Documentation du diagramme UML de l'application Qt pour ruche connectée

Créateur **Benjamin Bachelard** Créé **Jan 27, 2025, 14:22** Dernière mise à jour **Jan 27, 2025, 15:02**

Introduction

Ce document décrit la structure et le fonctionnement d'une application Qt conçue pour gérer une ruche connectée. L'architecture est représentée sous forme de diagramme UML, mettant en évidence les principales classes, leurs attributs, méthodes, et les relations entre elles.

1. Classes principales

1.1 MainWindow

- **Responsabilité** : Gère l'interface utilisateur de l'application et coordonne les interactions entre les autres classes.
- **Attributs et méthodes** :
  - `+ displayData()` : Affiche les données collectées sur l'interface utilisateur.
  - `+ startAcquisition()` : Lance le processus de collecte des données des capteurs.
  - `+ exportData()` : Permet l'exportation des données collectées.

1.2 Acquisition

- **Responsabilité** : Collecte les données des capteurs connectés (température, humidité, masse, IR, etc.).
- **Attributs** :
  - `- temperature: float` : Température mesurée par le capteur.
  - `- humidity: float` : Humidité mesurée par le capteur.
  - `- mass: float` : Masse mesurée par la balance.
  - `- irData: array` : Données infrarouges collectées.
  - `- i2cInterface: QSerialBus` (pour les capteurs DHT22 et AMG8833 via I2C).
  - `- spiInterface: QIODevice` (pour le capteur HX711 via SPI).
  - `- csiInterface: QImage` (pour les caméras IR et standard via CSI).
  - `- irMatrix: array` (pour les températures globales de la matrice IR).
  - `- standardImage: Image` (pour les images standards capturées).
- **Méthodes** :
  - `+ startSensors()` : Initialise les capteurs et démarre la collecte des données.
  - `+ captureImage()` : Capture une image à l'aide d'une caméra connectée.
  - `+ getData(): dict` : Retourne les données collectées sous forme de dictionnaire.
  - `initializeI2C()` : Configure l'interface I2C pour les capteurs.
  - `initializeSPI()` : Configure l'interface SPI pour le capteur de masse.

- `captureIRImage()` : Capture une image infrarouge.
- `captureStandardImage()` : Capture une image standard.

1.3 DatabaseManager

- **Responsabilité** : Gère la sauvegarde et la récupération des données dans une base de données.
- **Attributs** :
  - `- connection: string` : Paramètres de connexion à la base de données.
- **Méthodes** :
  - `+ connectDB()` : Établit la connexion à la base de données.
  - `+ saveData(data: dict)` : Sauvegarde les données collectées dans la base de données.
  - `+ retrieveData(): dict` : Récupère les données sauvegardées.

1.4 ImageProcessor

- **Responsabilité** : Analyse les images capturées pour détecter des éléments spécifiques (comme les abeilles).
- **Méthodes** :
  - `+ processImage(image: Image): ProcessedImage` : Analyse une image et applique un traitement.
  - `+ detectBees(image: Image): int` : Détecte le nombre d'abeilles présentes sur une image.

1.5 DataExporter

- **Responsabilité** : Gère l'exportation des données vers différents supports.
- **Méthodes** :
  - `+ exportToSD(data: dict)` : Exporte les données vers une carte SD.
  - `+ exportToLoRaWAN(data: dict)` : Envoie les données via une connexion LoRaWAN.

2. Relations entre les classes

2.1 MainWindow et Acquisition

- **Relation** : Association.
- **Description** : La classe `MainWindow` utilise `Acquisition` pour lancer la collecte des données des capteurs.

2.2 MainWindow et DatabaseManager

- **Relation** : Association.
- **Description** : `MainWindow` interagit avec `DatabaseManager` pour sauvegarder ou récupérer les données collectées.

2.3 MainWindow et DataExporter

- **Relation** : Association.
- **Description** : `MainWindow` utilise `DataExporter` pour gérer l'exportation des données vers différents supports.

2.4 Acquisition et DatabaseManager

- **Relation** : Agrégation.
- **Description** : `Acquisition` envoie les données collectées à `DatabaseManager` pour les sauvegarder.

2.5 Acquisition et ImageProcessor

- **Relation** : Association.
- **Description** : `Acquisition` fait appel à `ImageProcessor` pour analyser les images capturées par les capteurs.

2.6 DatabaseManager et DataExporter

- **Relation** : Association.
- **Description** : `DatabaseManager` transmet les données sauvegardées à `DataExporter` pour leur exportation.

3. Points forts de l'architecture

- **Simplicité** : Chaque classe a une responsabilité claire, ce qui facilite la compréhension et la maintenance du code.
- **Modularité** : Les classes sont découplées, permettant une évolution indépendante des différentes fonctionnalités.
- **Extensibilité** : De nouvelles fonctionnalités (capteurs, méthodes d'exportation, etc.) peuvent être ajoutées sans impact majeur sur l'architecture existante.
- **Réutilisabilité** : Certaines classes, comme `DatabaseManager` et `DataExporter`, peuvent être réutilisées dans d'autres projets similaires.

4. Conclusion

Ce diagramme UML offre une vue d'ensemble claire et structurée de l'application Qt pour ruche connectée. Il permet aux développeurs de comprendre rapidement les interactions entre les différentes composantes, d'améliorer la collaboration dans l'équipe et de guider le développement de l'application en respectant les principes de conception orientée objet.