : taille en nombre de capsules du bac de récupération*
e	*Integer : niveau d'eau max en nombre de cafés courts*
p	*Integer : nombre de programmation max*
v	*String : le numéro de version*

Données de la cafetière `'S1'`	
Indicateur	Informations
c	Booléen : état de la cafetière (0 = non disponible ou utilisé / 1 = disponible)
t	Booléen : présence de la tasse (0 = non présente / 1 = présente)
b	Booléen : trop plein du bac (0 = plein / 1 = non plein)
e	Entier : pourcentage de niveau d'eau (0 à 100)

Données du magasin **'S2'**	
Indicateur	Informations
c0	Booléen : présence de la capsule de la colonne 0 (0 = non présente / 1 = présente)
c1	Booléen : présence de la capsule de la colonne 1 (0 = non présente / 1 = présente)
c2	Booléen : présence de la capsule de la colonne 2 (0 = non présente / 1 = présente)
c3	Booléen : présence de la capsule de la colonne 3 (0 = non présente / 1 = présente)
c4	Booléen : présence de la capsule de la colonne 4 (0 = non présente / 1 = présente)
c5	Booléen : présence de la capsule de la colonne 5 (0 = non présente / 1 = présente)
c6	Booléen : présence de la capsule de la colonne 6 (0 = non présente / 1 = présente)
c7	Booléen : présence de la capsule de la colonne 7 (0 = non présente / 1 = présente)

Données complémentaires **'S3'**	
Indicateur	Informations
c	Integer : nombre total de cafés effectués par la machine
b	Integer: nombre total de bacs vidés
e	Integer : nombre total de remplissage d'eau
d	Integer : dureté de l'eau
q	Integer : qualité de l'eau (en option)

Données programmation **'S4'**	
Indicateur	Informations
x	Integer : l'identifiant de la programmation
e	Integer: état (0 = erreur / 1 = en cours)

3.5.2. Exemples

Client : Application Android
Serveur : ESP32

Test de communication :

Client → Serveur : **$ekawa;A\r\n**

Serveur → Client : **$ekawa;A\r\n**

Lancer la préparation d'un café :

Client → Serveur : **$ekawa;C;b;c\r\n**

Serveur → Client : **$ekawa;C;r\r\n**

Après la préparation d'un café ou si réponse (r) est faux (0) :

Serveur → Client : **$ekawa;S1;c;t;b;e\r\n**

Serveur → Client : **$ekawa;S2;c0;c1;c2;c3;c4;c5;c6;c7\r\n**

Trame d'erreur 'E' :

Serveur → Client : **$ekawa;r\r\n**

Programmer la préparation d'un café :

Client → Serveur : **$ekawa;P;c;b;j;h;f\r\n**

Serveur → Client : **$ekawa;P;r\r\n**

Modifier la programmation d'un café :

Client → Serveur : **$ekawa;M;x;c;b;j;h;f\r\n**

Serveur → Client : **$ekawa;M;r\r\n**

Supprimer la programmation d'un café :

Client → Serveur : **$ekawa;S;x\r\n**

Serveur → Client : **$ekawa;S;r\r\n**

Réinitialiser les paramètres machines :

Client → Serveur : **$ekawa;R;c;b;e;d;q;p;s\r\n**

Serveur → Client : **$ekawa;R;r\r\n**

Remarque : les nombres totaux de cafés effectués par la machine, nombres totaux de bacs vidés et nombres totaux de remplissage d'eau peuvent être remis individuellement à 0

Obtenir des informations sur la configuration :

Client → Serveur : **$ekawa;?\r\n**

Serveur → Client : **$ekawa;?;n;r;c;b;e;p;v\r\n**

Actualiser les données de la cafetière (état cafetière, tasse, bac, niveau eau)

Client → Serveur : **$ekawa;S1\r\n**

Serveur → Client : **$ekawa;S1;c;t;b;e\r\n**

Actualiser les données du magasin (présence capsule)

Client → Serveur : **$ekawa;S2\r\n**

Serveur → Client : **$ekawa;S2;c0;c1;c2;c3;c4;c5;c6;c7\r\n**

Actualiser les données complémentaires (cafés totals, nombre bac vidé, nombre d'eau remplie)

Client → Serveur : **$ekawa;S3\r\n**

Serveur → Client : **$ekawa;S3;c;b;e;d;q\r\n**

Actualiser les données de programmation :

Client → Serveur : **$ekawa;S4\r\n**

Serveur → Client : **$ekawa;S4;x;e\r\n**

3.5.3. Tests simulateur

Client : Application Android <--
Serveur : ESP32 -->

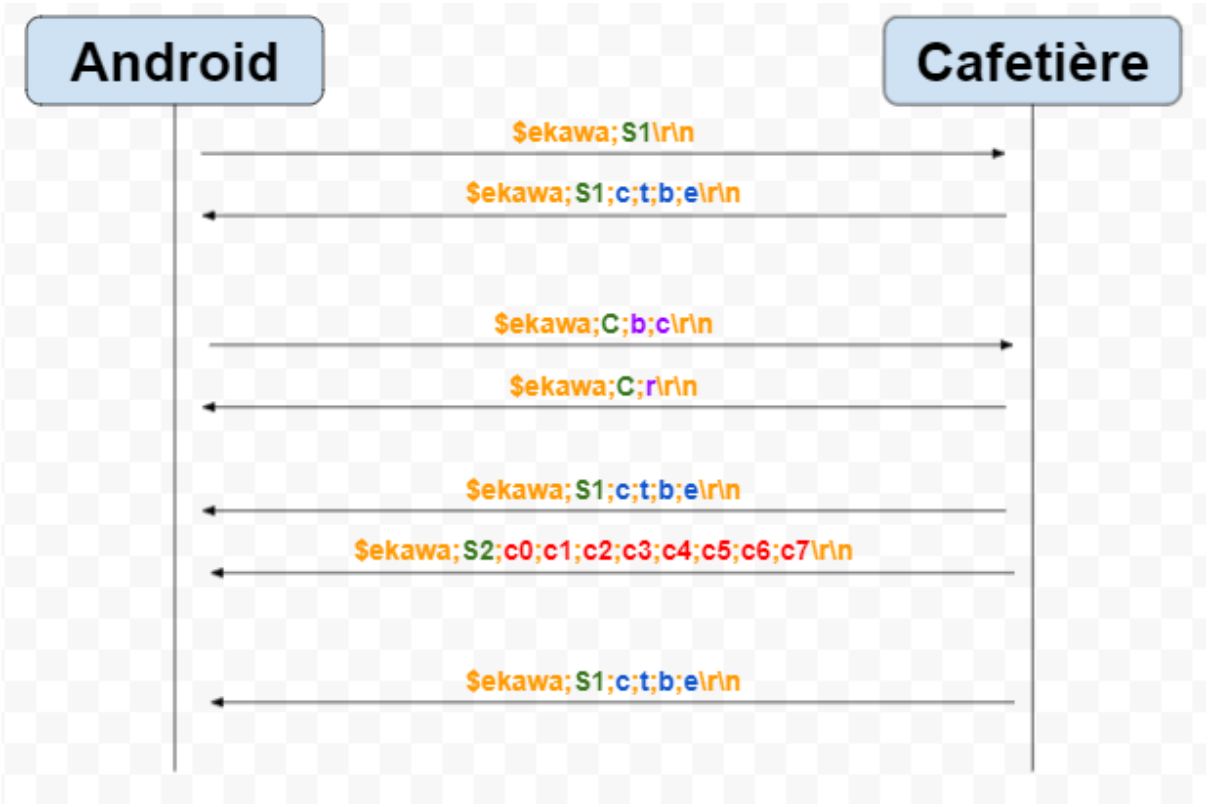
```
Simulateur Bluetooth ekawa
```

```
.....
192.168.52.25
<WiFi> avec NTP !
-> $ekawa;S1;1;1;1;100
-> $ekawa;S2;1;1;1;1;1;1;1;1
-> $ekawa;S3;8;2;0;0;0
<Init> Timer TIMER_SCRUTATION
<Init> idEnregistrement : 1

<- $ekawa;P;0;0;0;17h23;1
<Programmation> enregistrement id : 1
-> $ekawa;P;1p

<- $ekawa;P;0;0;0;17h24;0
<Programmation> enregistrement id : 2
-> $ekawa;P;2

<- $ekawa;P;0;0;0;17h25;2
<Programmation> enregistrement id : 3
-> $ekawa;P;3

<- $ekawa;C;0;1
<Cafe> encours !
<Machine> Bac capsules : 1
<Machine> Niveau eau : 9 capsules restantes
-> $ekawa;C;1
-> $ekawa;S1;0;1;1;90
<Cafe> terminé !
-> $ekawa;S1;1;1;1;90
-> $ekawa;S2;1;1;1;1;1;1;1;1

<Execution> heure NTP           : 17:23:00
<Execution> programmation id    : 1
<Cafe> encours !
-> $ekawa;S1;0;1;1;90
<Machine> Bac capsules : 2
<Machine> Niveau eau : 8 capsules restantes
-> $ekawa;S4;1;1
<Cafe> terminé !
-> $ekawa;S1;1;1;1;80
-> $ekawa;S2;1;1;1;1;1;1;1;1

<Execution> heure NTP           : 17:24:00
<Execution> programmation id    : 2
```

```
<Cafe> encours !
<Machine> Bac capsules : 3
<Machine> Niveau eau : 7 capsules restantes
-> $ekawa;S4;2;1
-> $ekawa;S1;0;1;1;70
<Cafe> terminé !
-> $ekawa;S1;1;1;1;70
-> $ekawa;S2;1;1;1;1;1;1;1;1

<- $ekawa;M;1;0;0;0;17h26;0
<Reprogrammation> enregistrement id : 1
-> $ekawa;M;1

<Execution> heure NTP           : 17:25:00
<Execution> programmation id    : 3
<Cafe> encours !
<Machine> Bac capsules : 4
<Machine> Niveau eau : 6 capsules restantes
-> $ekawa;S4;3;1
-> $ekawa;S1;0;1;1;60
<Cafe> terminé !
-> $ekawa;S1;1;1;1;60
-> $ekawa;S2;1;1;1;1;1;1;1;1

<Execution> heure NTP           : 17:26:00
<Execution> programmation id    : 1
<Cafe> encours !
<Machine> Bac capsules : 5
<Machine> Niveau eau : 5 capsules restantes
-> $ekawa;S4;1;1
-> $ekawa;S1;0;1;0;50
<Cafe> terminé !
-> $ekawa;S1;1;1;0;50
-> $ekawa;S2;0;1;1;1;1;1;1;1

<Simulation> Vidage du bac !
-> $ekawa;S1;1;1;1;50

<- $ekawa;M;1;0;0;0;17h30;0
<Erreur> Reprogrammation impossible avec id introuvable !
-> $ekawa;M;0

<- $ekawa;M;3;0;0;0;17h30;1
<Reprogrammation> enregistrement id : 3
```

```
-> $ekawa;M;3

<Execution> heure NTP           : 17:30:27
<Execution> programmation id    : 3
<Erreur> Plus de ce type de capsule !
-> $ekawa;S4;3;0
-> $ekawa;S2;0;1;1;1;1;1;1;1
<Execution> heure NTP           : 17:30:27
<Execution> programmation id    : 3
<Erreur> Plus de ce type de capsule !
-> $ekawa;S4;3;0
-> $ekawa;S2;0;1;1;1;1;1;1;1
...etc

rst:0x1 (POWERON_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
Simulateur Bluetooth ekawa
.....
192.168.52.25
<WiFi> avec NTP !
-> $ekawa;S1;1;1;1;100
-> $ekawa;S2;1;1;1;1;1;1;1;1
-> $ekawa;S3;13;3;0;0;0
<Init> Timer TIMER_SCRUTATION
<Init> idEnregistrement : 4

<- $ekawa;C;2;2
<Erreur> Longueur inconnue !
-> $ekawa;E;5
-> $ekawa;C;0
-> $ekawa;S1;1;1;1;100
-> $ekawa;S2;1;1;1;1;1;1;1;1

<- $ekawa;c;2;2
-> $ekawa;E;3
```

3.5.4. Visualisation



3.6. Informations

Auteur

Jean-Luc Lecomte

Date

2021

Version

0.3

Voir également

https://svn.riouxsvn.com/ekawa-2021/