PJI - Instrumenting Android Code :

MASTER -1 Informatique

Bizimungu Steve

1.Objectif

L'objectif de ce projet est de réaliser un outils qui permettra d'instrumenter un code (Java Android) , en y incorporant du code pouvant servir à tracer le déroulement d 'une instance d'une application en d'autre mots on doit pouvoir détecter quand un événement s'est déroulé et réagir

1.1. Gestion des événements (InputEvents) dans Android .

Pour la gestion des événements Android , Android utilise plusieurs techniques ,quand il s'agit de traiter des interactions entre l'utilisateur et l'interface de l'application , l'approche consiste à capturer les événements dans les éléments de l'interface avec laquelle l'utilisateur interagit .

La clasee View est une classe de base dans Android qui déclare des fonctions définissant comment le framework Android est sensé réagir quand un événement d'un type particulier survient . Ainsi pour un composant de l'interface utilisateur donnée si on désire intercepter un événement donné ou personnaliser la réaction lorsque cet événement survient il faut définir une classe étendant ce composant est surchargé ses méthodes .

Pour s' alléger de cette charge,Android a prévu une technique qui consiste à encapsuler des interfaces appelés **event listeners** qui permette de capturer les événement et propose des fonction à appeler quand ses événements survient.

1.2.Détection des Event Handler et event listener par Spoon

L'objectif de ce projet étant de collecter des mesures pendant l'exécution d'une application ,en se basant sur certains points essentiels, ici les événements , un façon pour y arriver serait de mettre en œuvre un moyen nous permettant d'inclure un code qui nous est propre dans les méthodes qui seront appelées lorsque ces événements sont déclenchés, c'est là qu'intervient **spoon** .

Spoon est une librairie qui permets d'analyser est de transformer un code Java .Il fournit un modèle (arbre) dans lequel chaque élément d'un programme donné peut être accédé en lecture ou en modification . Spoon prend en entrée un code source et retourne un code transformé près à être compiler .

Instrumentation du code java Android avec Spoon

1.Mise en oeuvre de la solution

Le code à instrumenter est un code java d'une application Android , les points essentiels à détecter sont les événements ,donc les méthodes qui seront appelées à l'issu de ces derniers .

Ces méthodes peuvent être subdiviser en deux grandes catégories :

1. **Les méthodes (call back) déclarées dans les composants grahiques(View,button,..)** ou dans des classes implémentant ces composants , pour accéder à ces méthodes on utilisera un processeur (Processor dans spoon) qui les détecte dans les classe où elles sont déclarées.

2. **Les méthodes se trouvant dans les interfaces encapsulés :** ces méthodes sont déclarées dans des interfaces particulières regroupées sous le nom de «Event Listener » pour les détecter on peut détecter toutes les classes implémentant ces event listeners .

2.Conception :

L'objectif est de fournir un architecture des classe la plus simple possible , tout en essayant de respecter toutes les bonnes pratiques de la programmation objet (extensibilité ,lisibilité ,etc..)

Un pattern de conception a spécialement dominer pour ce projet c'est le **pattern strategy** .
L'application est composée d'un processeur spoon ce processeur prend une liste d'objets « **Method »** qui sont des objets spécifiques aux méthodes désirées ,spécifiant comment reconnaître la fonction donnée dans le code source à analyser (méthodes **checkMethod , CheckStrategy,..) ,**et une fois détecter, défini quelles en sont les transformations à effectuer ou les information à recueillir (ce là peut être rajouter du code , récupérer des informations ,etc ….)

1.OnEventHandlerProcessor

Le projet contient un processeur spoon (**OnEventHandlerProcessor)**, qui prends un liste de méthodes de type **Method** le tout se lance dans une classe **Starter** qui permet de rajouter le processeur spoon utilisé les méthodes à chercher et le lancer avec un certain nombre d'arguments .

Architecture des Classes

Dans le framwork Android un « event handler » est une méthode qui est appelée quand un événement donné est déclenché .Pour des événements issus de l'interaction entre l'utilisateur et l'interface de l'application , ces méthodes sont déclarées dans dans android.view.View (classe android.view.View ,) ,
View est la classe de base pour construire des composants de l'interface utilisateur (bouton , champs de text..) cette classe est aussi utile pour la détection et le traitement des événements .
Pour détecter un événement il peut être judicieux de pouvoir se placer dans la méthode qui sera appelée quand cette événement sera déclenché, dans le monde android ces méthodes sont des méthodes de la classe View et toute classe qui étend cette dernière ou les méthodes des interfaces **event Listeners** . qui y sont encapsulées

L'objectif ici est de pouvoir les détecter dans n'importe quelle sous classe de android.view.View ou toute classe implémentant un une interface **Event Listener** (OnclickListener,…...

L'interface **Method** déclare 3 méthodes :
- ***checkMethod*** : Une méthode qui prends en paramètre une CtMethod <?> et vérifie si elle correspond à la méthode recherchée.
L'utilité de déclarer un méthode comme celle là est qu'elle permettra de rechercher une méthode donnée selon une stratégie voulue

-. Une méthode ***findMethod*** qui prend en paramètre une collection de CtMethod<?> et retourne la méthode ayant passe le test checkMethod .

-.Une méthode ***process*** qui prend en paramètre une CtMethod<?> et accèdent ou modifie les caractéristiques de celle ci par caractéristiques on entend sont type de retour ses paramètres etc..
l'utilité de cette méthode du point de vue stratégique est évidente elle permettra de personnaliser les traitements .

La classe **OnEventHandler** implement l'interface Method , toute la logique de l'identification des fonctions est définie dans cette dernière .
Les events handler y sont détectés par leur nom et leur signature ,un fois detecter la methode process se contente pour l'instant d'inclure un commentaire commentaire

Les classe **ItfOnEventHandler et ViewOnEventHandler ,** étendent OnEventHandler
ItfOnEventHandler est utilisé pour représente tous les event handlers définit dans les event listeners.

ViewsOnEventHandler quant à elle détecte les event handlers définit dans des éléments de l'interface graphique.

En fin la classe starter permet d' ajouter les méthodes et le processeur

Un autre processeur ViewProcessor est une classe qui sert de témoins pour les tests elle permet d'afficher toutes classes et les méthodes dans un projet donné

A réaliser...

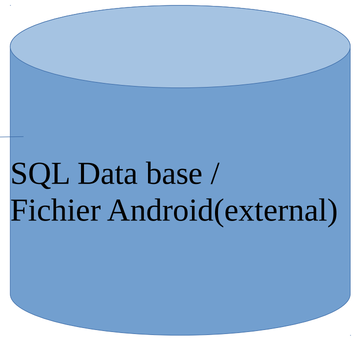
Ce qui reste à réaliser est d'ajouter dans les méthodes détecte un certains code .

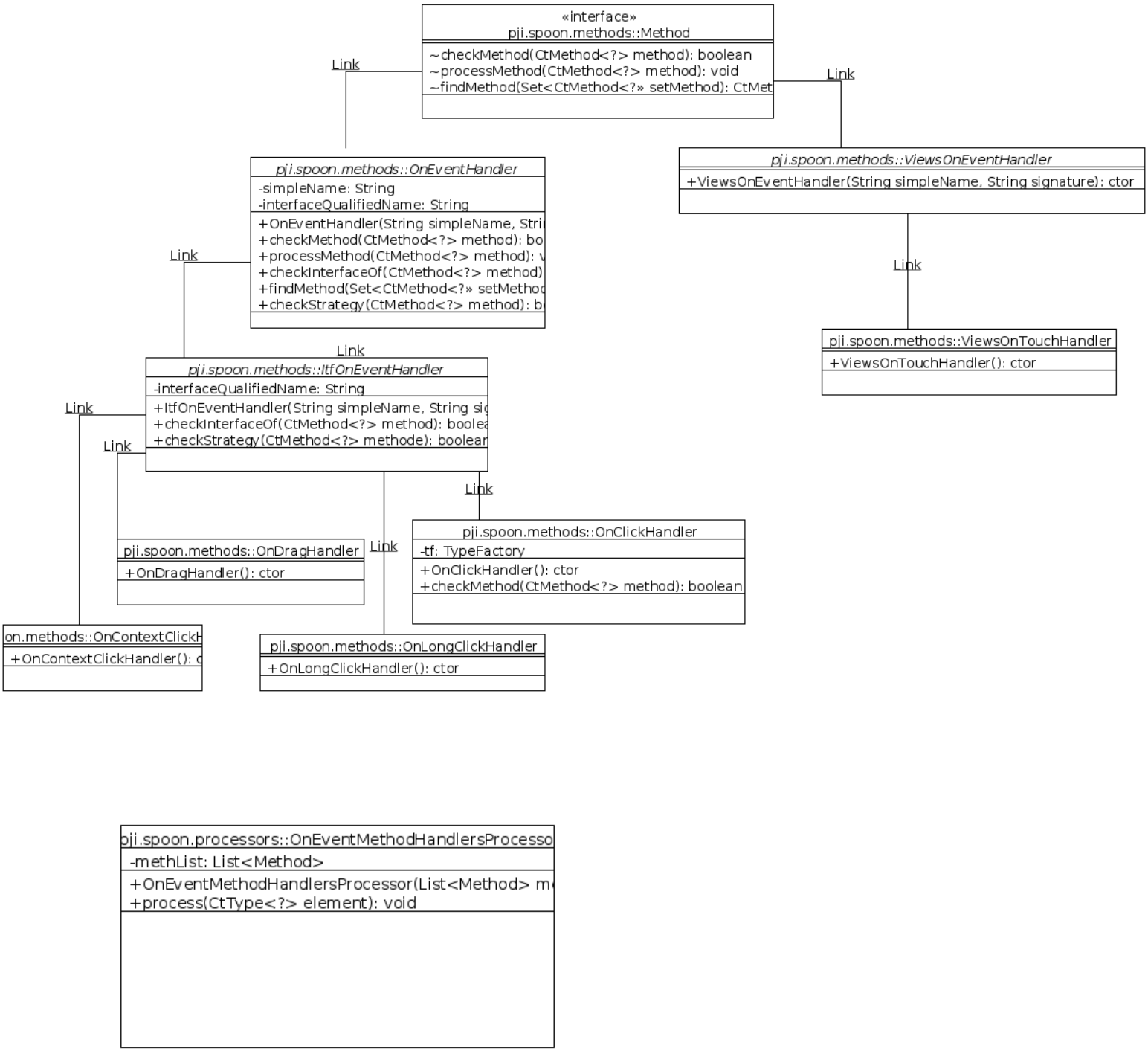
Pour ce qui est de la nature du code à ajouter j'envisage inclure un code permettant de d'enregistrer un événement quand il survient , l'enregistrement se ferait dans un fichier de log (fichier android , une base sql..)
Le code à ajouter dans l' event handler pour être du genre :
SI (un événement apparaît)
ALORS enregistrer les information sur la méthode qui a été appelée à l'issue de cet événémént
FIN SI

```
public class MyActivity extends Activity {

    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.content_layout_id);
        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
://///AJOUTER DU CODE ICI/////////////////////////
SI cet evenement apparaît neregister dans la base
////////////////////////////////////////////////////////////////////////


                // Perform action on click
            }
        });
    }
}
```




«interface»
pji.spoon.methods::Method
~checkMethod(CtMethod<?> method): boolean
~processMethod(CtMethod<?> method): void
~findMethod(Set<CtMethod<?» setMethod): CtMet
Link
Link
pji.spoon.methods::OnEventHandler
-simpleName: String
-interfaceQualifiedName: String
+OnEventHandler(String simpleName, Stri
+checkMethod(CtMethod<?> method): bo
+processMethod(CtMethod<?> method): v
+checkInterfaceOf(CtMethod<?> method)
+findMethod(Set<CtMethod<?» setMetho
+checkStrategy(CtMethod<?> method): b
pji.spoon.methods::ViewsOnEventHandler
+ViewsOnEventHandler(String simpleName, String signature): ctor
Link
Link
pji.spoon.methods::ViewsOnTouchHandler
+ViewsOnTouchHandler(): ctor
Link
pji.spoon.methods::ItfOnEventHandler
-interfaceQualifiedName: String
+ItfOnEventHandler(String simpleName, String sig
+checkInterfaceOf(CtMethod<?> method): boolea
+checkStrategy(CtMethod<?> methode): boolean
Link
Link
Link
pji.spoon.methods::OnClickHandler
-tf: TypeFactory
+OnClickHandler(): ctor
+checkMethod(CtMethod<?> method): boolean
pji.spoon.methods::OnDragHandler
+OnDragHandler(): ctor
Link
on.methods::OnContextClickH
+OnContextClickHandler(): c
pji.spoon.methods::OnLongClickHandler
+OnLongClickHandler(): ctor
pji.spoon.processors::OnEventMethodHandlersProcesso
-methList: List<Method>
+OnEventMethodHandlersProcessor(List<Method> m
+process(CtType<?> element): void

Le schéma si dessous illustre les méthodes à détecter selon la hiérarchie des packages dans android .pour l'instant il n'a que les méthodes se trouvant dans la classe **View** dont la détection est implémentée ainsi que les méthodes déclarée dans les interfaces **Android.view.*Event*Listener »(**en remplaçant E**vent** par , Cliclk ,LongClick,……)