Université de Technologie de
Belfort-Montbéliard

Programmation Orientée Objet : Concepts fondamentaux et mise en pratique avec le langage C++

AP4A

Rapport de projet

Quentin Balezeau

Table des matières

1 Présentation

Ce projet constitue un simulateur dédié à la modélisation d'un écosystème IoT axé sur la surveillance de la qualité de l'air. Il est structuré en plusieurs classes distinctes, notamment la classe Server, Sensor et Scheduler.

L'objectif fondamental de ce projet est de mettre en pratique l'ensemble des concepts abordés en cours de programmation orientée objet, et son implémentation est réalisée en C++.

Le fonctionnement du projet s'articule de la manière suivante :

Une variété de capteurs est impliquée dans le processus, incluant ceux dédiés à la lumière, à la température, à l'humidité et au son. Le Scheduler rassemble les données provenant de ces capteurs pour les acheminer vers le serveur. Ce dernier, équipé d'un algorithme intégré, orchestre l'envoi des données des capteurs à des intervalles de temps diversifiés.

2 UML du projet

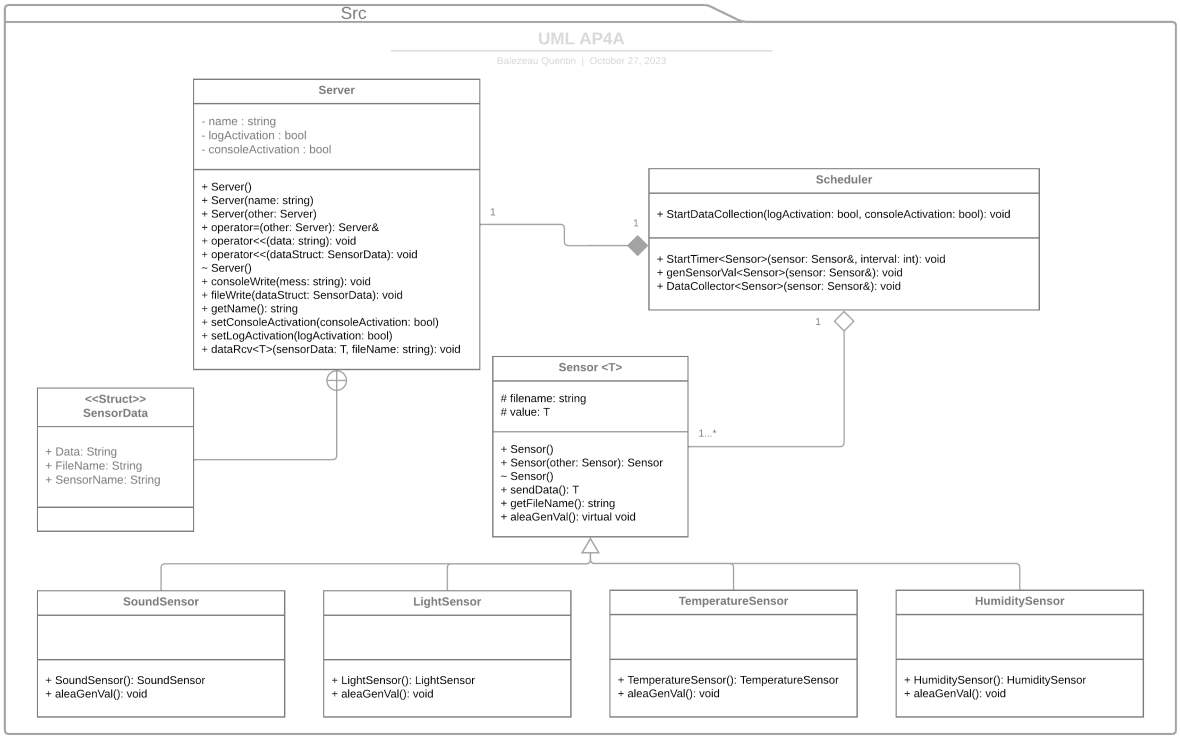

Figure 1 – UML du projet

3 La classe Server

La classe Server, comme évoqué précédemment, a pour mission de récupérer les données des capteurs, les imprimer dans la console et les stocker dans des fichiers "logs". Maintenant, plongeons dans les détails de cette classe.

La classe Server est dotée de plusieurs méthodes et fonctionnalités. Elle possède un constructeur par défaut, un constructeur prenant un nom en paramètre, un constructeur de copie, et un opérateur d'assignation. Le destructeur est également présent pour gérer la libération des ressources.

La méthode consoleWrite permet d'afficher les données dans la console, tandis que la méthode fileWrite enregistre les données dans des fichiers. Le chemin du fichier est défini en fonction du chemin actuel du programme et du dossier de sortie spécifié. La méthode d'opérateur ¡¡ est surchargée pour permettre l'écriture de données dans les fichiers de manière plus intuitive.

De plus, la classe propose deux méthodes pour activer ou désactiver l'impression dans la console et l'écriture dans les fichiers logs.

En résumé, la classe Server agit comme un gestionnaire central des données des capteurs, fournissant des méthodes pour afficher et enregistrer ces données selon les besoins spécifiques du projet.

4 La classe Sensor

Comme précédemment évoqué, le projet comprend plusieurs types de capteurs, dont les capteurs de température, de lumière, d'humidité et de son. Ces capteurs sont représentés par la classe Sensor et ses classes dérivées. Explorons davantage ces composants.

La classe Sensor est une classe abstraite qui fournit une structure de base pour tous les capteurs. Elle contient des méthodes pour obtenir la valeur du capteur et le nom de fichier associé. Les classes dérivées, telles que TemperatureSensor, LightSensor, HumiditySensor, et SoundSensor, implémentent ces méthodes de manière spécifique à chaque type de capteur.

Chaque classe dérivée de Sensor a une méthode aleaGenVal qui génère une valeur aléatoire correspondant aux propriétés du capteur. Par exemple, la classe TemperatureSensor génère une valeur de type float dans une plage spécifique.

En résumé, la hiérarchie des classes Sensor offre une abstraction cohérente pour les différents types de capteurs du projet, avec des méthodes spécifiques pour générer des valeurs aléatoires en fonction des caractéristiques de chaque capteur.

5 La classe Scheduler

La classe Scheduler s'intègre harmonieusement dans le fonctionnement global du projet. Reprenons les éléments du début du texte et examinons de plus près les détails de cette classe.

Comme mentionné précédemment, la classe Server récupère les données via la classe Scheduler, qui, à l'aide d'un algorithme intégré, envoie périodiquement les données des capteurs au serveur. La classe Scheduler est cruciale dans ce processus.

La classe Scheduler est constituée de plusieurs méthodes et fonctionnalités. La méthode StartDataCollection initialise la collecte de données en activant ou désactivant l'impression dans la console et l'écriture dans les fichiers logs, puis lance des threads pour chaque capteur avec des intervalles de temps spécifiques.

Les méthodes StartTimer, genSensorVal, et DataCollector sont des éléments clés de cette classe. StartTimer démarre un thread pour un capteur donné avec un intervalle spécifique. La méthode genSensorVal génère une valeur aléatoire pour le capteur, et DataCollector utilise cette valeur pour collecter et transmettre les données au serveur via la classe Server.

En résumé, la classe Scheduler orchestre la collecte de données des capteurs en utilisant des threads et des intervalles de temps définis. Elle interagit étroitement avec la classe Server pour assurer la réception et le traitement appropriés des données des capteurs.