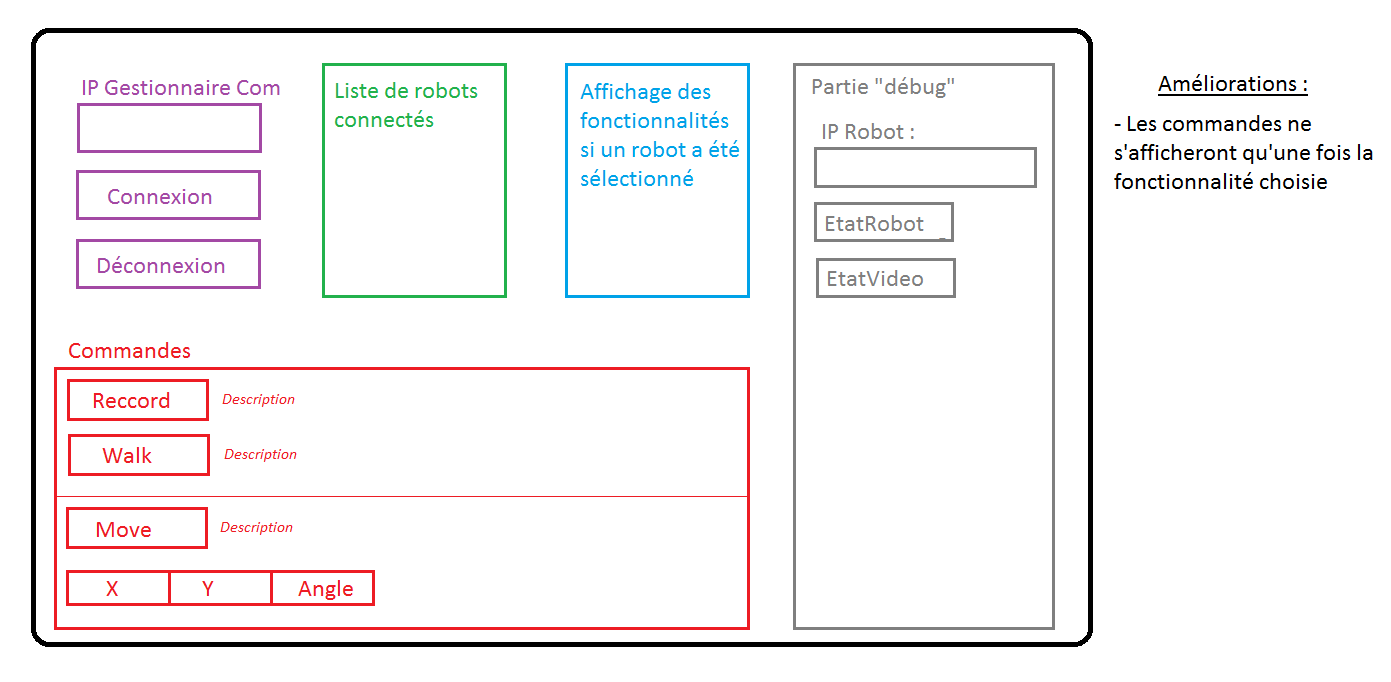
# Conception

## « Prototype » de l’application

La conception « globale » de notre application ayant été établie au semestre précédent, nous nous avons ensuite réfléchi à la conception de chacune des sous-parties de notre application, à avoir la tablette, la Kinect, le gestionnaire de communication, Nao et Robotino.

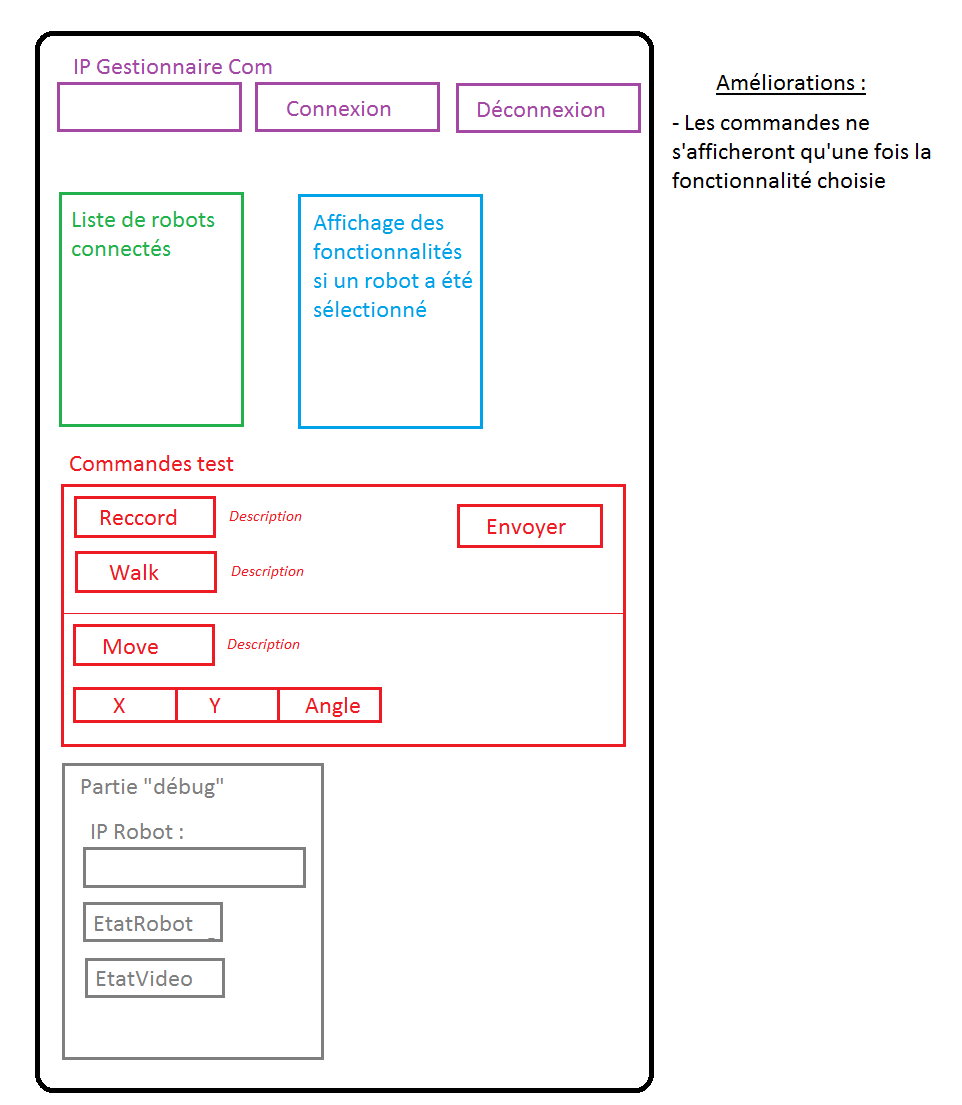
En ce qui concerne la partie tablette, j’ai dans un premier temps chercher à déterminer quels serait les différents composants de l’application, ainsi qu’à la façon de les placer. L’application peut être découpée en 5 parties :

* Une partie pour la connexion au gestionnaire de communication,
* Une liste permettant l’affichage des robots connectés,
* Une liste pour l’affichage des fonctionnalités du robot sélectionné,
* Une partie « commande » pour mettre l’envoi d’ordres aux robots,
* Une partie « débug » permettant l’affichage d’informations liées à un robot sélectionné.

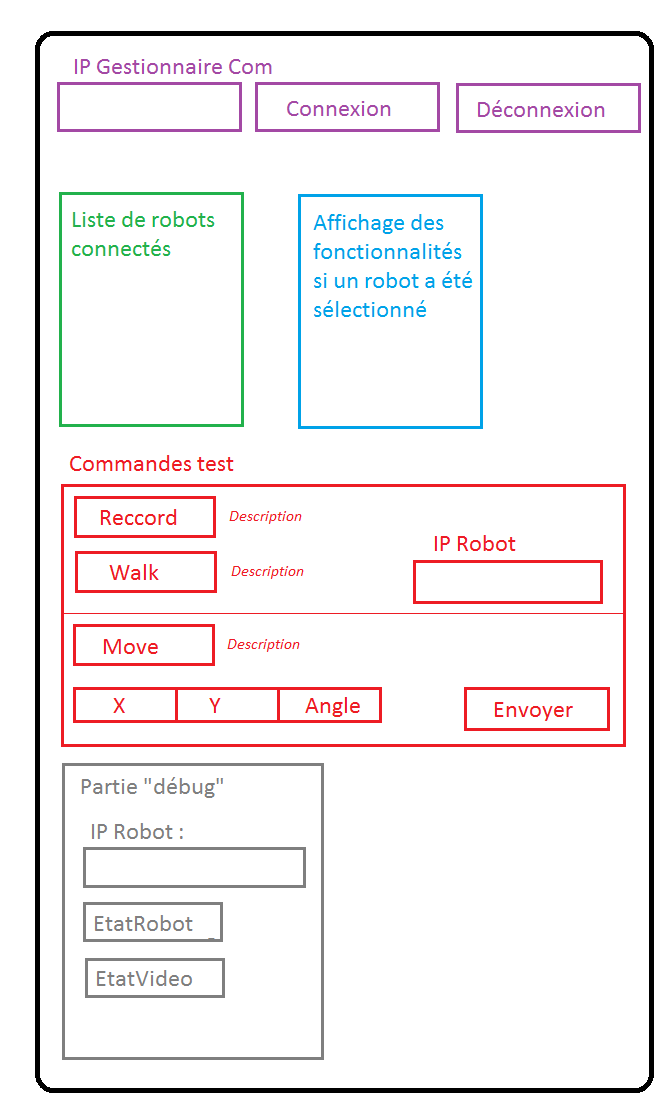


Dans un premier temps, l’application permettra à l’utilisateur de se connecter au gestionnaire de communication, et de sélectionner un robot dans une liste de robots connectés. La liste des fonctionnalités se mettra à jour, afin d’avoir une vision des possibilités offertes par le robot précédemment sélectionné. Cependant, dans une première approche, cette liste ne permettra pas de choisir une de ces fonctionnalités pour ensuite transmettre l’ordre au robot. La commande des robots se fera grâce à l’interface de commande. Par la suite, le choix d’une fonctionnalité entrainera l’ouverture d’une « fenêtre » permettant de renseigner les paramètres liés à la fonctionnalité choisie.

Suite à la présentation de ce modèle pour l’application tablette au reste de l’équipe, il est apparu que la partie « commande » devrait être renommée en « commandes test » puisqu’elle permet le débogage de l’application, et qu’il manquait un bouton envoyé. De plus, la présentation de l’application en format portrait nous a paru plus judicieuse, étant donné qu’elle permet une séparation plus claire des différentes parties.



Dans l’optique de pouvoir tester l’application le plus tôt possible, nous avons choisi d’ajouter un champ « IP Robot » dans le bloc « Commandes test ».



## Diagramme de classe de l’application

Une application ayant déjà été développée lors du semestre précédent, nous sommes repartis de celle-ci :



* Classe Client.java

*Méthode connexion(String url)*

Cette méthode instancie un nouveau socket. Elle tente ensuite une connexion au serveur qui doit être à l'adresse passée en paramètre et sur la socket 6020. Si la connexion réussie, on définit alors les canaux de communication vers et venant du serveur. Pour finir, on envoie au serveur l'id de la tablette venant de se connecter à lui.

*Méthode traitementReception(String sMessage)*

Cette méthode traite les informations en provenance du serveur. Elle est appelée dans la méthode run().

*Méthode writeMessage(String sLeMessage)*

Cette dernière va écrire vers le serveur le message passé en paramètre.

*Méthode run()*

Cette méthode représente l'exécution associée au thread client. Elle va faire appel à connexion() puis attendre les message provenant du serveur.

* Classe Flux.java

*Méthode lectureMessage(DataInputStream in)*

Méthode statique permettant de lire les informations venant du réseau et de les stocker dans une chaîne de caractères qui est ensuite retournée.

*Méthode ecritureMessage(DataOutputStream out, String s)*

Méthode statique permettant d'écrire sur le réseau la chaîne de caractères passée en paramètre.

* Classe MainActivity

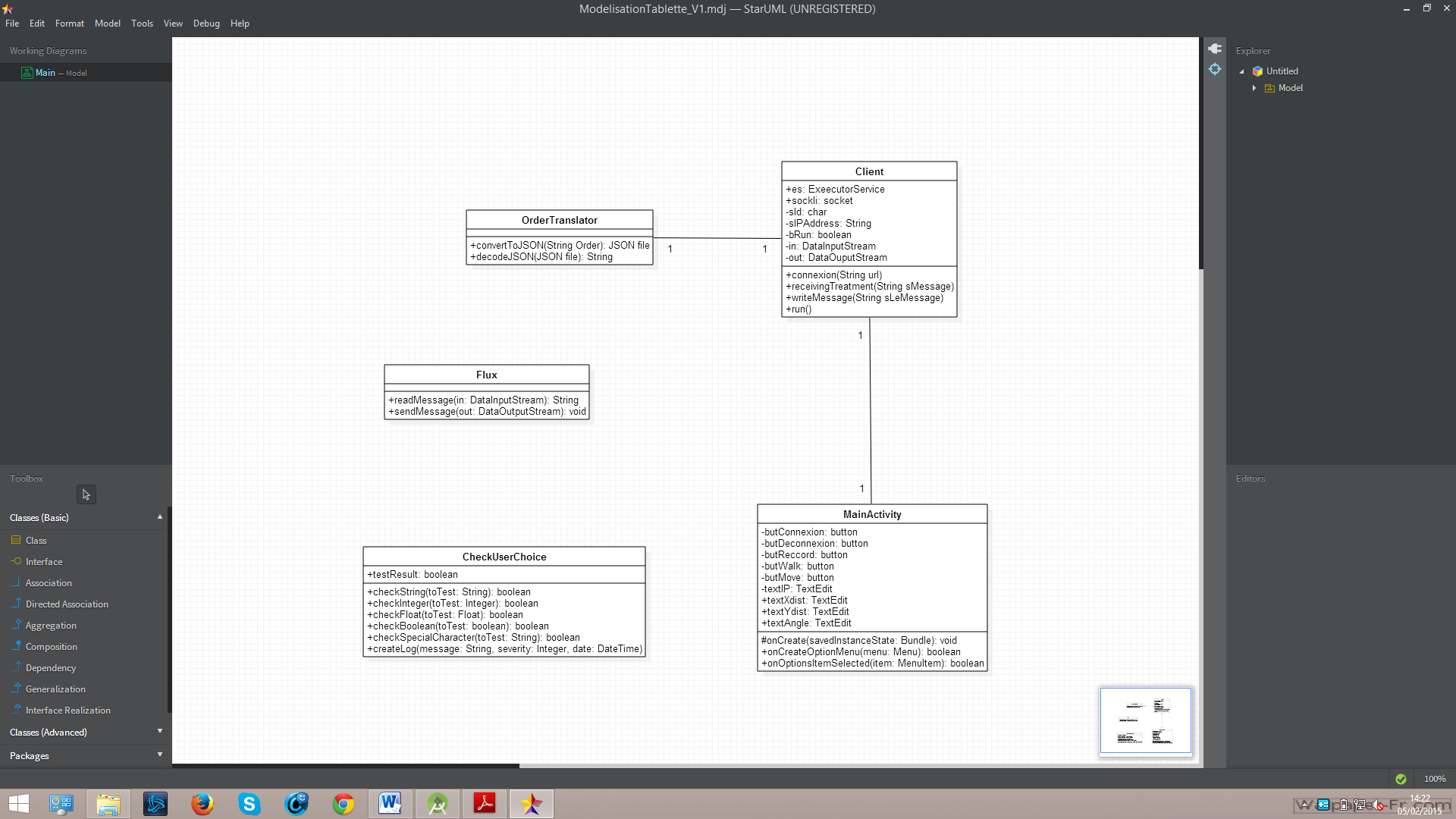
*Méthode onCreate(Bundle savedInstanceState)*

Cette méthode permet au lancement de l'application d'effectuer des tâches particulières. Elle sert tout d'abord à récupérer l'instance de l'interface graphique. Puis elle fait correspondre les différents boutons et autres champs apparaissant à l'écran avec leurs « représentations logicielles ». On met ensuite en place les différents listener sur les clics sur les boutons, afin qu'un appui sur un bouton déclenche une action précise.

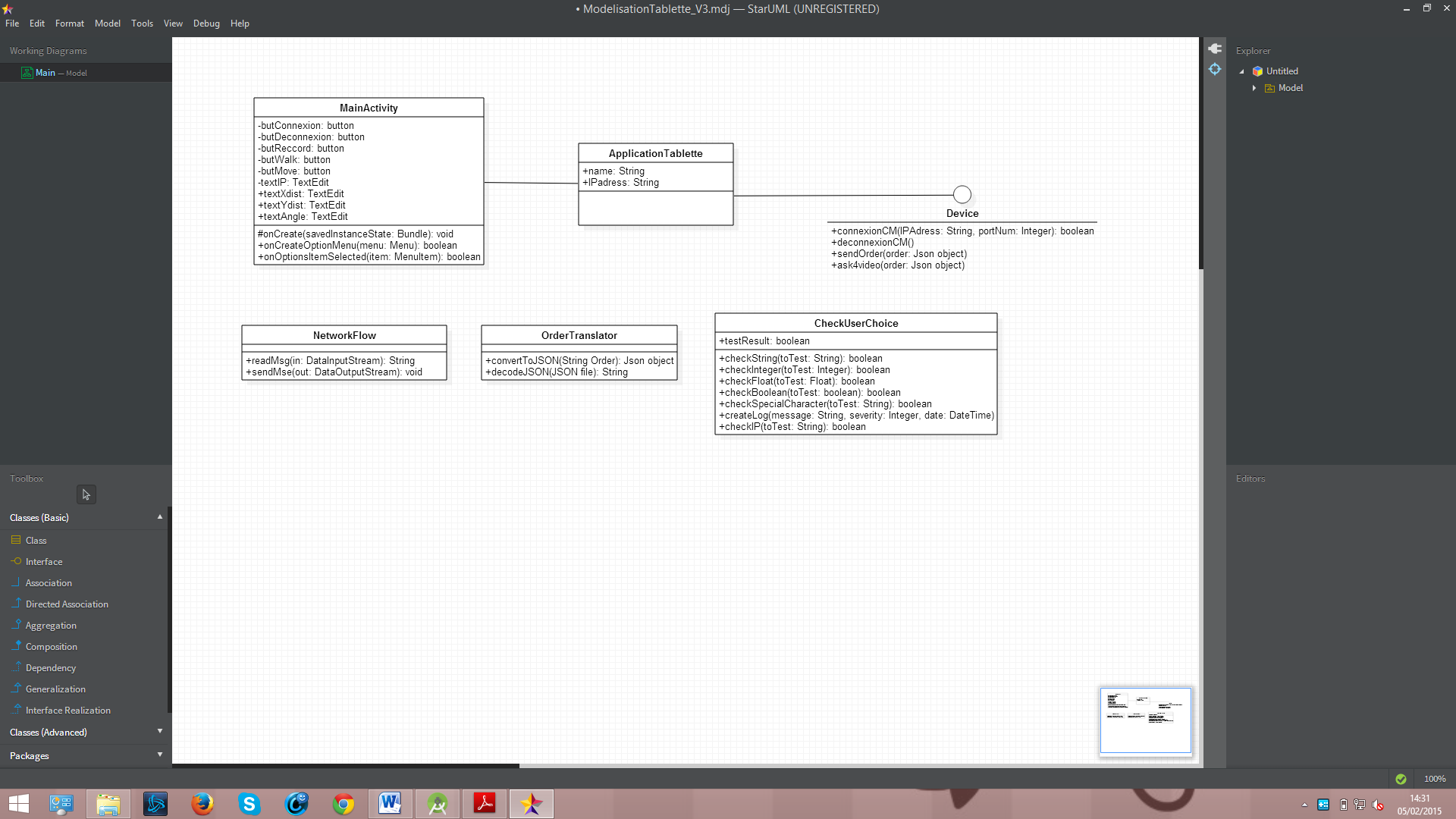
Les autres méthodes ont été auto générées à la création du programme et n'ont pas été retouchées.

Nous avons dans un premier temps de modifier les noms de nos méthodes, attributs et classes, afin de les traduire en anglais. L’ajout de deux classes a aussi paru nécessaire :

* Une classe OrderTransator, permettant de mettre en forme les ordres à envoyer aux robots ; en effet, nous avons décidé de transmettre des objets JSON.
* Une classe outil CheckUserChoise, permettant de vérifier que les informations renseignées par l’utilisateur sont cohérentes, et qu’elles pourront ensuite être traitées.



Lors de la présentation de ce modèle et de la discussion qui en a suivi, nous avons décidé d’ajouter une classe « ApplicationTablette », permettant d’implémenter l’interface « Device » ; ainsi, la structure sera la même quel que soit le type de device et le langage utilisé (Java, C++, Python…). Cela nous permettra entre autre de mieux comprendre la façon dont seront codées chaque parties. De plus, la classe CheckUserChoice aura une nouvelle méthode, permettant de vérifier qu’une chaîne de caractère est bien une adresse IP.



# Implémentation

## Passage d’ADT dans Eclipse à Android Studio

Notre précédente application avait été développée grâce au « pluggin » ADT ajouté à Eclipse, cependant, Google Inc. ayant développé son propre IDE, Andoird Studio, et étant donné que nous avons choisi de travailler sur notre projet en dehors des séances d’APP, nous avons choisi de migré sous Android Studio. Voici un tutoriel expliquant la réalisation de cette migration :

<https://developer.android.com/sdk/installing/migrate.html>

Cependant, Android Studio permettant de créer des mises en pages facilement, et la nouvelle version de l’application étant très différentes de la première, nous avons choisi de créer une toute nouvelle application sous cet IDE.

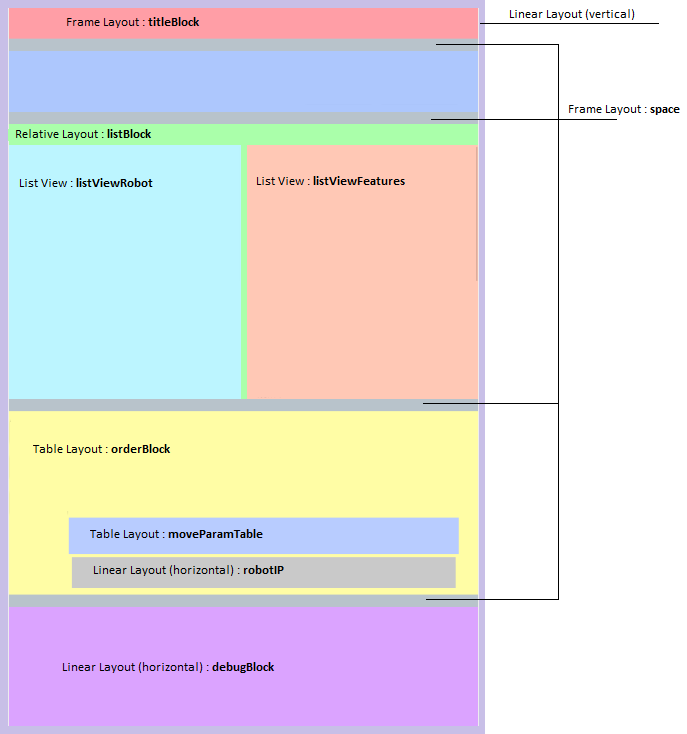
## Développement de l’interface graphique

Dans un premier temps, nous avons choisi de développer l’interface graphique, c’est-à-dire la « mise en page » de notre application. Nous avons choisi de développer l’application pour Android 4.0, afin de pouvoir tester celle-ci aussi bien sur le matériel mis à notre disposition par Polytech, que sur du matériel personnel.

* **Créations différents « blocs »**

La première étape a été de placer les différents blocs de notre application et de choisir le type de layout qui conviendrait le mieux pour les informations qu’ils contiendront par la suite. Nous avons ensuite choisi de fixer le sens de l’application en portrait, afin de ne pas avoir à gérer le changement de mise en page lors de la rotation de l’écran.

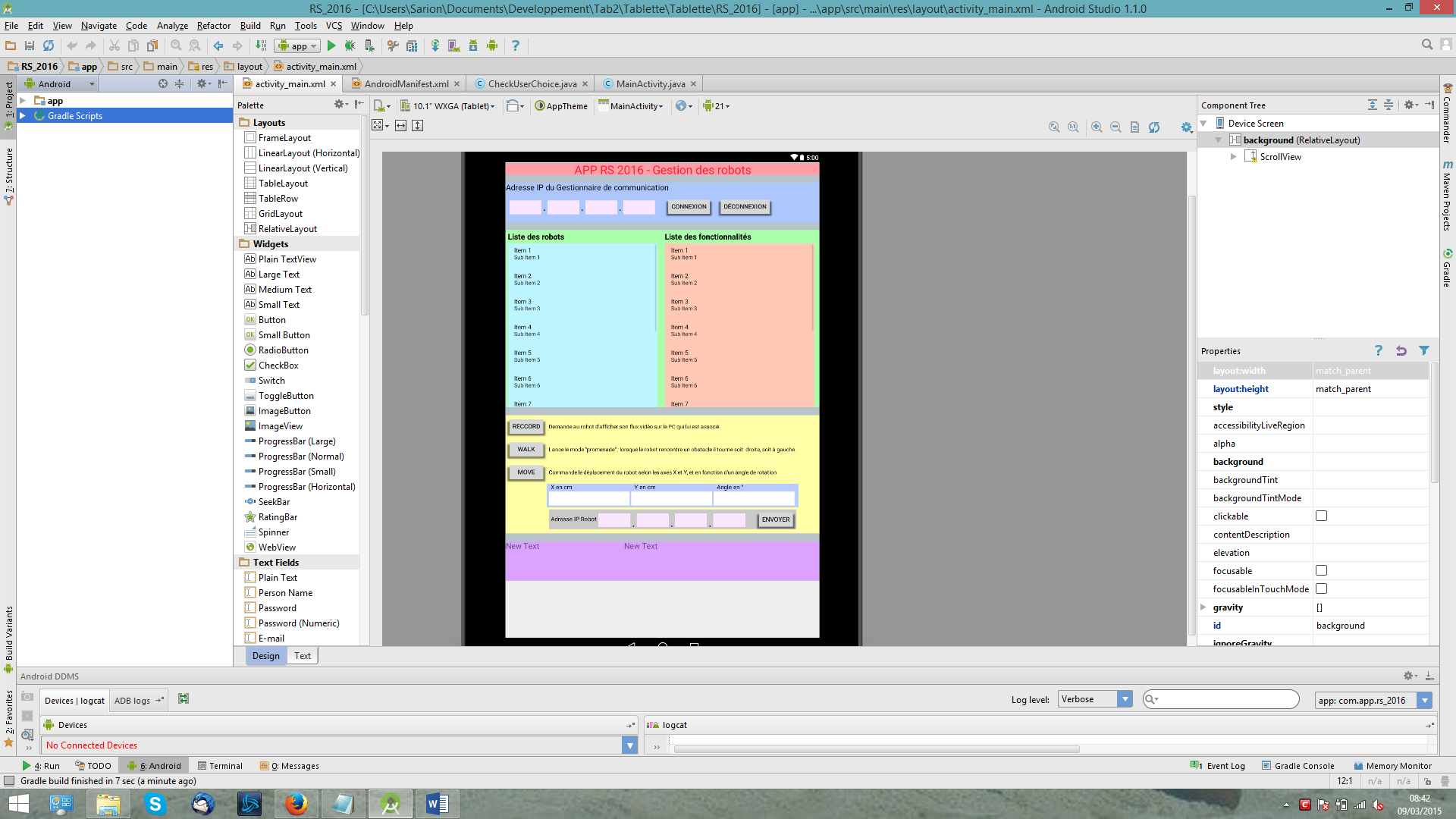
|  |  |
| --- | --- |
| Nom | Fonctionnalité |
| Frame Layout | Insère un cadre « fixe » dans lequel d’autres éléments peuvent être ajoutés. |
| Relative Layout | Insère un cadre « relatif ». |
| Table Layout | Insère une mise en page de type tableau. |
| Table Row | Insère une nouvelle ligne dans un tableau. |
| Linear Layout (vertical) | Insère une mise en page permettant de placer des éléments verticalement. |
| Linear Layout (horizontal) | Insère une mise en page permettant de placer des éléments horizontalement. |
| Liste View | Insère une liste. |



* **Tests pour l’adaptation de la taille de l’application en fonction de la taille de l’écran**

Dans le but d’obtenir une application dont la taille s’adapte à celle de l’écran, nous avons essayé de créer deux fichiers de type « layout ressource » ; un pour les écrans de 7 pouces, et un pour les écrans de 10 pouces. De plus, nous avons créé des variables dans le fichier *dimens.xml* afin de stocker dans un seul et même endroit les tailles des différents éléments. Deux variables ont été créées pour chacun d’eux ; un pour les tablettes dont l’écran est de 7 pouces (variable dont le nom se termine par *\_small*), et un pour celle de 10 pouces (variable dont le nom se termine par *\_normal*). Cependant, nous n’avons pas trouvé assez chacun de ces fichiers de layout à la taille d’écran associée.

Les différents composants ont ensuite été ajoutés dans les layouts afin d’obtenir la mise en page suivante :



Afin de contrôler les choix de l’utilisateur, les champs de saisie d’adresses IP ont été paramétrés de façon à n’accepter que des nombres composés de 3 entiers au maximum. De même, les champs de saisie des paramètres ont eux aussi été paramétrés. De plus, en fonction du EditText dans lequel on se trouve, soit le bouton « suivant » soit le bouton « ok » est affiché.

android:inputType="number" //EditText de type nombre

android:maxLength="3" //Taille maximale = 3 chiffres

android:numeric="integer" //Seul le type entier est accepté

android:imeOptions="actionNext"/> //Bouton « suivant » affiché

## Développement des fonctionnalités

* **Création d’une classe de vérification des données saisies**

Dans un premier temps, une classe *CheckUserChoice* a été développée, afin de vérifier les données saisies par l’utilisateur. La fonction *checkIP()* prend en paramètre d’entrée 4 nombres entiers correspondant aux différents octet de l’adresse IP à tester. Si ces 4 nombres sont compris entre 0 et 255 la fonction renvoie 1, sinon elle renvoie 0.

* **Ajout des objets dans la classe MainActivity**
* **Ajout des bibliothèques JSON**

<http://www.tutos-android.com/parsing-json-jackson-android>

<http://stackoverflow.com/questions/22127724/how-import-json-org-library-to-android-studio>

<http://theoryapp.com/parse-json-in-java/>

* **Récupération de l’adresse IP**

Attention, il est nécessaire d’ajouter les permissions suivantes dans le manifest Android :

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/>

<http://android-er.blogspot.fr/2011/01/get-wifi-ip-of-android-device-using.html>

<http://stackoverflow.com/questions/16730711/get-my-wifi-ip-address-android>