HAUTE ÉCOLE D'INGÉNIEUR DU VALAIS

LABO - BUTTONSCONTROLLER

REAL TIME PROGRAMMING

Samy Francelet
6 décembre 2021

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Ce laboratoire a servi a implémenter des patternes orienté-objet, servant à découpler différents modules. Ces patternes permettent de réaliser du code plus portable, moins dépendant du Hardware de base. Le module *XF (Execution Framework)* est utilisé ici pour créer des machines d'états et aidé a réaliser un code pseudo-parallèle.

1.2 Objectifs

L'objectif de ce laboratoire est de réaliser un loggeur de bouton, qui affiche si le bouton a été appuyé rapidement, ou laissé enfoncé. Chaque *couche* de ce projet doit être découplée au mieux de la *couche* en dessous, pour rendre le projet portable et adaptatif. Les *couches* sont les suivantes : Application, avec la *Factory* et le *ButtonEventsLogger*. Middleware, avec le *ButtonEventsHandler* et les *ButtonStateSM*. Et au fond, la Board Layer avec le *ButtonsController*. Le module *XF* développé aux précédents labos a été utilisé pour permettre un programme pseudo-parallèle.

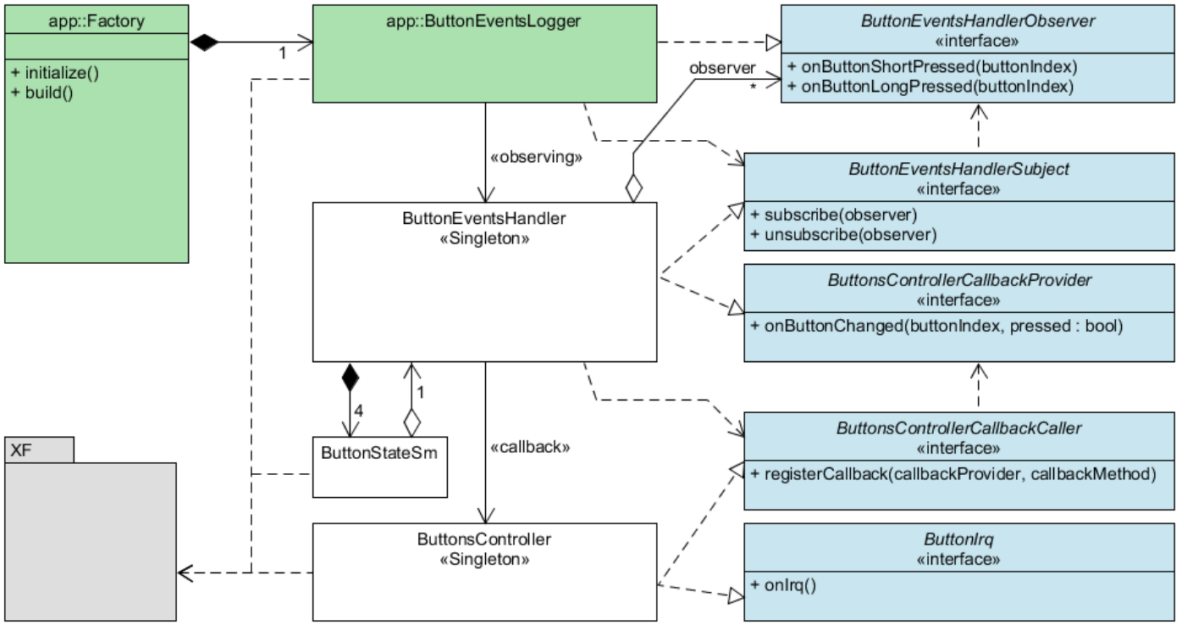


FIGURE 1.1 – Diagramme de classe complet du contrôleur (provenant du *Guide ButtonsController* [1])

1. Médard RIEDER. et Thomas STERREN. « Button Manager - Laboratory Guide ». In : (). URL : `https://cyberlearn.hes-so.ch/mod/folder/view.php?id=1539125`. (accessed : december 2021)

Chapitre 2

Conception

2.1 Board Layer

Après une configuration de la board réalisé avec *STM32CubeMX* (détail de la configuration dans le *Guide ButtonsController* [1]) Dans la *Board Layer*, la machine d'état du *ButtonsController* est implémentée.

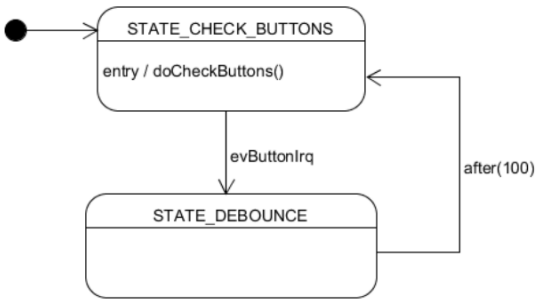


FIGURE 2.1 – State diagram du BoutonsController (provenant du *Guide ButtonsController* [1])

Le fonctionnement de cette couche est simple : une interruption Hardware enclenche le *STATE_DEBOUNCE*. Après 100 millisec, on retourne dans l'état d'attente, et on contrôle l'état de tous les boutons. Ce temps d'attente permet de réaliser un anti-rebond software.

1. RIEDER. et STERREN., « Button Manager - Laboratory Guide »

2.2 Middleware Layer

Le *Middleware* implémente la détection d'appui court ou long sur un bouton. Elle reçoit les informations de la *Board Layer* pour son fonctionnement, et transmet les informations d'appui court ou long à la couche *Application*

2.2.1 Liaison avec la Board Layer

La liaison avec la couche en dessous ce fait ainsi :

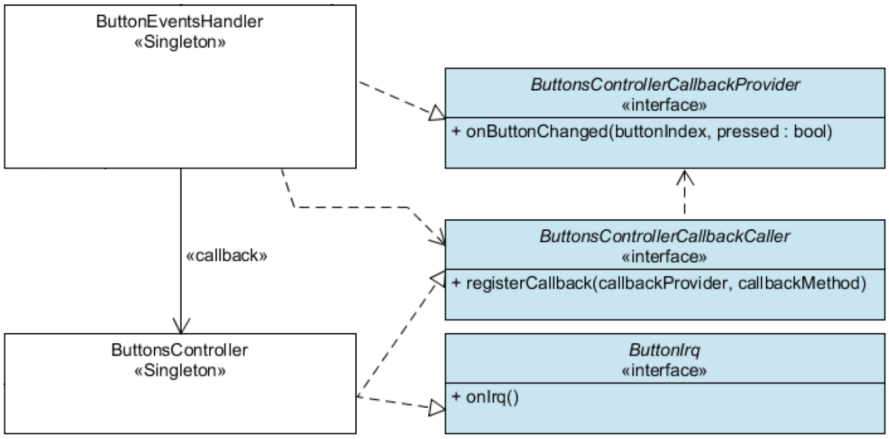


FIGURE 2.2 – Diagramme de classe du mdw (provenant du *Guide ButtonsController* [1])

Le *ButtonEventsHandler* implémente l'interface *ButtonsControllerCallbackProvider* pour fournir la méthode **onButtonChanged** en méthode de callback. La méthode de callback sera appelée par le *ButtonsControllerCallbackCaller*. Cela permet donc au *ButtonsController* d'informer le *ButtonEventsHandler* d'un changement sur un bouton, sans aucune liaison direct entre les deux classes.

1. RIEDER. et STERREN., « Button Manager - Laboratory Guide »

2.2.2 ButtonEventsHandler

Le gestionnaire d'évenement boutons *ButtonEventsHandler* fonctionne ainsi :

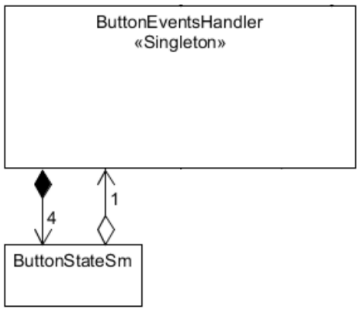


FIGURE 2.3 – Diagramme de classe du *ButtonEventsHandler* (provenant du *Guide ButtonsController*[1])

Lorsqu'il reçoit une information sur un bouton, il transmet un évenement au *ButtonStateSM* correspondant. Chaque *ButtonStateSM* possède une machine d'état interne :

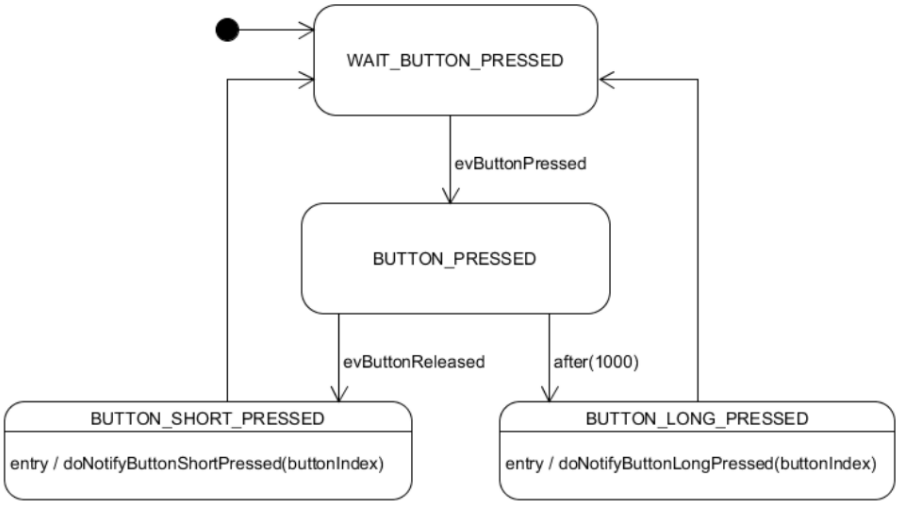


FIGURE 2.4 – Diagramme d'état d'un *ButtonStateSM* (provenant du *Guide ButtonsController*[1])

La machine d'état reçoit l'évenement **evButtonPressed**. Une fois à l'état pressé, si le bouton est relaché avant 1 sec, elle notifie le *ButtonEventsHandler* d'un appui court, sinon d'un appui long.

1. RIEDER. et STERREN., « Button Manager - Laboratory Guide »

2.3 Application Layer

2.3.1 Liaison avec le Middleware

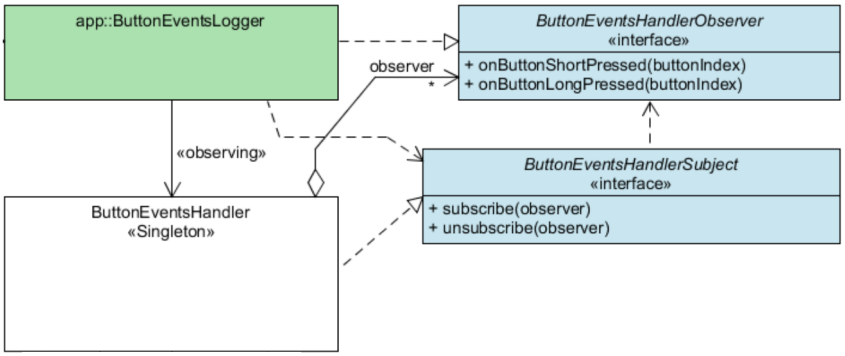


FIGURE 2.5 – Diagramme de classe de l'App (provenant du *Guide ButtonsController* [1])

Pour permettre d'ajouter plusieurs module a la couche *Application*, un patterne *Subject - Observer* a été utilisé. Le *ButtonEventsHandler* possède une liste d'*Observer*, a qui il va notifier les appuis longs ou court. Les *Observers* vont pouvoir **subscribe** au *Subjet* pour informer qu'ils souhaitent être notifié des évenement boutons.

2.3.2 Découplage complet du ButtonEventsLogger

Pour découpler "totalement" le *ButtonEventsLogger*, il a été demandé d'en faire une machine d'état.

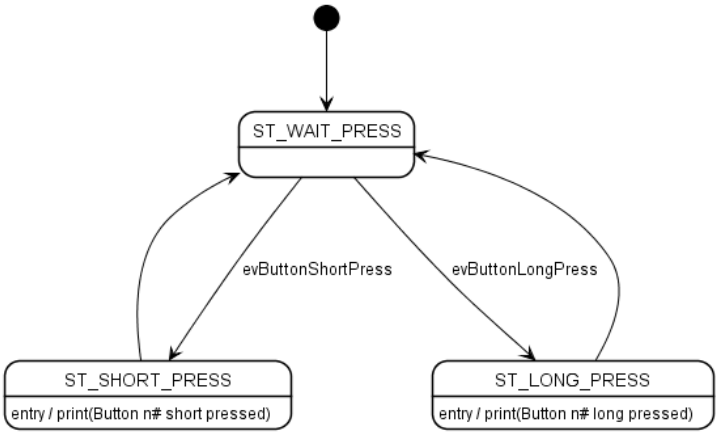


FIGURE 2.6 – Diagramme d'état du *ButtonEventsLogger*

1. RIEDER. et STERREN., « Button Manager - Laboratory Guide »

2.3.3 ButtonEventsLedFlasher

Le *ButtonEventsLedFlasher* transmet les évenement **evButtonShortPress** et **evButtonLongPress** à des machines d'états *LedStateSM* :

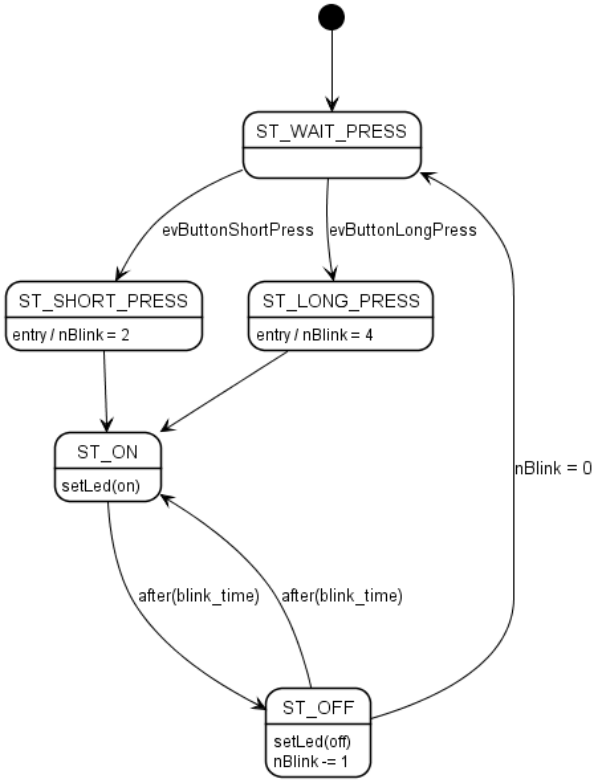


FIGURE 2.7 – Diagramme d'état d'une *LedStateSM*

Ces machines d'états appellent ensuite des méthode du *LedsController* fournit dans les modules de base.

Chapitre 3

Résultats et tests

3.1 Résultats

L'implémentation complète fonctionne parfaitement :

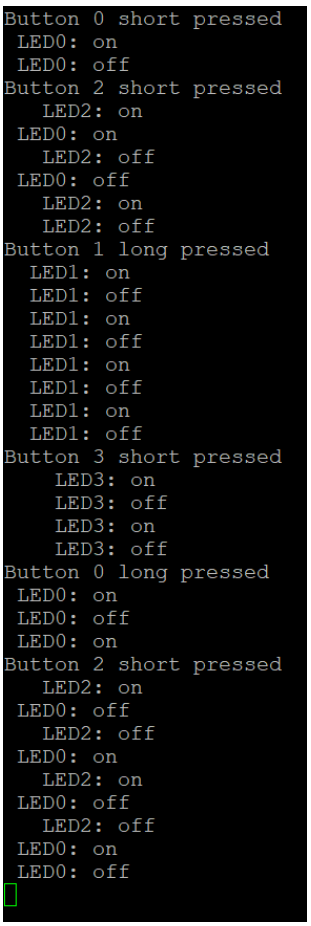


FIGURE 3.1 – Sortie série du programme

Le programme a subi plusieurs stressTests, ou les boutons ont été martelés le plus vite possible, laissés appuyé pendant plusieurs secondes, et rien n'a créer de bug dans l'execution.

Chapitre 4

Conclusion

4.1 Bilan

Le programme fonctionne parfaitement, toutes les méthodes de debounce, découplage,

4.2 Conclusion personnelle

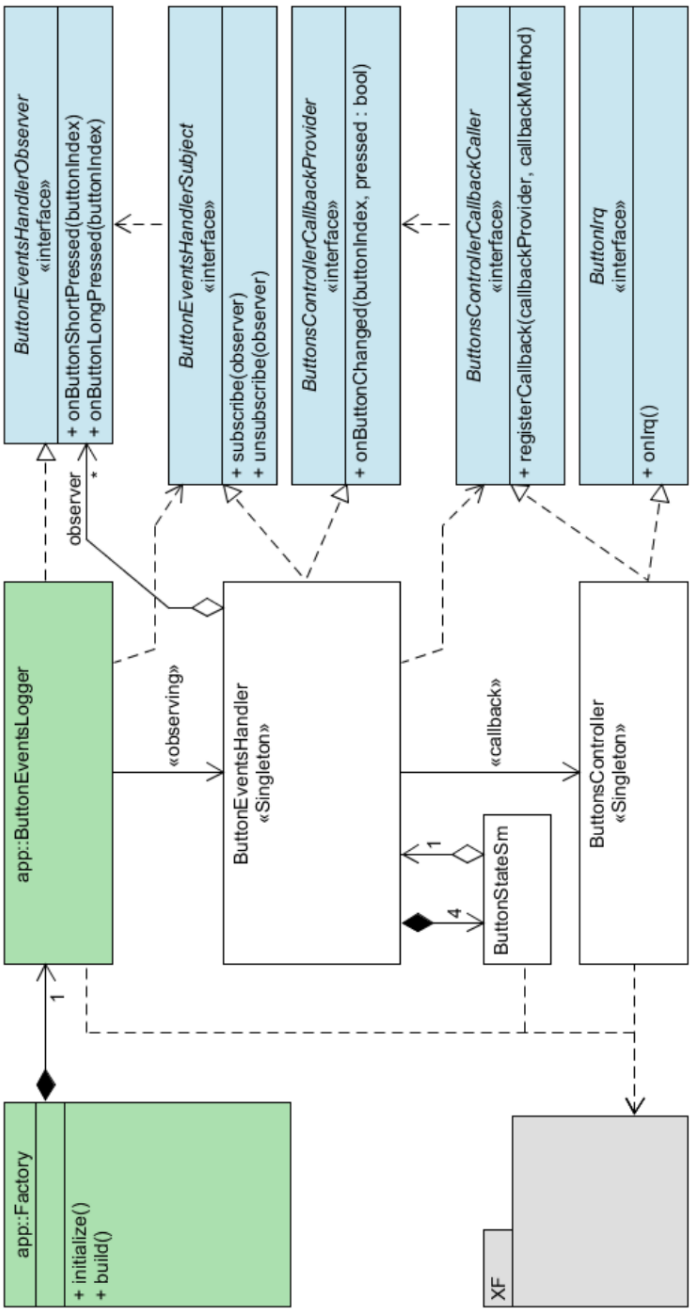
Ce projet a permis de se rendre compte de l'importance du découplage et des patternes. L'ajout du *ButtonEventsLedFlasher* a été extrêmement simple une fois l'intégralité du programme opérationel.

4.3 Signature

Samy Francelet

Annexe A

Diagrammes de classe



Bibliographie

RIEDER., Médard et Thomas STERREN. « Button Manager - Laboratory Guide ». In : (). URL : https://cyberlearn.hes-so.ch/mod/folder/view.php?id=1539125. (accessed : december 2021).