RAPPORT BE C++

Afficheur interactif de paramètres réglables via un écran LCD

BOURLOT Xavier, ZENNARO Thomas, 4IMACS-AE-SE-GROUPE 1, Binôme 6

30 mai 2020

Table des matières

Introduction

Ce bureau d'études a pour but de mettre en pratique nos connaissances acquises en conception et programmation orientée objet dans le langage C++ au travers de la réalisation d'une application pour le domaine des objets connectés. Ce projet vise, d'une part, à nous permettre de réaliser une bibliothèque modulaire de classes afin d'utiliser efficacement les capteurs et actionneurs à notre disposition. D'autre part, ce projet nous permet de réfléchir aux problématiques à prendre en compte lors de la création d'une application, à savoir la fiabilité, la robustesse ou bien son côté interactif.

1 Conception de notre bibliothèque modulaire

Concernant le déroulement du projet, nous avons d'abord réalisé la hiérarchisation des classes associées aux différents types de devices (capteurs/actionneurs). Par suite, nous avons réfléchi au modèle de notre application, en s'interrogeant sur les classes les plus intéressantes permettant de satisfaire notre idée : être capable de pouvoir afficher et modifier des valeurs de paramètres (Température...) ou activer des actionneurs (Allumer une LED...) via un écran LCD.

1.1 Classes spécifiques aux devices

Nous avons décidé de séparer en deux catégories aussi bien nos capteurs que nos actionneurs : analogiques et numériques. Il est important de faire cette distinction car leur comportement est différent, ce qui implique que l'héritage d'attributs ou de méthodes peut être différent.

— Capteurs analogiques : Capteur de température, de pression...
— Capteurs numériques : Bouton externe ON/OFF...
— Actionneurs numériques : LED...

Remarque : nous avons fait en sorte que l'écran LCD soit un actionneur indépendant par soucis de manipulation plus aisée.

Vous pourrez trouver ci-dessous le diagramme de classes associé :

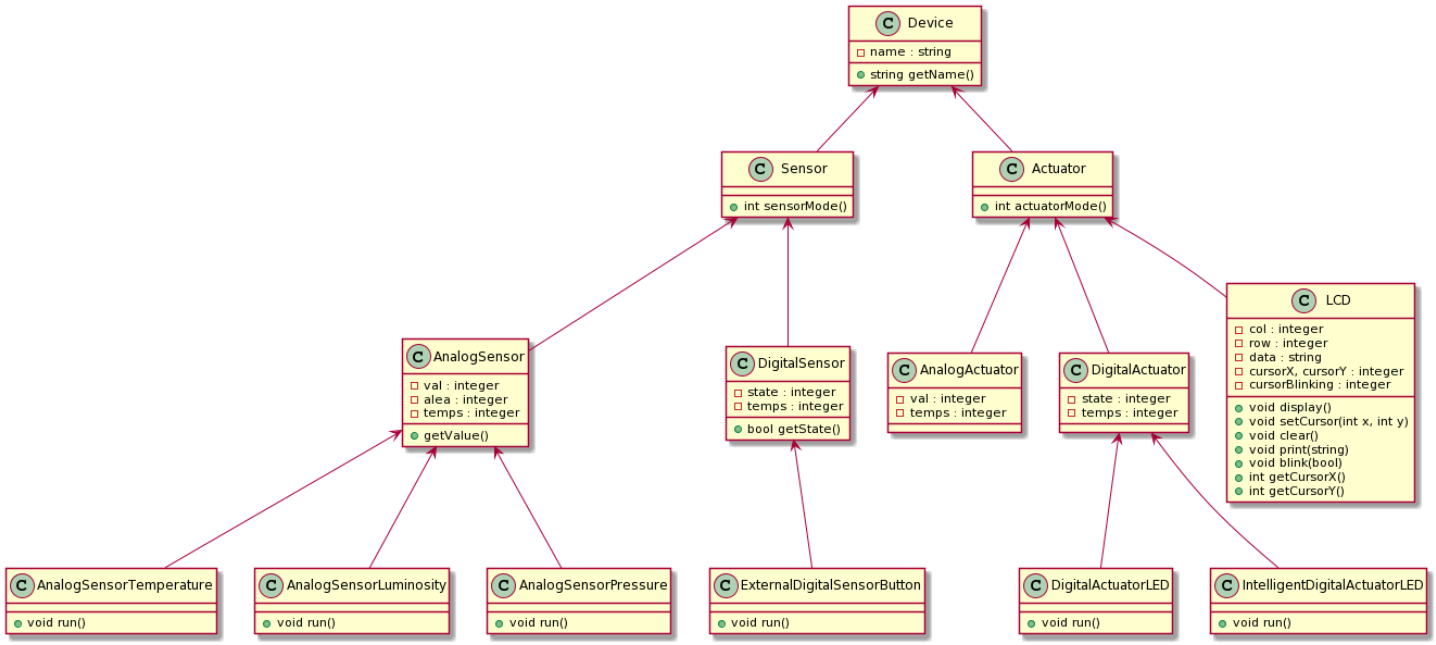

FIGURE 1 – Hiérarchisation des classes pour les capteurs et actionneurs

1.2 Classes spécifiques à l'architecture modulaire de notre application

Nous avons décider de décomposer notre architecture en différentes classes et sous-classes pour pouvoir manipuler facilement les données des devices et réaliser indépendamment tout le côté interactif lié à l'affichage des écrans. Voici notre diagramme de classes associé à cette architecture :

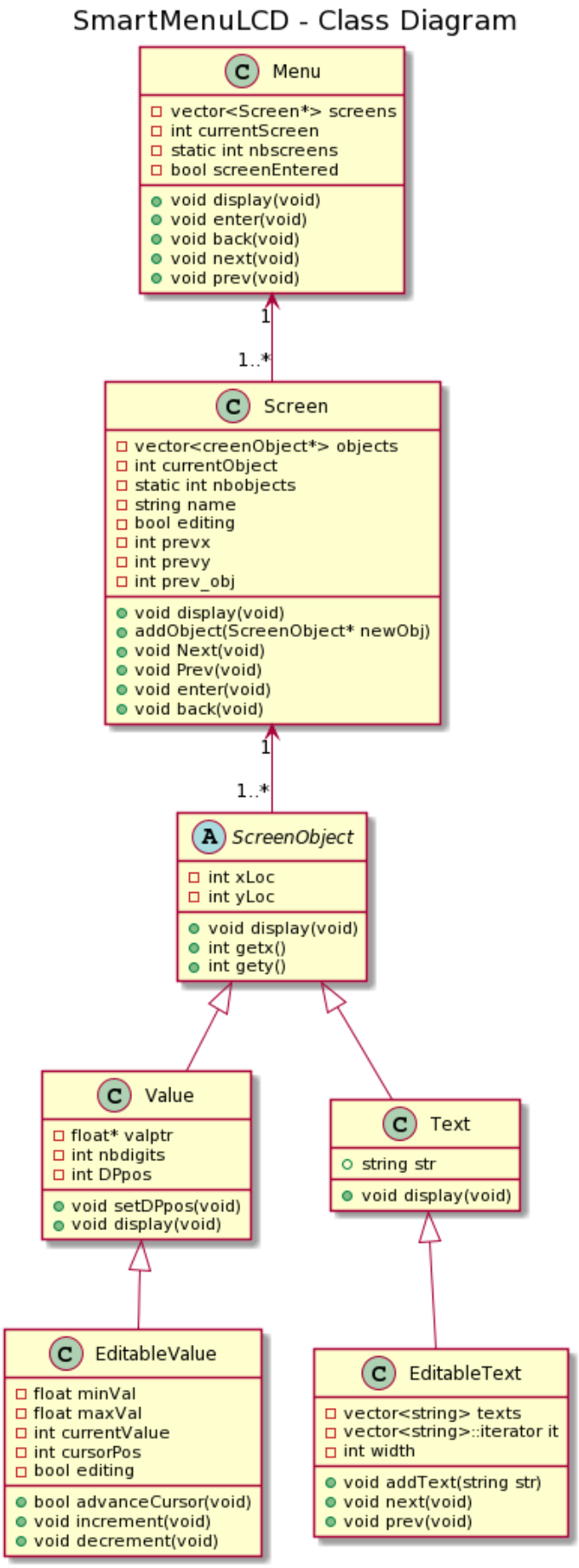

FIGURE 2 – Hiérarchisation des classes pour notre architecture de menu LCD interactif

1.2.1 Manipulation des données collectées via les devices

Les $Screens$ peuvent contenir des $ScreenObjects$, qui sont ici définis sous la forme de Valeurs ou de Texte. D'autres types d'objets peuvent êtres ajoutés facilement grâce à l'approche modulaire concernant leur manipulation (via les fonctions $Up/Down/Enter/Back$). Chaque objet peut aussi être éditable, soit dans le cas des textes : afficher un parmi plusieurs textes définis préalablement ; et pour des valeurs : modifier la valeur entre des bornes min et max, en se déplaçant de dizaine en dizaine.

1.2.2 Menu interactif pour l'affichage et la modification

Pour le menu, nous avons deux classes principales :
— **Screen** : cette classe permet d'afficher les dfférents objets de type ScreenObject, à savoir des petits textes et des valeurs. Egalement, elle va permettre de visualiser les objets éditables qui seront donc selectionnables par le curseur d'écran.
— **Menu** : cette classe est le centre de la navigation au sein de l'afficheur LCD. En effet, elle permet de dérouler les différents "Screens" possibles" et par conséquent de rentrer à l'intérieur de l'écran selectionné.

1.2.3 Classe LCD

Bien que ne faisant pas partie en soi de notre application, nous avons du développer une classe LCD fournissant un affichage à l'écran semblable à un afficheur réel. Sa dimension est réglable à l'initialisation. Pour la démonstration, nous avons utilisé la taille la plus commune, à savoir $16 * 2$. Cet affichage étant très réducteur sur la quantité d'informations disponibles à la fois, nous avons trouvé que c'était un bon exemple pour justifier la nécessité de notre sytèms de menus dans la lisibilité de l'information.
En ce qui concerne les fonctionnalités offertes par cette classe, nous avons suivi les fonctionnalités de base offertes par la librairies LCD Arduino, afin d'être au plus proche d'une application réelle. L'usage du curseur pour indiquer la position du chiffre actuellement édité dans une valeur éditable était par exemple nécessaire. Nous avons donc utilisé une syntaxe particulière basée sur des codes de contrôle Unicode afinc de pouvoir surligner la position du curseur (en rouge), et de le faire clignoter, comme sur un vrai écran LCD.
De plus, pour augmenter la lisiblité de l'affichage, au lieu de reimprimer à l'écran un nouvel afficheur à chaque modification et accès de notre librairie à l'écran LCD, nous avons manipulé le terminal pour remplacer l'écran précédent par le nouveau. Les deux commandes utilisées sont les suivantes :

```
//...//
cout << "\033[2J";  // efface l ecran et repositionne le curseur a l origine
//...//
cout << "\033["<<(cursorBlinking? '5' :'1')<<";41m"<<data[r*col+c]<<"\033[0m";
    //affiche un curseur rouge et/ou clignotant
```

2 Réalisation d'une application interactive via un afficheur LCD

2.1 Fonctionnement

Nous pouvons déceler 3 acteurs principaux lors de l'utilisation de notre application :
— **Utilisateur externe** :par l'intermédiaire d'un encodeur rotatif (simulé par le biais d'un clavier standard), l'utilisateur va pouvoir naviguer dans le Menu et au besoin, éditer des valeurs et ou des états (ON/OFF).
— **Arduino** : il permet de récupérer les valeurs des paramètres physiques renvoyés par les capteurs (connectés à certaines broches de l'Arduino) de manière périodique.
— **Ecran LCD** : va s'occuper de l'affichage des paramètres.

L'ensemble des interactions est exposé sur la Figure 3.

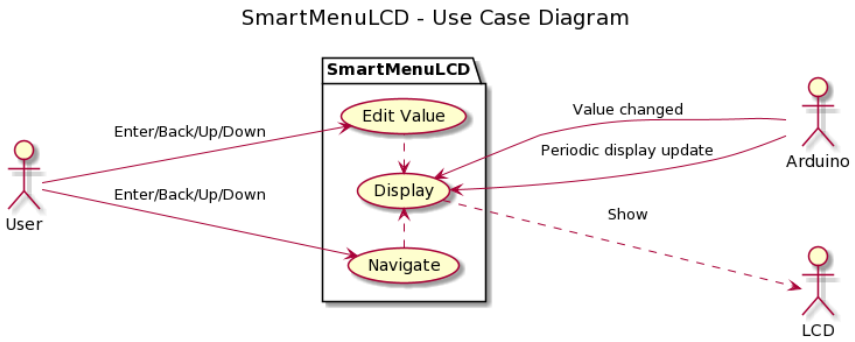

FIGURE 3 – Diagramme des cas d'utilisation de notre application

Concernant son fonctionnement, le programme principal va initialiser le Menu, puis se mettre en attente de la réception d'un des 4 caractères possibles sur l'entrée standard.

- **e** : va simuler un comportement de type "ENTER" de l'encodeur (appui sur l'encodeur) pour rentrer dans un écran ou se placer en mode édition d'un objet.
- **b** : va simuler un comportement de type "RETOUR", soit pour revenir au menu principal, soit pour sortir du mode d'édition d'objet. Il peut être implémenté physiquement par un bouton auxiliaire, ou une longue pression sur l'encodeur.
- **+** : va permettre, soit de passer au prochain nom d'écran du menu (visible via un curseur de type '>'), soit de se positionner sur le prochain objet éditable d'un écran, soit d'incrémenter une valeur numérique, soit de changer l'état de fonctionnement d'un appareil (ex : ENABLE/DISABLE...). Cette action correspond à une rotation horaire d'un cran de l'encodeur.
- **-** : va permettre, soit de passer au nom d'écran précédent (visible via un curseur de type '>'), soit de se positionner sur l'objet éditable précédent, soit de décrémenter une valeur numérique, soit de changer l'état de fonctionnement d'un appareil (ex : DISABLE/ENABLE...).Cette action correspond à une rotation anti-horaire d'un cran de l'encodeur.

Le point clef dans ce fonctionnement est que la gestion des menus internes et la modification des valeurs est entièrement gérée de manière interne par la librairie. Ainsi, le programme principal reste léger dans son utilisation du menu, et la complexité de l'affichage est cachée à l'utilisateur.

Le diagramme de séquence de la Figure 4 illustre un exemple fonctionnement. On observe bien que l'utilisateur (via l'encodeur), interagit uniquement avec le menu principal, et que celui-ci répercute de manière interne les actions à mener sur les différents composants (écrans, valeurs, afficheur).

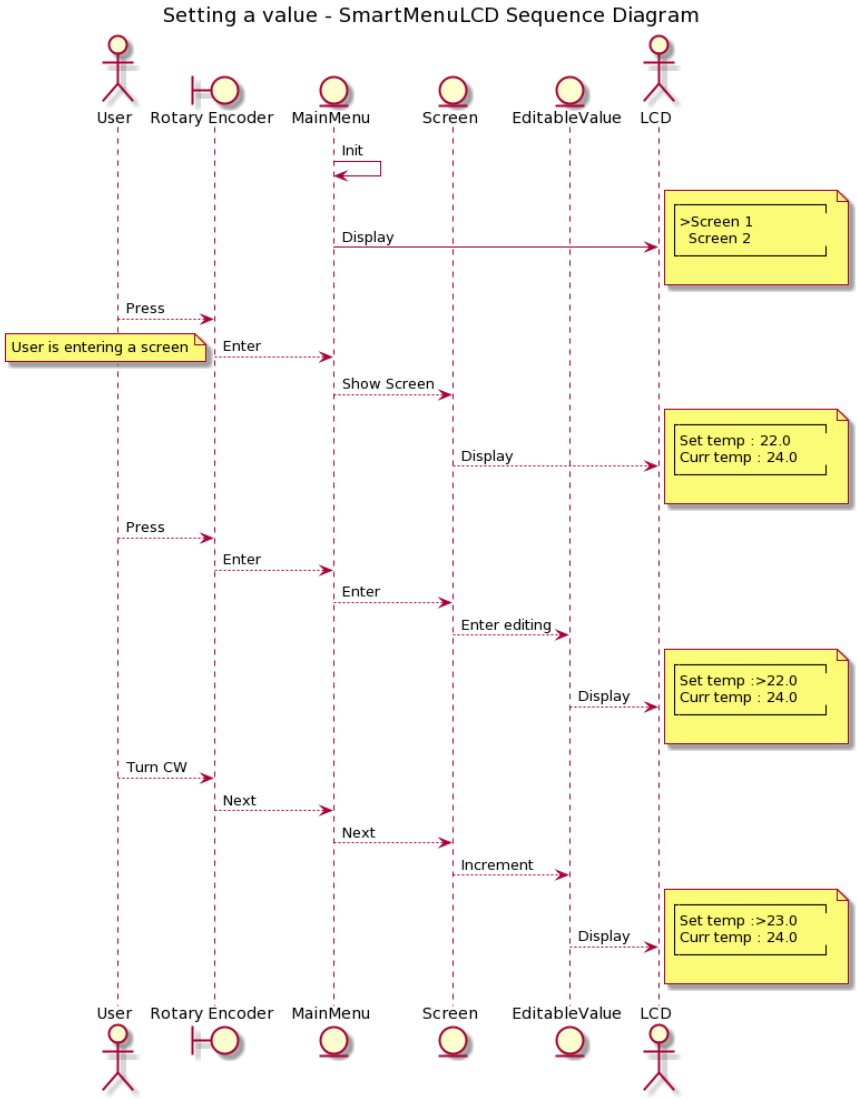

FIGURE 4 – Diagramme de séquence d'une utilisation possible de notre application

2.2 Résultats de tests

2.2.1 Tests relatifs à l'affichage

Dans un premier temps, nous avons testé l'affichage simple de valeurs et de texte en utilisant l'écran LCD drectement. Cle nous as permis d'améliorer le rendu du périphérique, ainsi que de déceler des problèmes liées à la taille de l'affichage.
Dans un second temps, les classes $Value$ et $Text$ ont été developpées et testées, en implémentant uniquement les fonctions les plus basiques. Cette démarche nous as permis d'éliminer rapidement un certain nombres d'erreurs et d'établir une base fonctionnelle pour les classes $EditableValue$ et $EditableText$, qui reposent sur ces dernières.
Une attention particulière a été portée pour contraindre les longueurs des champs (textes et valeurs), afin d'éviter tout problème de sur-écriture par la suite. Par exemple, le nombre de chiffres d'une valeur éditable est fixe, avec une résolution adaptée automatiquement, ce qui permet de passer d'un affichage de 9.99 à 10.0 sans débordement (en tenant compte du caractère virgule). Une amélioration de ces protections serait le lancement d'alertes à la compilation (via $assert$ par ex) afin de prévenir l'utilisateurs de ces sur-écritures liées à un mauvais positionnement des éléments entre eux.

2.2.2 Tests des objets éditables

Nous avons réalisé des tests unitaires et séquentiels (sans interaction) pour vérifier le bon fonctionnement du changement de text pour $EditableText$ par exemple, ainsi que pour la navigation par décades pour les valeurs. Cependant, l'absence de tests interactifs dans cette deuxième phase ne nous a pas permis de déceler l'intégralité des erreurs, qui ont été revelées plus tard.

2.2.3 Tests de Menu et Screen

En parallèle, les classes $Menu$ et $Screen$ ont été implementées. Une fois que la structure basique d'insertion d'objets dans leurs conteneurs respectifs a été developpée, la partie la plus ardue, à savoir l'aiguillage des commandes $Up/Down/Enter/Back$ aux sous-composants respectifs. En effet, le menu principal dois par exemple savoir que son écran actuel est sélectionné, et donc retransmettre l'instruction $Next$ à l'objet $Screen$ concerné.
La majorité des tests a donc concerné ces aspects de communication inter-classes, ainsi que des problèmes d'affichage liés à un mauvais séquencement des méthodes d'affichage de chacun des objets (sur-écritures).

2.2.4 Tests de l'application

Enfin, une application basique permettant de montrer une utilisation de ces focntionnalités a été developpée, afin de bien vérifier l'usabilité de la librairie. L'exemple choisi ets celui d'un contrôleur de température, avec une température consigne et un retour capteur de température, ainsi qu'un actionneur $Heater$ et d'autres capteurs divers.

Conclusion

Ainsi, nous avons pu, à travers le développement de cette application, appréhender de nombreux concepts spécifiques aux C++, comme l'héritage et la surchage, ou encore le polymorphisme (avec les $ScreenObjects$). La gestion du projet de A à Z, quoique un peu lourde à cause de l'écriture des périphériques, nous a permis d'aborder des aspects de conception, avec les différents diagrammes. Ce projet peut se révéler utile dans des applications pratiques, même si certaines améliorations peuvent encore être ajoutées. En particulier, pouvoir insérer des menus dans des menus permettrait d'élargir la navigation et d'augmenter les possiblités de l'interface. Une autre piste serait la création d'une interface de conception graphique de menus, qui génèrerait du code automatiquement, afin de faciliter la mise en place des menus et écrans.