Modelleren UML

Beschrijving van de klassen voor de SenseBox

Luc Appelman (640696@student.inholland.nl)

6 april 2022

Inhoudsopgave

1 Inleiding

In dit document wordt de samenhanging van de klassen beschreven, deze klassen zijn gemodelleerd voor het SenseBox project. Er worden in totaal drie cruciale onderdelen van de SenseBox beschreven:

- Abstractie laag voor de sensoren
- WiFi verbinding
- Logsysteem

Het doel van de SenseBox is om iedere minuut data van verschillende sensoren uit te lezen en deze te versturen naar een IoT (Internet of Things) Platform waarop de data uitgelezen kan worden en opgeslagen wordt in een lokale database. Om de gemeten data te versturen wordt een WiFi verbinding gebruikt. Voor het project wordt er gebruik gemaakt van een RTOS (Real Time

OS), FreeRTOS met ESP-IDF. De UML diagrammen zijn ook met de bestaande structuur van dit OS in gedachten gemaakt.

1.1 Schrijfwijze

Om de resulterende code zo overzichtelijk mogelijk te houden hebben we als team een aantal regels ingesteld hoe we variabelen, klassen etc. kunnen onderscheiden. Om eventuele verwarring te voorkomen, voor bijvoorbeeld een volgend team dat aan het project gaat werken, hebben we besloten om deze regels ook toe te passen in het UML ontwerp van de code.

- In klassen krijgen alle member functies en properties een *m* als voorvoegsel. Als het een private member of property is krijgt deze *m_* als voervoegsel. Andere voervoegsels komen na de *m* (maar voor het liggende streepje als het om een private property gaat).
- Iedere variabele start met een hoofdletter, tenzij er geen voorvoegsel is.
- Argument van een functie: een argument start altijd met een *a*.
- Pointer: de variabele krijgt een *p* als voervoegsel.
- Reference: de variabele krijgt een *r* als voorvoegsel.
- Klasse: de variabele krijgt een hoofdletter *C* als voorvoegsel.
- Enum/Struct/Datatype: Gebruikt pascal casing

2 Modellen

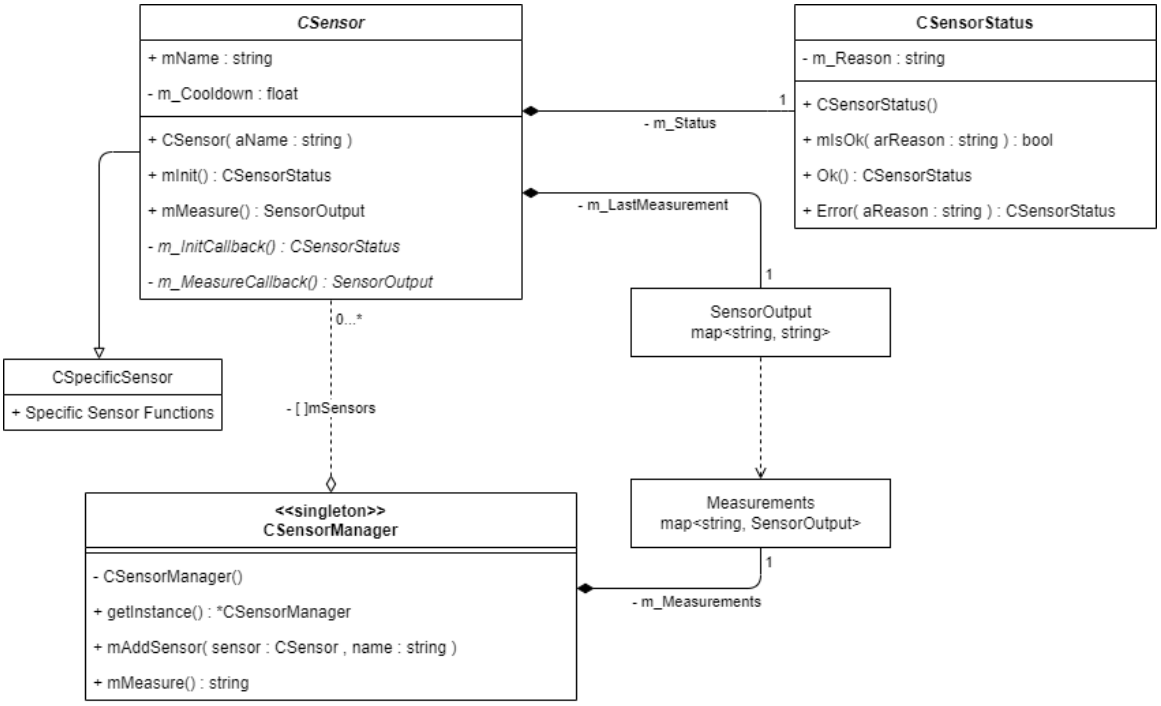
2.1 Sensoren

2.1.1 Klassendiagram

Het doel van de klassen die de sensoren uitlezen is om een structuur op te zetten waarbij zo gemakkelijk mogelijk nieuwe sensoren vervangen/toegevoegd kunnen worden. Ook is het van belang dat van buiten af voor het meten van alle sensoren maar één functie aangeroepen hoeft worden. Hierom is er voor gekozen om de zogeheten CSensorManager aan te maken. Dit is een singleton omdat er altijd maar één manager aangemaakt mag worden in het programma. Aan deze manager kunnen een x aantal sensoren toegevoegd worden. Wanneer de *mMeasure()*

member aangeroepen wordt zal iedere sensor uitgemeten worden. Het resultaat van iedere sensor is beschreven in een SensorOutput map. Deze worden gecombineerd in het Measurements datatype, dit wordt door de *mMeasure()* functie. Om ieder type sensor te ondersteunen is er een

CSensor klasse aangemaakt die de minimale functionaliteit van iedere sensor beschrijft, een daadwerkelijke sensor klasse erft deze klasse en kan toegevoegd aan de sensor manager. De *m_InitCallback()* en *m_MeasureCallback()* members zijn virtual en worden gebruikt om de sensor specifieke functies van een sensor te implementeren. Iedere *CSensor* heeft ook een *m_Status* property van het type *CSensorStatus* waarin de status van de sensor opgeslagen is.

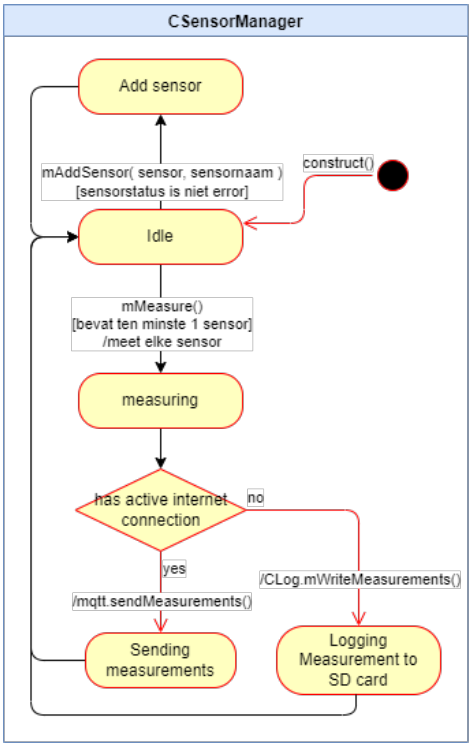


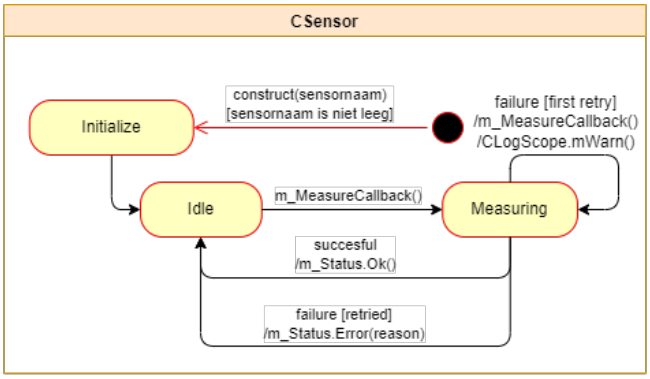
2.1.2 Toestandsdiagram

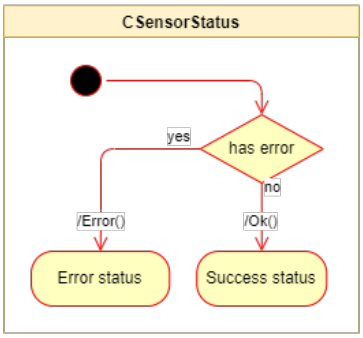
De klassen hebben een aantal states waarin de klasse kan verblijven. Over de grootste tijd zullen ze in de idle state staan omdat we slechts één keer per minuut meten. Het meten kan alleen gedaan worden als er al een sensor toegevoegd is aan de sensor manager. Het initialiseren van een sensor gebeurt buiten de sensor manager, een sensor mag alleen toegevoegd worden als de status van de status geen error is. Het meten van een sensor mag één keer fout gaan, hierdoor

wordt er ook een warning in de log opgeschreven. Als hierna het meten van de sensor weer niet goed gaat zal er een error opgeslagen worden in de sensor klasse. Deze status zal later uitgelezen en gelogd moeten worden voordat de metingen opgestuurd worden naar het IoT platform. Het

opslaan van de gemeten waarden kan op twee manieren. Bij voorkeur gaat dit via een bericht met alle getemeten waarden over het internet zodat de waarden direct in het IoT platform staan. Mocht de SenseBox geen actieve internet verbinding hebben wordt de data alsnog in een log bestand op de interne SD kaart opgeslagen zodat deze informatie later alsnog uitgelezen kan worden.



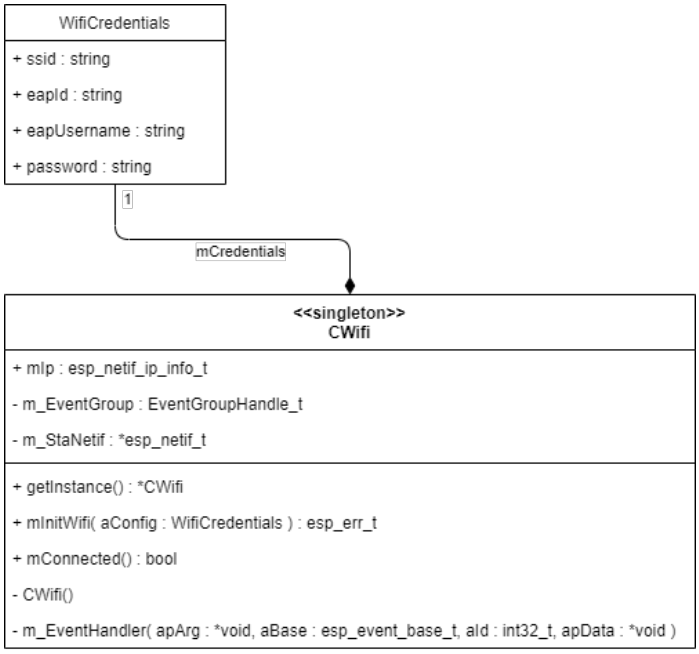




2.2 WiFi verbinding

2.2.1 Klassendiagram

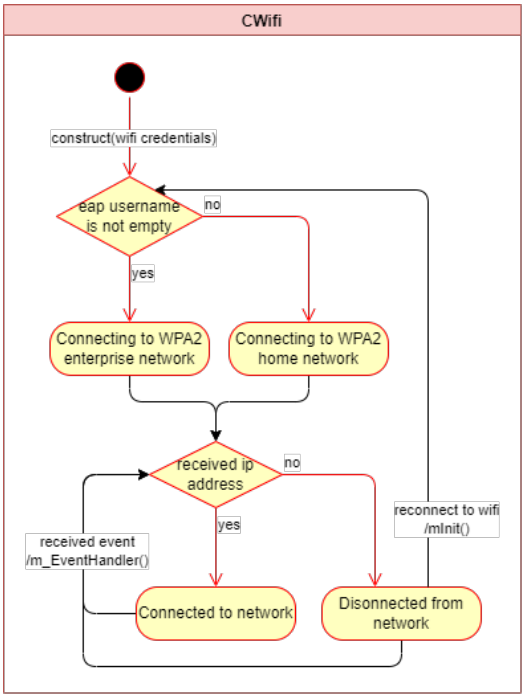
Om de verzamelde informatie te versturen wordt er gebruik gemaakt van een WiFi verbinding die aangemaakt wordt wanneer de SenseBox opstart. Dit is ook een singleton omdat er maar één wifi antenne is en er dus maar met één netwerk tegelijk verbinding gemaakt kan worden. Het ip address wordt opgeslagen in de klasse zodat deze later opgevraagd kan worden buiten de klasse. Bij het aanmaken van een WiFi verbinding moeten de inloggegevens voor het netwerk meegegeven worden in de constructor.



2.2.2 Toestandsdiagram

Wanneer de WiFi verbinding gestart wordt moeten de inloggegevens meegegeven worden met de constructor. Als de EAP gebruikersnaam hierin niet leeg is weten we dat we met een WPA2 Enterprise netwerk willen verbinden, dit is nodig om met het netwerk op school te kunnen verbinden. Als de gebruikersnaam leeg is moet er verbonden worden met een normaal WPA2 netwerk zoals je dat thuis hebt. Vanuit de WiFi verbinding kunnen er verschillende events gegenereerd worden die opgevangen moeten worden door de *m_EventHandler* member functie. Als dit gebeurt wordt de status van de verbinding bijgewerkt. Omdat er gebruik gemaakt wordt

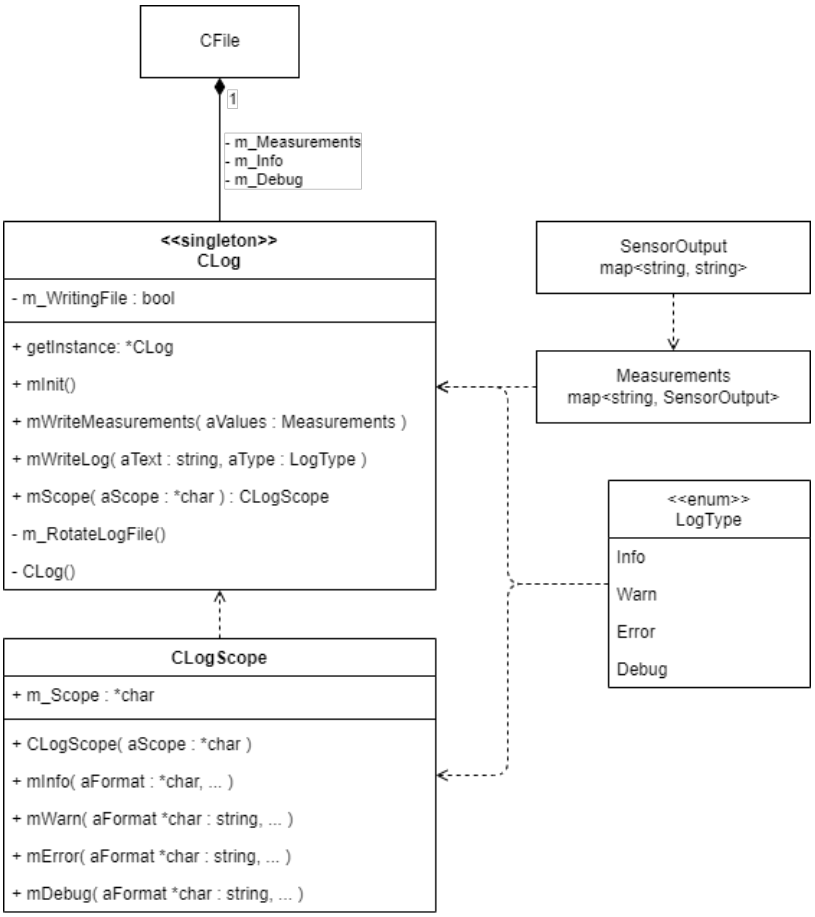
van FreeRTOS kunnen andere functies van FreeRTOS direct gebruik maken van de WiFi verbinding zonder dat hiervoor een speciale write of read functie aangemaakt moet worden. Dit is dan ook niet opgenomen in de toestandsdiagram.



2.3 Log systeem

2.3.1 Klassendiagram

Om informatie over het programma op te slaan terwijl er metingen etc. gedaan worden kan er informatie naar één van de drie log bestanden geschreven worden. Er worden in totaal drie logbestanden bijgehouden: *m_Measurements*, *m_Log* en *m_Debug*. Er is een enum voor het type log dat we willen schrijven, in de implementatie zal hiervoor dan een herkenbare tag als prefix gezet worden voor de log entry. Er wordt ook gebruik gemaakt van een log scope, deze wordt gebruikt om aan te geven welk onderdeel van het programma iets naar het log geschreven heeft. Een scope zal een zelfde soort tag krijgen als het logtype.



2.3.2 Toestandsdiagram

Een nieuwe regel kan aan het log toegevoegd worden via twee verschillende functies. *mWriteMeasurements()* en *mWriteLog()*, de eerste hiervan wordt gebruikt om naar het log bestand in *m_Measurements* te schrijven wanneer er geen actieve internet verbinding is. De tweede kan zowel naar *m_Log* en *m_Debug* schrijven, afhankelijk van het *LogType* dat wordt meegegeven in de functie. De *LogType* wordt bepaald door de functie die wordt aangeroepen in *CLogScope*. Wanneer

er geen SD kaart in de SenseBox zit wordt dit verwerkt in de *CFile* klasse en zal dit niet zorgen voor problemen in de *CLog* klasse.