Hes·so VALAIS WALLIS
School of Engineering

PATTERN BUTTON MANAGER

Jean Nanchen

OCTOBER 23, 2020

HES-SO – 3ÈME ANNÉE

Table des matières

1. INTRODUCTION

Dans ce laboratoire, nous allons implémenter des « patterns » vu en cours, tel que le « Subject/Observer » ainsi que le « callback ». Pour ce faire nous avons développé un « Button Manager » qui permet de détecter un appui long ou court sur les boutons de notre board. Le résultat est envoyé en UART à notre ordinateur. Un retour visuel grâce aux LEDs présentent sur notre board est aussi implémenté.
Github : https://github.com/73jn/ButtonManager

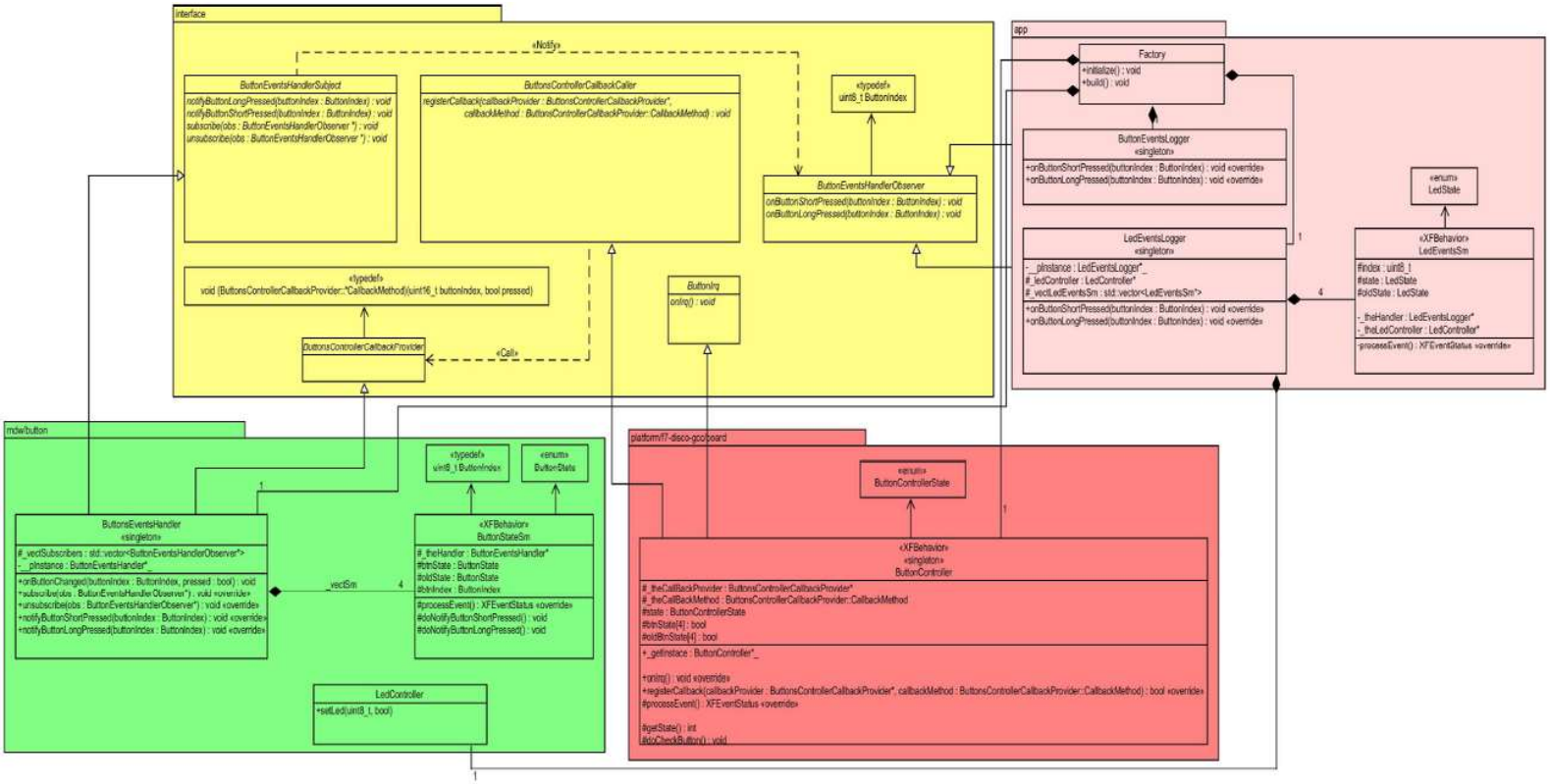
2. CAHIER DES CHARGES

Les objectifs à remplir sont :

- Créer une classe ButtonController qui reçoit des interruptions & qui filtre les rebonds.
- La classe ButtonController envoie les informations « Release & Press » via un pattern callback.
- La classe ButtonsEventsHandler doit détecter les long press et les short press.
- La classe ButtonsEventsHandler notifie la classe ButtonEventsLogger via un pattern Observer.
- Extra : Implémenter la classe LedEventsLogger.

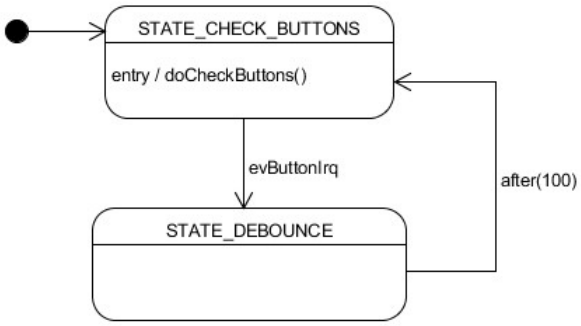
3. DIAGRAMME DE CLASSE

Dans ce diagramme de classe, nous observons les relations entre les classes. On y retrouve 4 différents « package », tel que le package interface qui contient des classes abstraites de pattern. Il y a ButtonEventsHandlerSubject et ButtonEventsHandlerObserver qui permettent un pattern observer.
Les deux autres classes (ButtonsControllerCallbackCaller & ButtonsControlerCallbackProvider) permettent un pattern callback.
Le package « board » contient la classe ButtonController qui lors d'une interruption filtre les rebonds de l'interruption. Le signal est alors envoyé à la classe ButtonEventsHandler avec un callback.
La classe ButtonEventsHandler dans le package « button » permet de détecter un appui long sur le bouton d'un appui court. Le résultat est envoyé à tous les « observers » grâce à un pattern « observer ».
Les observer sont dans la classe app, qui contient un ButtonEventsLogger (envoie sur le port série des informations) ainsi qu'un LedEventsLogger (flash les leds en fonction des informations).



4. ANALYSE

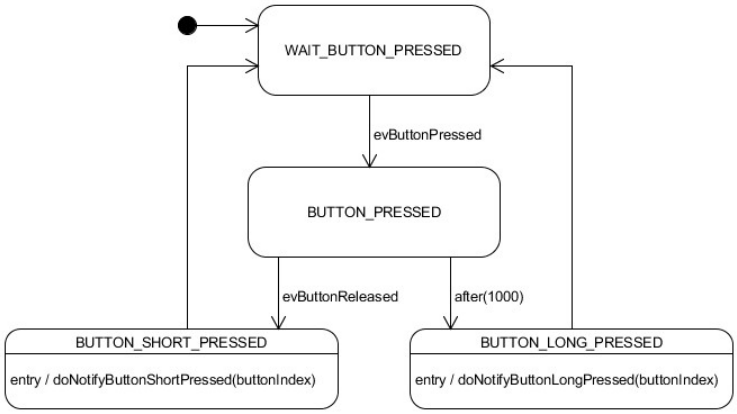
BUTTONCONTROLLER



Dans la classe ButtonController, nous effectuons un filtrage pour éviter les rebonds de notre interrupteur. Pour cela, nous utilisons une machine d'état qui lors d'un rising edge ou un falling edge, passe dans un état STATE_DEBOUNCE et attends 100ms avant de retourner dans l'état STATE_CHECK_BUTTONS où l'on vérifie l'état des boutons.

Ensuite, on envoie avec un pattern « callback » les boutons qui ont changé d'état à la classe ButtonEventsHandler

BUTTONEVENTSHANDLER

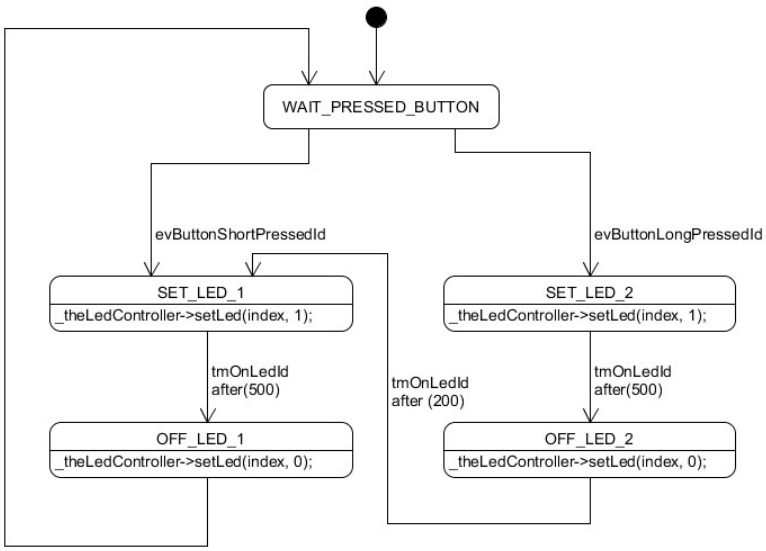


La classe ButtonEventsHandler possède une méthode « onButtonChanged(index, pressed) » qui se fait appeler par la classe ButtonController..

La classe reçoit alors l'ID du bouton ainsi que son nouvel état.

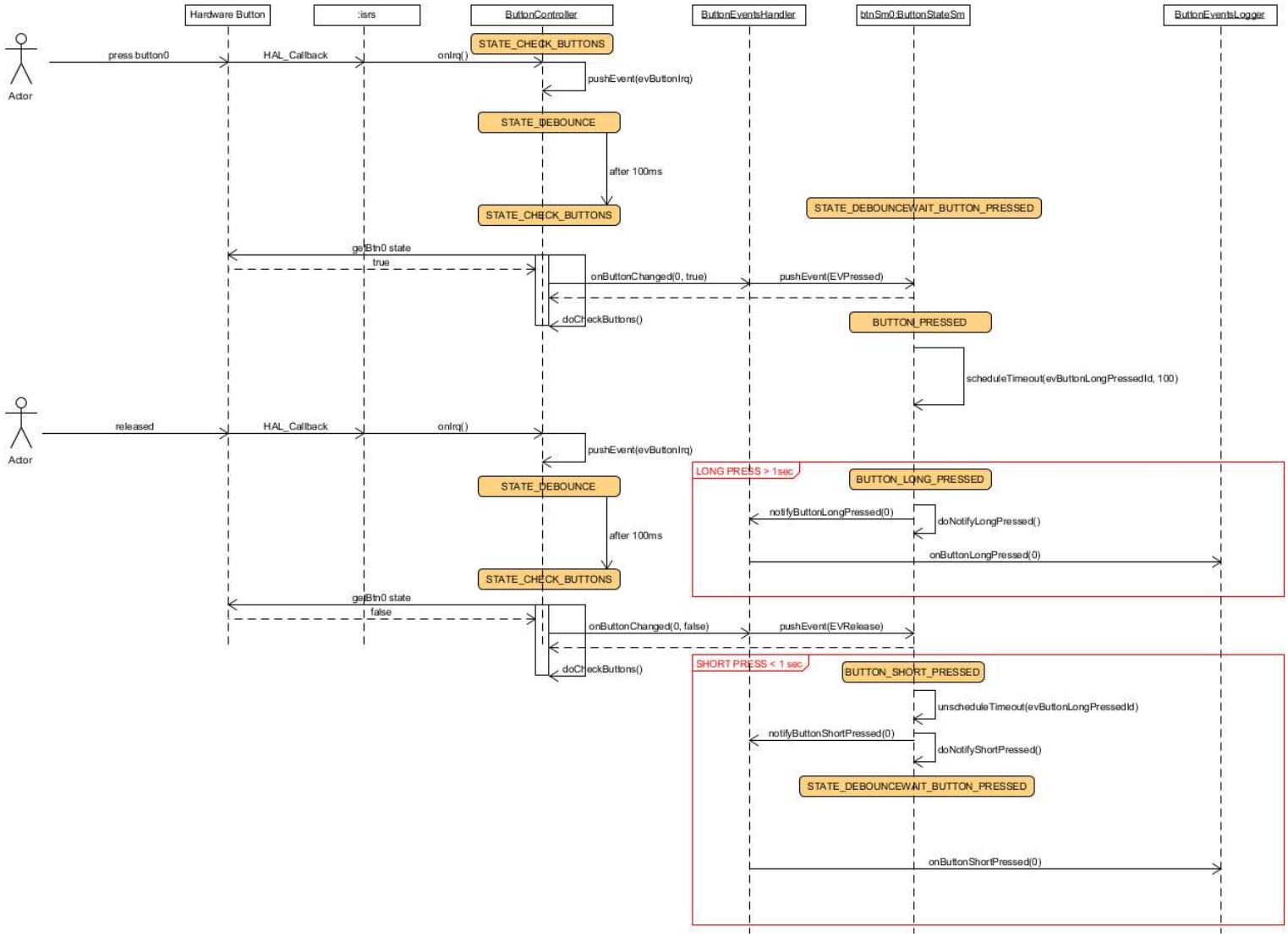
Avec ses informations, une machine d'état **PAR BOUTON** (ButtonStateSm) permet de filtrer les appuis court des appuis long. Au départ dans WAIT_BUTTON_PRESSED, on passe à l'état BUTTON_PRESSED lorsque l'on reçoit un événement que le bouton a été pressé. Dans cet état, on lance un timeout de 1 seconde. Si l'on relâche avant 1 seconde le bouton, on passe dans l'état BUTTON_SHORT_PRESSED et on notifie le ButtonEventsHandler d'un appui court. Dans le cas contraire (on ne relâche pas le bouton avant 1 seconde), BUTTON_LONG_PRESSED et l'on notifie le ButtonEventsHandler d'un appui long.

LEDEVENTSLOGGER



Cette classe est similaire au fonctionnement du ButtonsEventsHandler, car elle contient aussi une machine d'état par bouton (LedEventsSm).
L'état WAIT_PRESSED_BUTTON est un état où l'on attend une notification par le ButtonEventsHandler. Si le « handler » notifie un appui court, on passe dans SET_LED_1 qui allume la LED correspondante et active un timeout de 500ms qui va permettre de passer au prochain état, OFF_LED_1. Cet état éteint la LED et envoie un event pour retourner dans WAIT_PRESSED_BUTTON.
Si c'est un appui long, on fait la même chose que pour un appui court mais on flash la LED 2 fois.

DIAGRAMME DE SÉQUENCE



Voici un diagramme de séquence qui représente une personne qui appuie sur le bouton moins d'une seconde (SHORT PRESS).

On peut voir que l'interruption appelle la méthode « onIrq() » du ButtonController. Dans cette méthode, on envoie un « event IRQ » dans la queue. Cet « event » va faire changer d'état la machine d'état (on passe à STATE_DEBOUNCE pour attendre la fin des rebonds). Une fois avoir attendu 100ms, on mesure l'état des boutons dans STATE_CHECK_BUTTONS. On envoie l'informations des boutons qui ont changés d'état par un pattern callback avec la méthode « onButtonChanged(index, state) » de la classe ButtonEventsHandler. Une fois dans cette méthode, on envoie un « event » dans la queue du XF de ButtonEventsHandler comme qui le bouton à été pressé. On passe alors dans un état (BUTTON_PRESSED) qui attend soit la fin du timeout de 1 seconde, soit le relâchement du bouton (< 1 seconde). Notre utilisateur relâche le bouton avant ces 1 seconde, et nous allons passer à l'état BUTTON_SHORT_PRESSED. Dans cet état on unschedule le timeout de 1 seconde et on notifie les observers.

5. TESTS

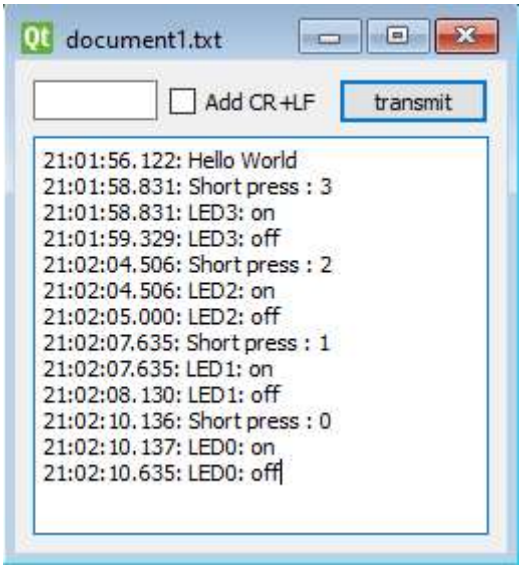


Figure 1 - Short press sur tous les boutons

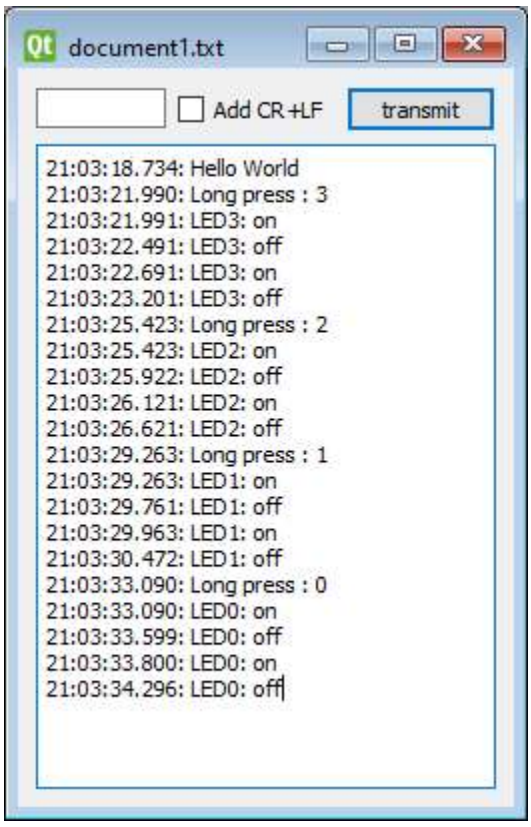


Figure 2 - Long press sur tous les boutons

6. RÉSUMÉ DES TESTS

Test	Résultat	Remarque
Recevoir des interruptions	Fonctionnel	
Lire l'état des boutons	Fonctionnel	
Filtre antirebond	Fonctionnel	
Callback ButtonController-ButtonEventsHandler	Fonctionnel	
Détecter les appuis long et court	Fonctionnel	
Notifier les observers	Fonctionnel	
Envoyer sur le port série les informations	Fonctionnel	
Flasher les LEDs	Fonctionnel	
Écrire sur la carte SD	Non implémenté	Optionnel

7. CONCLUSION

Durant ce projet, nous avons appris à utiliser des patterns pour communiquer entre nos classes. Nous avons utilisé l'architecture Exécution Framework développé dans un projet précédent. Le projet fonctionne à 100% (il manque uniquement l'extra carte SD, mais non obligatoire).

8. ANNEXES

Github : github.com/73jn/ButtonManager