Julien Kinderf
Guillaume Vincent **4AE-SE3**

Application du C++ au domaine des objets connectés

Conception et implémentation d'une alarme incendie

Table des matières

Julien Kinderf
Guillaume Vincent **4AE-SE3**

1 Introduction

Ce bureau d'étude s'installe dans le cadre des cours de programmation et de conception orientée objet. Il a pour objectif de nous faire réfléchir à une conception orientée objet en utilisant le langage C++. Nous devrons imaginer un problème à résoudre, qui s'articule autour d'une carte Arduino, de capteurs, et d'actionneurs. Ces équipements seront uniquement simulés, dans une simulation qui reprendra les grands principes de leurs fonctionnement.

Nous avons choisi de réaliser une alarme incendie automatique, s'appuyant sur des capteurs de fumée et de température. Nous allons donc présenter dans ce rapport les différentes étapes de conception de notre dispositif, exposer son fonctionnement, et analyser son implémentation logicielle.

2 Conception

2.1 Établissement des fonctionnalités

La première étape de la conception était de réfléchir aux fonctionnalités que nous allions donner à notre dispositif.

Nous avons décidé de fonder notre alarme sur deux capteurs principaux, un capteur de température, et un détecteur de fumée. Ces capteurs nous permettront de détecter la présence d'un incendie, et donc de faire retentir une alarme.

Nous avons également pensé à simuler la présence d'une batterie, dont il faudra gérer la charge. Si la batterie est trop faible, il faudra être capable d'en prévenir l'utilisateur, pour qu'il puisse la remplacer. Dans le cas contraire, le système cessera de fonctionner.

L'utilisateur pourra évidemment interagir avec le dispositif de différentes manières. Il pourra déclencher volontairement l'alarme, afin d'en vérifier le bon fonctionnement. Il devra aussi être capable d'arrêter l'alarme. L'utilisateur pourra choisir d'armer ou de désarmer l'alarme. Si l'alarme n'est pas armée, elle ne sonnera pas, même si un incendie se déclenche.
Au final, nous avons pu dresser une liste d'appareils qui seront connectés à notre Arduino. Cette liste est présentée sur la figure 1.

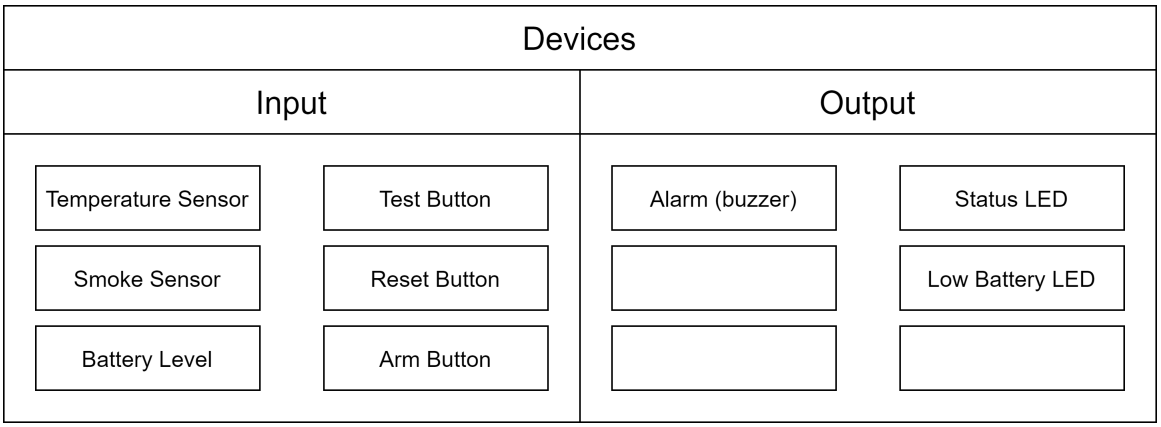

Figure 1 – Liste d'appareils connectés à la carte Arduino

2.2 Diagramme de cas d'utilisation

Après une brève étude du système, on peut dresser le diagramme de cas d'utilisation présenté sur la figure 2

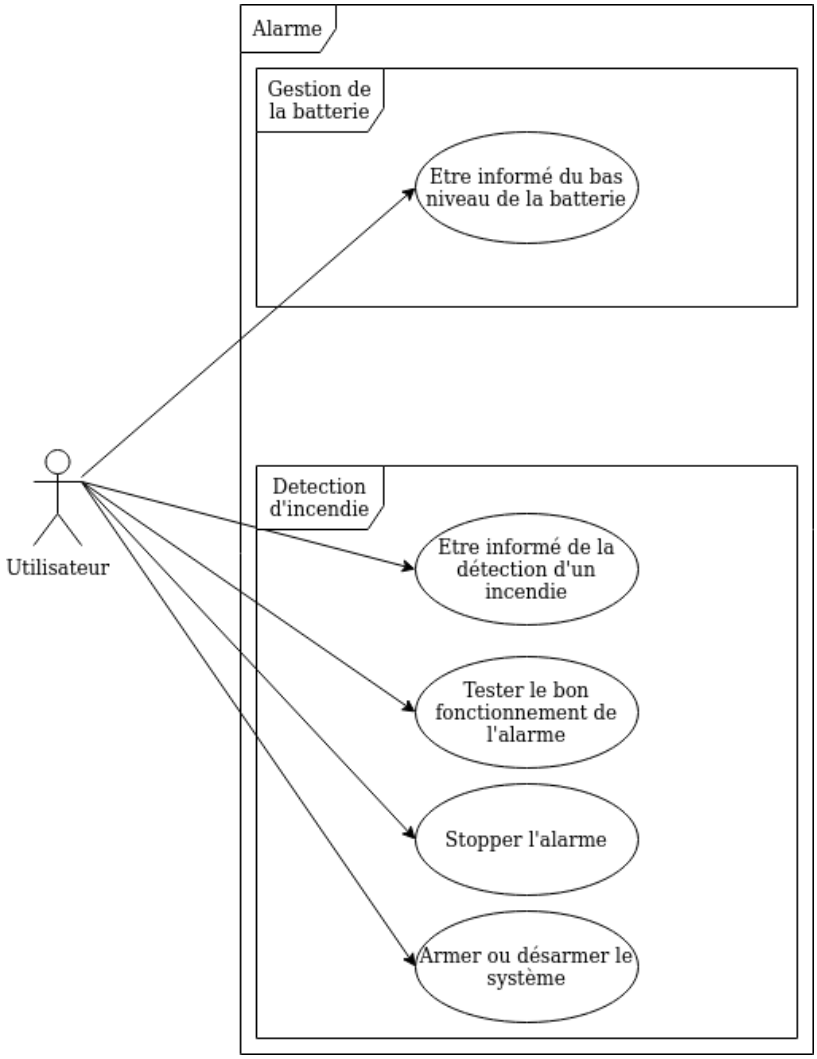

FIGURE 2 – Diagramme de cas d'utilisation associé à notre projet

2.3 Diagramme de classes

Pour créer et simuler les différents appareils que nous allons utiliser, nous nous appuierons sur des classes et des sous-classes qui sont présentées dans le diagramme de classes de la figure 3.

Les capteurs de température, de fumée, et de niveau de batterie seront définis en tant que *AnalogSensor*. Les trois boutons utilisés seront quant à eux des instances de la classe *DigitalSensor*. L'alarme sera bien évidemment un *Buzzer*. Toutes ces classes, avec la classe *LED*, découlent de la

classe *Device*. On a simplement ajouté une étape d'héritage pour différencier les capteurs (*Sensor*) en entrée des actionneurs (*DigitalActuator*) en sortie.

Les cartes Arduino disposant en temps normal d'un terminal, nous l'avons laissé dans l'architecture de notre projet, même si nous ne nous en sommes jamais servi.

On remarque enfin que les classes *Board* et *Device* peuvent respectivement lancer des exceptions *BoardException* et *DeviceException*.

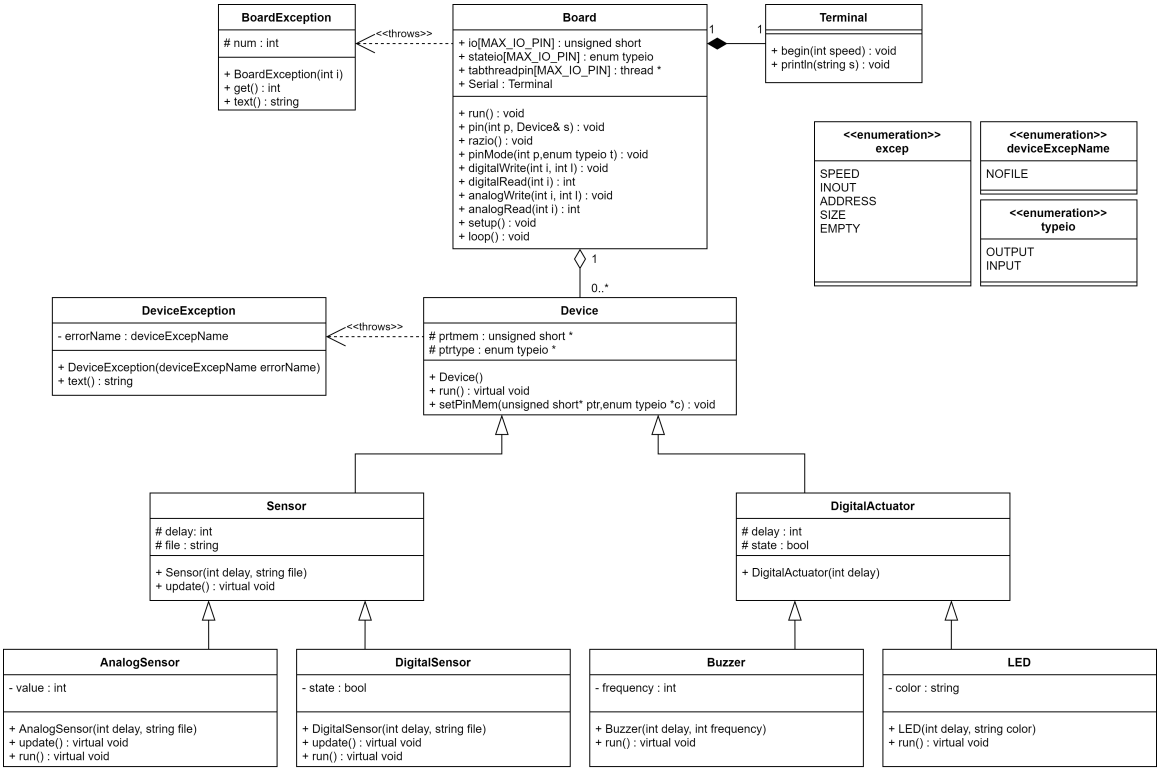

FIGURE 3 – Diagramme de classes de l'alarme incendie

3 Schéma de fonctionnement matériel et logiciel

4 Implémentation

Nous allons voir dans cette partie comment nous avons réussi à implémenter notre dispositif et à le faire communiquer avec son environnement.

4.1 Simulation des capteurs et des boutons

Commençons tout d'abord par la simulation des capteurs. Comme vu précédemment avec le diagramme de classes (figure 3), il en existe de deux types. On trouve dans un premier lieu les capteurs analogiques, pouvant lire des valeur entières, dont on se sert pour définir les capteurs de

température, de fumée, et de niveau de batterie. Il existe également des capteurs numériques, qui instancient nos boutons, et qui peuvent seulement prendre deux valeurs, *True* ou *False*.

Tous ces capteurs fonctionnent globalement de la même manière. Lorsque l'on définit un objet de classe *DigitalSensor* (un bouton), on lui donne en paramètre de son constructeur le nom d'un fichier. On regardera ensuite, lorsque l'on voudra mettre à jour l'état du bouton, si ce fichier est présent ou non. S'il n'est pas présent, on considérera que le bouton est relâché. A l'inverse, s'il est présent, le bouton sera considéré appuyé.

Les capteurs analogiques disposent du même système de fichiers. Cependant, en plus de vérifier que ces fichiers existent bien, on ira lire leur contenu, supposé représenter une valeur entière, pour mettre à jour nos capteurs.

Au niveau des sorties, que ce soit pour l'alarme ou les LEDs, tous les actionneurs disposent d'un état qui leur est propre. Selon la valeur de cet état, l'actionneur sera allumé ou éteint. La seule différence logicielle entre les l'alarme et les LEDs réside dans leur constructeurs : l'alarme est définie par une fréquence sonore, tandis que les LEDs sont définies par leur couleur. On a attribué arbitrairement une valeur de 500Hz à l'alarme, une couleur rouge pour la LED indicatrice d'état, et une couleur orange pour la LED signalant un faible niveau de batterie.

4.2 Gestion des exceptions

En plus des exceptions déjà existantes dans le projet d'origine, nous avons voulu rajouter une exception qui permettrait de notifier l'utilisateur dans le cas où l'un des fichiers sensés accueillir les données d'environnement (pour les capteurs analogiques) n'existait pas. Cette exception sert uniquement à la simulation, et ne serait pas présente dans le cadre de l'implémentation sur une vraie carte Arduino. Cette exception est une instance de la classe *DeviceException*.

Si jamais ces fichiers sont bien présents, mais que leur contenu n'est pas un entier, nous capturons également une exception et en informons l'utilisateur.

4.3 Utilisation de la STL

Nous avons utilisé la STL, visible dans notre fichier *board.cpp*. Nous avons créé un vecteur dans lequel nous rangeons tous les appareils que nous avons créé. Pour les configurer, nous avons alors seulement à parcourir ce vecteur dans une boucle *for*, et ainsi nous épargner quelques lignes supplémentaires.

4.4 Outils de simulations annexes

Pour simplifier l'utilisation de notre alarme simulée, nous avons décidé de nous aider de deux scripts python. L'utilisation de ces derniers est décrite dans le README associé au projet.

- Le premier permet de modifier automatiquement les variables environnementales. Le but est de simuler un incendie, et de modifier en temps réel les valeurs des fichiers lus par les capteurs de l'alarme afin qu'elles soient en accord avec la simulation de notre incendie.
- Le second sert de raccourci pour appuyer sur les boutons. Au lieu de devoir manuellement créer les fichiers correspondant aux pressions sur les boutons, nous avons implémenté une petite interface graphique qui le fait pour nous. Ce dernier script necessite le module python tkinter pour pouvoir fonctionner.

5 Conclusion

Pour conclure, on peut dire que le projet a été terminé et est désormais fonctionnel. Le fait qu'il aie du se faire à distance et donc totalement en simulation a été en fin de compte très intéressant du point de vue des nouvelles possibilités que des problèmes matériels auraient empêché de réaliser. Le distanciel n'a pas été un vrai problème pour nous pour ce projet, et nous sommes plutôt contents du résultat.