PROJET ONE SWITCH

Rapport de conception

Description

Document	Rapport_De_Conception.docx
URL	http://redmine.iut-info-vannes.net
Sujet	Étude préalable
Révision	1
Auteur	Groupe One Switch B
Etat	Finalisé
Diffusion	Client & tuteur

Membres du projet

MOA (client)	Étudiants		MOE (tuteur)
Willy ALLEGRE CMRRF de Kerpape	*Chef de projet*	Raphaël LE GORANDE	Matthieu LE LAIN
	Resp. com.	Mehdi HADDAD	
	Resp. doc.	Florent CATIAU-TRISTANT	
	Resp. tests.	Yoann BOYERE	
	Developpeur	Sacha Loriot	

Table des matières

1. Spécification du projet

1.1 Présentation du client

Ce projet a été réalisé pour le compte du CMRRF de Kerpape, et plus particulièrement son centre de recherche en électronique et informatique unique en France. Ses différentes missions sont l'adaptation et le développement de solutions techniques permettant l'accès aux TIC (**T**echniques **I**nformation **C**ommunication) par des personnes à mobilité réduite. Ils travaillent sur l'aide aux déplacements, à la communication, et à la rééducation.

1.2 Contexte

Ce projet est réalisé dans le cadre de notre cursus à l'IUT Informatique de Vannes. Durant celui-ci, nous devons produire une application répondant au besoin d'un client réel. Cela s'inscrit sous un module nommé "Projet de synthèse". Le projet se déroule du mois d'Octobre 2014 à début Avril 2015 et s'intitule « One Switch ». Une soutenance de projet sera mise en œuvre en fin de période pour évaluer le projet.

1.3 Objectifs

Notre objectif fut de produire, avant la date limite, une application permettant l'utilisation d'une tablette ou d'un smartphone exécutant Android à partir d'un unique contacteur (mécanique, pneumatique, logiciel, etc.). Le développement de cette application a suivi une organisation "projet" telle que celle établie en entreprise (méthode agile). Les membres du groupe avaient un rôle, les tâches étaient réparties au sein du groupe à l'aide d'un planning définit au préalable.

1.4 Cibles

One Switch cible toutes les personnes souffrant de difficultés motrices. Notamment les patients du centre de rééducation de Kerpape. La finalité de OneSwitch est de rendre l'accès à des appareils Android le plus facile et fluide possible.

1.5 Livrables attendus

Il est attendu une application exécutable sous Android, autrement dit un package au format APK installable sous ce système. Un manuel d'utilisation et d'installation de notre application ainsi qu'une documentation complète sont requis.

1.6 Contraintes et existant

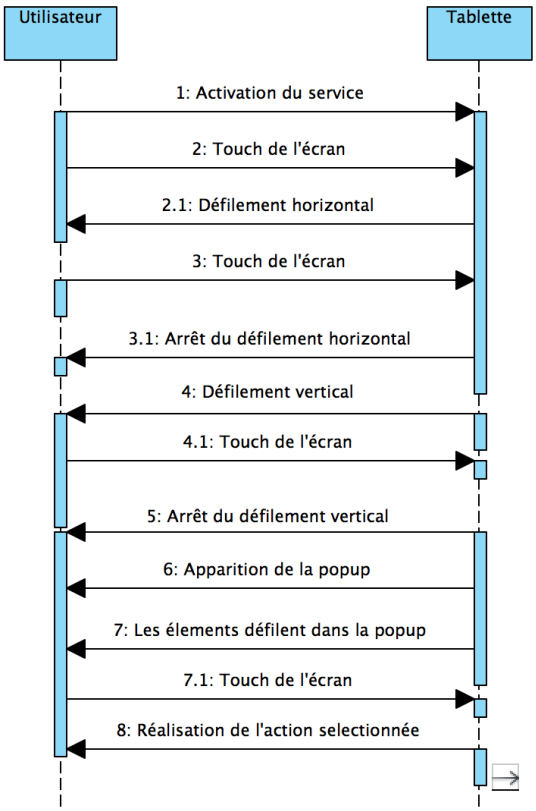
La contrainte principale est ici l'utilisation d'un seul et unique contacteur pour contrôler tout un environnement tactile (Smartphone, tablette), que l'on a l'habitude d'utiliser avec plusieurs doigts et gestes. Il faudra aussi permettre l'utilisation des boutons mécaniques disponibles sur ces appareils (Marche – Arrêt, Volume, etc.). L'application est de type service, elle tournera en permanence. Nous allons devoir utiliser l'API Android pour y implémenter l'application.

L'application, utilisant des fonctions systèmes de l'appareil, nécessite des droits et accès particuliers. De plus, la gestion d'éléments en premier plan hors application constitue une difficulté supplémentaire. L'utilisation d'un périphérique externe ajoute une contrainte qu'il nous faudra prendre en compte.

Il n'existe pas sur le marché de solution similaire pour Android, cependant, un équivalent est disponible sur le système d'Apple, iOS 8, qui se rapproche le plus de ce qui est attendu ici en terme d'accessibilité. Sous Windows, nous pouvons par exemple citer Civikey, permettant de diriger le curseur de la souris à l'aide d'un clic.

2. Spécifications UML

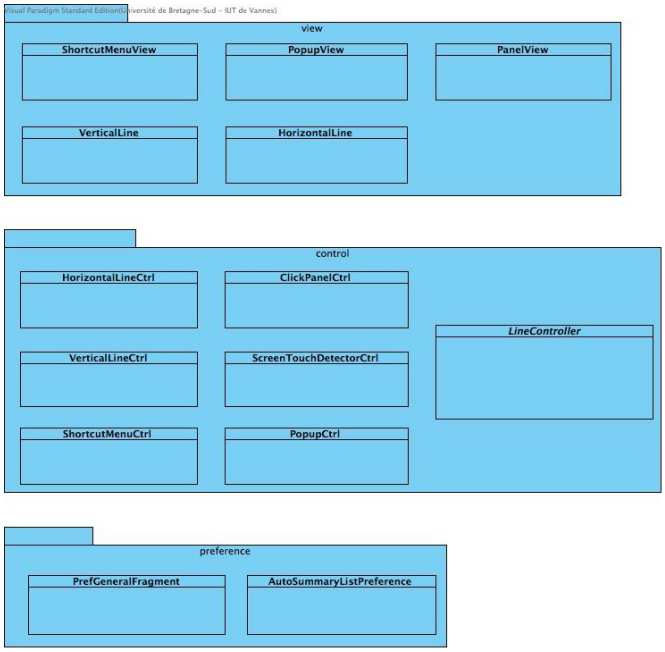
a) Diagramme de séquence



Ce diagramme de séquence présente une utilisation basique de l'application, sans rentrer dans les détails des méthodes du code.

En effet, l'application usant de l'API Android, beaucoup de méthodes sont utilisées par le système d'exploitation sans pouvoir en retracer le chemin d'exécution.

b) Diagramme de classes



Ce diagramme présente l'organisation globale des packages de l'application.
Une organisation MVC a été implémentée.

Afin de présenter un schéma simple et compréhensible, notre diagramme de classe a été grandement simplifié et n'est donc pas complet ici.

Le package **action** contient deux classes permettant d'interagir avec le système, autrement dit de simuler les clics, les gestes, et les boutons physiques. La classe *SpeakAText* permet quant à elle la synthèse vocale.

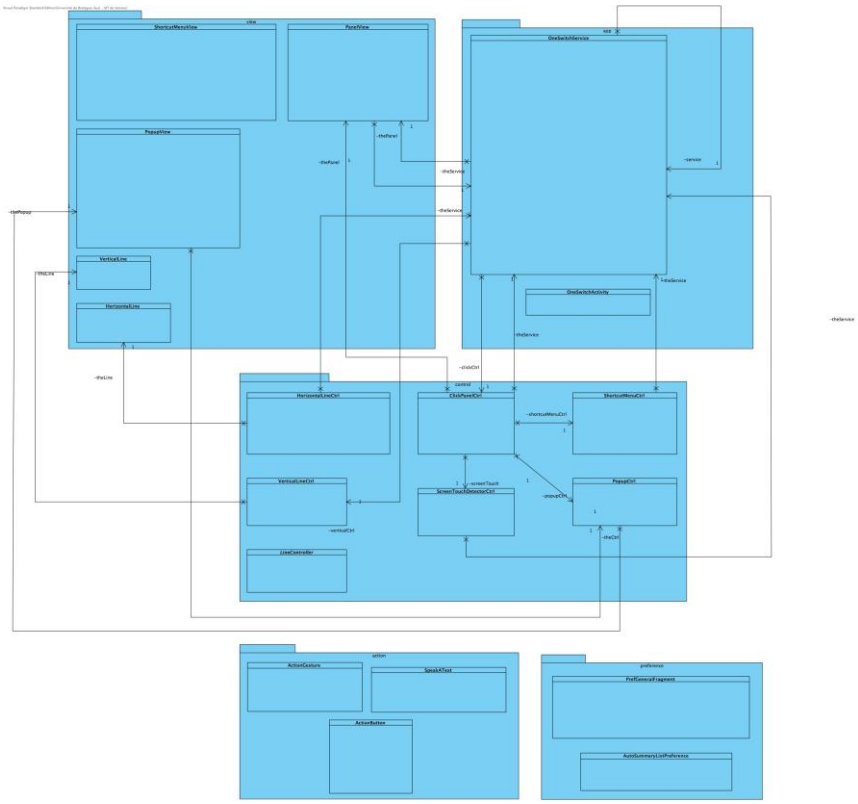
Le package **preferences** contient les classes relatives à la gestion des préférences.

Le package **view** contient les classes représentant les éléments visuels. Les classes *HorizontalLine* et *VerticalLine* représentent les lignes qui défileront sur l'écran. *PopupView* et *ShortcutMenuView* représentent elle les deux menus apparaissant sur l'écran lors d'un clic ou d'un clic long.

Le package **control** permet de contrôler les différentes vues. La classe *ClickPanelCtrl* nous permet de récupérer un « touch » sur la dalle tactile. C'est celle-ci qui coordonne les différentes actions des lignes, des pop-ups, etc…
Pour gérer le défilement des lignes, nous avons utilisés une classe abstraite *LineController* qui sera utilisée par *HorizontalLineCtrl* et *VerticalLineCtrl*.

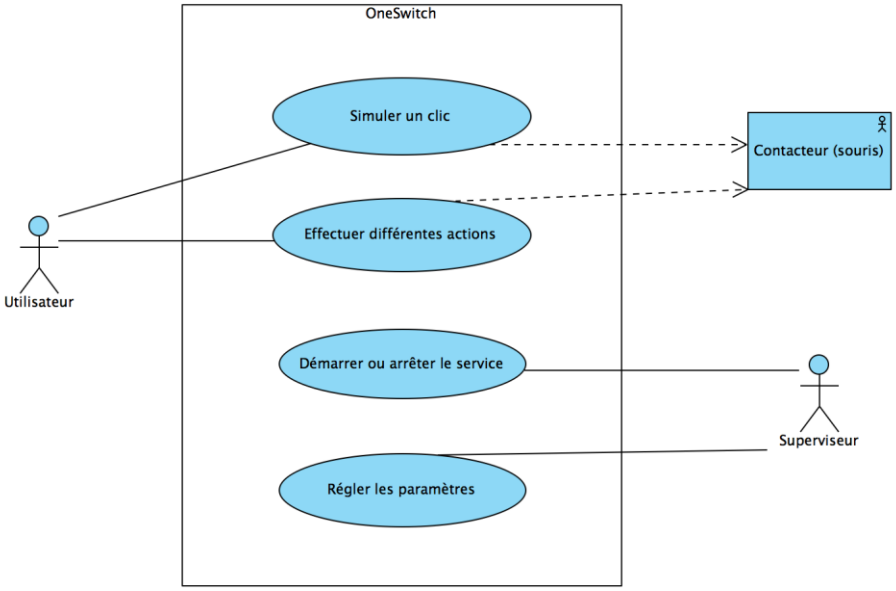
Le package **app** contient le service qui est le cœur de notre application, c'est lui qui s'exécute en permanence sur le périphérique. On y retrouve également les autres classes nécessaires à notre application. Entre autre les deux activités (principale et page à propos). Nous y avons également les classes pour gérer le démarrage automatique et la notification.
Enfin, y est présente la classe permettant d'avoir notre clavier personnalisé.

Ici sont schématisés les liens les plus importants entre les classes.



Un diagramme plus complet, bien que toujours simplifié, est disponible en annexe.

c) Diagramme de cas d'utilisation



Ce diagramme présente une schématisation des acteurs et de leurs quelques liens de l'application.

3. Choix algorithmiques

L'application a subi de nombreuses contraintes (voir 1.6), ce qui implique des choix algorithmiques rigoureux et complexes.

a) Interface

Nous avons choisi de faire passer les paramètres dans l'activité principale afin de rendre leur accès plus rapide, l'application ne nécessitant pas d'autre interface interne. Cette activité, composée d'un PreferenceFragment, contient de nombreuses préférences dont le type varie selon la fonctionnalité associée (switch, slider, liste etc.).

Nous nous sommes ensuite inspirés du modèle existant sur iOS pour construire nos menus (pop-ups). Chaque item est constitué d'un Button, d'un icone blanc et d'un icone noir (quand il est sélectionné). Chaque bouton possède un listener, correspondant à l'action qu'il doit effectuer.

b) Capture du clic

Afin d'annuler et de réinterpréter les clics sur l'écran tactile, nous utilisons des panels (vues) transparentes au premier plan ayant un listener. Ainsi, chaque clic est capturé par l'application et non interprété par le système d'exploitation.

c) Lignes de défilement

Chaque ligne est gérée avec de nombreux booléens qui indique son état (en mouvement, sens de déplacement, visible etc.) ainsi que des variables d'états, notamment pour le nombre d'itérations effectuées.

Pour leur déplacement, deux méthodes se sont offertes à nous.

La première consiste à utiliser les méthodes d'animations d'objet d'Android (*.animate()*). Les lignes défilent ainsi sur tout l'écran en un temps donné et sont fluides. Cependant, cette méthode ne fonctionne que sur l'API 21 ou au-dessus. En effet, l'animation d'objets n'est pas fonctionnelle sur un service, les vues ne sont pas mises à jour même si leur déplacement s'effectue.

La seconde est implémentée par des « Runnable » (Thread), les « AsyncTask » causant des bugs visibles sur les différentes fonctionnalités. Les lignes se déplacent pixel par pixel (en fonction de la densité) pour garantir une certaine fluidité. Chaque déplacement est séparé par un délai, calculé en fonction de la vitesse paramétrée. Pour un changement de vitesse plus visible, la valeur du déplacement (en pixel) peut être multipliée par 2 voire 3.

```
private class HorizLineRunnable implements Runnable{
	public void run(){
		try{
			if (getIterations() == ite)
				stop();
			if(isMoving){
				if((horizParams.y <= size.y)&&(isMovingDown)){
					horizParams.y += density;
					if(horizParams.y >= (size.y-density))
						isMovingDown = false;
				}
				else{
					horizParams.y -= density;
					if(horizParams.y <= density){
						isMovingDown = true;
						addIterations();
					}
				}
				theService.updateViewLayout(theLine, horizParams);
				handler.postDelayed(this, (int) (55-(5*speed)));
			}
		}
	}
}
```

Notre code possède les 2 méthodes, afin d'utiliser la meilleur des deux lorsque cela est possible. Ainsi, si l'application est installée sur Android Lollipop (5.0) ou plus, le défilement des barres sera nettement plus fluide.

d) Simulations

Nous utilisons à plusieurs reprises des processus système super-utilisateur afin de lancer certaines actions demandant des droits privilégiés (le clic, le glisser, ou bien la simulation de boutons physiques). Ces commandes sont effectuées via le *shell* de l'appareil.

```
Runtime.getRuntime().exec("su -c input keyevent " + KeyEvent.KEYCODE_BACK);
```

Cette technique évite de passer l'application à l'état d'application *système*, et comprend donc une installation basique et rapide de celle-ci.

Par exemple, le clic à l'intersection des lignes de pointage est effectué via une de ces commandes. Les boutons physiques sont lancés via des KeyEvent, qui reproduisent l'action envoyée lors d'un clic sur ces boutons.

Nous utilisons également plusieurs BroadcastReceiver pour récupérer des évènements comme le verrouillage du téléphone, le démarrage du téléphone, ou bien la réception d'un appel.

```
public class MyReceiver extends BroadcastReceiver {
	@Override
	public void onReceive(Context context, Intent intent) {
		//Réception d'un appel
		if (intent != null
			&& intent.getAction().equals(
			TelephonyManager.ACTION_PHONE_STATE_CHANGED)
			&& intent.hasExtra(TelephonyManager.EXTRA_STATE)
			&& intent.getStringExtra(TelephonyManager.EXTRA_STATE)
			.equals(TelephonyManager.EXTRA_STATE_RINGING)
			&& !call){
		call=true;
```

Pour chaque fonction particulière, un panel avec un listener spécifique est ajouté sur l'écran pour capturer l'action du clic, puis celui-ci est retiré pour continuer le fonctionnement normal du service, sans permettre à l'utilisateur de cliquer sur l'écran de façon normale.

La synthèse vocale est implémentée par l'objet *TextToSpeech*, présent dans l'API Android. Une méthode permet d'énoncer via la voix de synthèse de l'appareil la chaine de caractère passée en paramètre.

Le code est entièrement documenté et commenté en **Français** pour une meilleure compréhension.