Entwicklung einer Hardware-Health-Monitoring Lösung für Pepperl+Fuchs HMI Systeme

Bachelorarbeit

des Studienganges Elektrotechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

von
Vitali Mostowoj

018.09.2023

Bearbeitungszeitraum:	12 Wochen
Matrikelnummer, Kurs:	9960312, Tel20 At1
Ausbildungsfirma, Abteilung:	Pepperl + Fuchs SE, HMI
Standort:	Lilienthalstraße 200, 68307 Mannheim
Betreuer der Ausbildungsfirma:	Dr. Marc Seissler
Gutachter der Dualen Hochschule:	Prof. Dr. Joachim Priesnitz

Unterschrift (Betreuer)

Todo list

Sperrvermerk

„Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung des Dualen Partners vorliegt.“ [Ende der Sperrfrist: 31.12.2222]

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: Entwicklung einer Hardware-Health-Monitoring Lösung für Pepperl+Fuchs HMI Systeme selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Abstract

Inhaltsverzeichnis

Inhaltsverzeichnis **III**

Abbildungsverzeichnis **V**

Tabellenverzeichnis **VI**

1 Einleitung **1**
1.1 Pepperl + Fuchs / HMI 1
1.2 Intention und Ziel der Arbeit 2
1.3 Anforderungen 3

2 Stand der Technik **5**
2.1 Vorarbeiten zur Bachelorthesis 5
2.1.1 SQLite Embedded Datenbank 5
2.1.2 HWiNFO 6
2.2 Software Design Konzepte 6
2.2.1 Adapter Pattern 7
2.2.2 Strategie Design Pattern 8
2.3 Datenbanken 8
2.3.1 SQL - Structured Query Language 9
2.4 MTBF und Reliability 10
2.5 FuzzyLogic 11
2.5.1 Architektur eines Fuzzy Logic Systems 12
2.6 Grafana 13

3 Architektur Konzept **14**
3.1 Datenerfassung 14
3.1.1 Entwurf einer Architektur zum Auslesen der Systemhardware . . . 15
3.1.2 Entwurf eines Datenbankmodells zum Speichern der Messwerte . . . 17

Abbildungsverzeichnis

1.1 Standorte der Pepperl+Fuchs SE 1

2.1 Adapter Pattern Struktur [5] 7
2.2 Strategie Pattern Struktur [5] 8
2.3 Volumen der weltweit generierten Daten bis 2027 [7] 9
2.4 SQL Befehls Kategorien [8] 9
2.5 Bathtub Curve [12] 10
2.6 Vergleich von Fuzzy Logic zu Boolescher Logik [14] 11
2.7 Architektur eines Fuzzy Logic Systems [14] 13
2.8 Grafana Dashboard Beispiel [17] 13

3.1 Architektur der HM.HWServices für das Auslesen der Hardware 15
3.2 Datenstruktur zum zwischenspeichern der Sensordaten 16

Tabellenverzeichnis

Abkürzungsverzeichnis

P+F Pepperl+Fuchs

MTBF Mean Time Between Failures

API Application Programming Interface

SDK software development kit

HMI Human-Machine-Interface

SQL Structured Query Language

SEQUEL Structured Englisch Query Language

1 Einleitung

Im folgenden Kapitel wird die Grundlage für die vorliegende Bachelorarbeit geschaffen. Hierbei werden die Motivation und das Ziel der Arbeit erläutert, um einen klaren Überblick über den Themenkontext zu bieten. Des Weiteren werden die Anforderungen an das System gestellt.

1.1 Pepperl + Fuchs / HMI

Die Pepperl+Fuchs (P+F) wurde 1945 von Walter Pepperl und Ludwig Fuchs gegründet. Anfangs war sie eine Radioreparaturwerkstadt, welche sich erst nach der Entwicklung eines eigenen Näherungsschalters so wie eines eigensicheren Transistorverstärkers auf das Gebiet der Elektronik ausweitete. Inzwischen entwickelt, produziert und vertreibt P+F Baugruppen und Sensoren für den Automatisierungsmarkt.

Abbildung 1.1: Standorte der Pepperl+Fuchs SE

Im Bereich der Prozessautomation ist P+F führender Hersteller industrieller Sicherheitsausstattungen. Das Produktportfolio umfasst eine Reihe von industrieller Computersysteme, welche zur Überwachung und Steuerung von Prozessen in explosionsgefährdeten Bereichen genutzt werde. Die Human-Machine-Interface (HMI) Abteilung beschäftigt sich mit der Entwicklung dieser Systeme, welche eine Schnittstelle zwischen Mensch und Maschine bilden. Da es eine Vielzahl an Anwendungen für HMI Systeme im explosionsgefährdeten Bereichen gibt, wurde die Produktfamilie *VisuNet* speziell für den Einsatz in diesen Zonen konzipiert. Solche Ex-Zonen sind überall da zu finden, wo explosionsgefährliche Stoffe gelagert oder gehandhabt werden. In diesen Zonen kann durch Gas oder Staub, eine explosionsfähige Atmosphäre entstehen. Zudem kommen auch umwelttechnische Einflüsse wie Sonne, Nässe, Hitze und Kälte, aber auch Einflüsse, welche beispielsweise durch die Reinigung mit aggressiven Chemikalien entstehen, hinzu. Um in einen solchen Computer feindlichen Umgebung dennoch ein Prozessleitsystem zu integrieren, setzte P+F mit den *VisuNet* Remote Monitoren auf die Thin-Client-Technologie. Hierbei verbindet sich der Ex-Geschützte Monitor aus der Explosionsgefährdeten Zone mit der Zentralen, meist leistungsstärkeren, Recheneinheit in der No-Ex-Zonen. Eingaben über Tastatur und Maus werden anschließend über den Monitor an die Zentrale Recheneinheit weitergeleitet, welche anschließend die neuen Ausgaben an den Monitor zurückschickt.

1.2 Intention und Ziel der Arbeit

Durch die Ex-Schutzzertifizierung der *VisuNet* Plattformen, sind vorgeschriebene Betriebstemperaturgrenzen der Systeme einzuhalten. Durch integrierte Schutzschaltungen und weitere Sicherheitsmechanismen ist es den Geräten nicht möglich, diese Grenzwerte zu überschreiten. Durch den Einsatz dieser Geräte in industriellen Umgebungen, können sich verschiedene Umwelteinflüsse, wie beispielsweise die Sonneneinstrahlung, negativ auf das System auswirken. Zudem können diese Umwelteinflüsse das Gerät auch in Systemzustände bringen, welche den Zertifizierungsrichtlinien widersprechen. Durch eine falsche Einschätzung der Umweltfaktoren werden solche unzulässigen bzw. schädlichen Betriebe vom Endkunden meist nicht wahrgenommen.
Eine Möglichkeit dem zu begegnen, ist die auf der Hardware verbaute Sensorik zu nutzen, um den Zustand des Systems zu überwachen. Aus den Daten können anschließend Rück-

schlüsse, auf Nutzungsverhalten und die daraus resultierenden Betriebszustände gezogen werden. Dem Nutzer können somit wichtige Informationen zum Systemzustand vermittelt werden, sodass schädliche bzw. kritische Zustände entdeckt und vermieden werden können. Primäres Ziel dieser Bachelorarbeit ist daher, die prototypische Entwicklung einer Hardware-Health Monitoring Lösung für die Pepperl+Fuchs HMI Plattformen *VisuNet GXP* und *FLX*. Hierbei soll eine Architektur und Konzepterweiterung für den Aufbau einer verteilten Health-Monitoring Lösung, welche möglichst plattformunabhängig ausgeführt werden kann, erstellt werden. Dazu gilt es Sensoren und Messwerte, welche bei der„Health"- Status Definition berücksichtigt werden sollen, auszuwählen und auszulesen. Des Weiteren soll ein Modell definiert werden, welches die Sensor-Messwerte in einen plattformspezifischen Health-Status übersetzt, um dem Benutzer den aktuellen Zustand (Ampel-Prinzip) mitzuteilen. Als prototypische Umsetzung soll ein Health Agent, entwickelt werden, welche Geräte-Daten (z.B. aktuelle Temperatur) der HMI Systeme ausliest, abspeichert, in der VisuNet RM Shell 6 dem Benutzer visualisiert und per Netzwerkprotokoll (z.B. MQTT oder SNMP) einem zentralen Server bereitstellen kann.

1.3 Anforderungen

Durch das in Abschnitt 1.2 erläuterte Ziel dieser Arbeit, ergeben sich die unten aufgelisteten Anforderungen.

1. Schnittstelle zur Datenerfassung

    a) Auslesen der Systemsensorik
    Das System benötigt eine zentrale Schnittstelle, welche das Auslesen der Sensoren der VisuNet Plattformen ermöglicht. Die *VisuNet* Plattformen unterscheiden sich in der ausgestatteten Elektronik so weit, dass verschiedene Ansätze zum Auswerten dieser benötigt werden.

    b) Speichern der ausgelesenen Messwerte
    Zur Verwaltung der gesammelten Daten benötigt das System eine zentrale Datenbank. Diese muss über eine übersichtliche und effiziente Struktur verfügen, welche das Verarbeiten der gesammelten Daten im Nachgang ermöglicht.

2. Datenverarbeitung

    a) Ermittelung des Health Status
    Für die jeweiligen *VisuNet* Plattformen soll der System Health Status ermittelt werden. Dieser soll eine Aussage über den Betrieb des Systems geben.

    b) Ermittelung der Health Status Historie
    Über die gesammelten Health Status Daten soll zudem eine Historie erstellt werden. Diese soll eine Aussage über das generelle Nutzungsverhalten des Systems treffen.

    c) Ermittelung der System Reliability
    Zudem soll eine Aussage über den Zustand des Systems, mittels Reliability, getroffen werden.

3. Datendistribution

    a) Visualisierung der Systemdaten
    Die gesammelten Daten des Systems sollen über ein Dashboard visualisiert werden. In diesem soll dem Kunden, eine Auswertung des Systemverhaltens präsentiert werden.

    b) Distribution der Daten zu einem externen Service
    Die gesammelten Daten des Systems sollen über ein Netzwerkprotokoll (MQTT, SNMP) an einen dritten Service übermittelt werden können.

2 Stand der Technik

In diesem Kapitel werden die in dieser Arbeit verwendeten Konzepte und Technologien beleuchtet.

2.1 Vorarbeiten zur Bachelorthesis

Zum Thema dieser Bachelorarbeit wurden bereits zwei Vorarbeiten geleistet. Zum einen wurde im Rahmen einer Praxisphase, eine Voruntersuchung zum Thema *Condition-Based-Monitorig für industrielle PCs* vorgenommen. Die im Rahmen dieser Arbeit [1] durchgeführte Grundlagenuntersuchung und Marktrecherche hat auf zwei Computer Monitoring Technologien aufmerksam gemacht. Diese wurden anschließend in der zweiten Vorarbeit [2] evaluiert. Aus dieser Bewertung heraus, wurde sich für die *HWiNFO* Software, zum Auslesen der auf der Hardware verbauten Sensoren entschieden. Die Software wird genauer in Abschnitt 2.1.2 behandelt. Des Weiteren wurde in der Vorarbeit [2] auch eine geeignete Datenbank für das Health Monitoring System ausgewählt. In Abschnitt 2.1.1 wird die ausgewählte Datenbanktechnologie genauer beschrieben.

2.1.1 SQLite Embedded Datenbank

In der Vorarbeit zu dieser Bachelorarbeit wurde bereits eine Auswahl für eine Datenbank getroffen. Hierbei wurden drei Datenbanken in den Punkten Performanz, Größe der Anwendung, Ressourcennutzung und der Dokumentation miteinander verglichen. Aus dem Vergleich hervorgehend, wurde sich anschließend für die Verwendung der SQLite Embedded Datenbank Engine entschieden. Diese bietet eine zuverlässige, kleine, schnelle und voll funktionale Datenbank Engine, welche vollständige in das Gesamtsystem integriert

werden kann [3]. Zur Implementierung der Datenbank in die Anwendung wird die System.Data.SQLite Bibliothek für C# verwendet.
Auf das Thema Datenbanken wird in Abschnitt 2.3 genauer eingegangen.

2.1.2 HWiNFO

Hwinfo ist eine Software der Firma REALiX, welche zum Überwachen und Analysieren der Hardware eines Computers konzipiert wird. Über die grafische Oberfläche des Programms, lassen sich alle gesammelten Daten anzeigen. Der Nutzer kann diese Informationen nutzen, um Defekte an der Hardware zu erkennen.
Das ausschlaggebende Argument für die Nutzung der Software liegt in der Application Programming Interface (API). Über die Shared Memory Funktion der Software lassen sich alle Gerätedaten, die die Software auslesen kann über eine C# Bibliothek auslesen. Hierzu muss die Funktion in den Einstellungen des Programms eingeschaltet werden. Da die 64 bit Version des Tools dies nur für einen Zeitraum von 12h erlaubt, wird für den Verlauf der Arbeit die 32-bit Version der Software verwendet. Zum anderen bietet der REALiX einen software development kit (SDK) welcher alle Funktionen der Software in Form einer Bibliothek bereitstellt.[4]
Im Verlauf der Arbeit wird die Shared Memory Funktion der Software für die prototypische Implementierung des Health Monitorig Systems genutzt.

2.2 Software Design Konzepte

Eine solide Softwarearchitektur ist entscheidend für die erfolgreiche Entwicklung und Wartung eines Programmes. Sie legt den Grundstein für die anschließende Implementierung. Durch eine gute Architektur wird sichergestellt das Programm Skalierbar, Effizient, robust und gut zu warten ist.
Hierbei bieten sogenannte Design Patterns Abhilfe. *Jedes Muster beschreibt zunächst ein in unserer Umwelt immer wieder auftretendes Problem, beschreibt dann den Kern der Lösung dieses Problems, und zwar sodass man diese Lösung millionenfach anwenden kann, ohne sich je zu wiederholen* (Christof Alexander *Eine Muster-Sprache* [Löcker verlag, Wien, 1995, Seite x]). Diese Definition für muster bezieht sich auch auf objektorientierte Design Patterns. Das Verwenden dieser Patterns ermöglicht Entwicklern von der Erfah-

rung anderer zu profitieren, um bereits gelöste Probleme nicht nochmal lösen zu müssen. Zudem steigern sie auch die Codequalität. Der Code wird lesbarer und die Wartung dessen wird leichter. Zudem wird auch die Implementierung neuer Erweiterungen und das Eindenken in die Software durch gängige Designpatterns erleichtert. [5, S.25 ff]
Alle gut strukturierten objektorientierten Architekturen basieren auf Mustern (Grady Booch [5, S.21]). In den folgenden Kapiteln wird genauer auf die in dieser Arbeit verwendeten Design Patterns eingegangen.

2.2.1 Adapter Pattern

Zweck des Adapter Patterns ist die Anpassung der Schnittstelle einer Klasse an eine andere von dem Client erwarteten Schnittstelle. Somit ermöglicht das Pattern die Zusammenarbeit von zwei Klassen, welche aufgrund ihrer Schnellen nicht möglich wäre. Das Adapter Patern ist auch unter dem Namen Wrapper bekannt, welcher im folgenden Verlauf der Arbeit verwendet wird.
Das Pattern kommt immer dann zum Einsatz, wenn eine bereits existierende Klasse genutzt werden soll, jedoch die Schnittstelle der Klasse nicht mit den aktuellen Anforderungen des Clients übereinstimmt. Des Weiteren wird das Pattern verwendet, wenn eine wiederverwendbare Klasse erzeugt werden soll, welche mit unabhängigen und nicht vorhersehbaren Klassen interagieren soll.

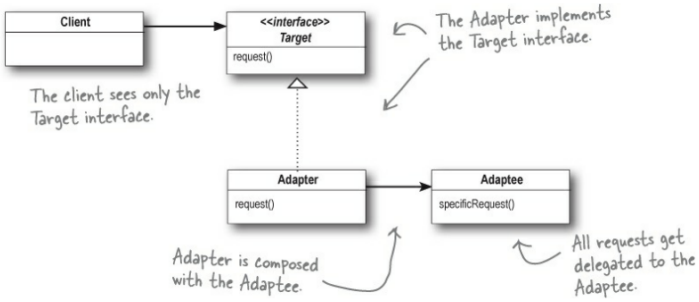

Abbildung 2.1: Adapter Pattern Struktur [5]

Das Design Pattern besteht aus einem *Target*, welches die vom Client verwendete Schnittstelle definiert. Zudem kommt der *Client*, welcher mit den Objekten zusammen arbeitet, die der Zielschnittstelle entsprechen. Zuletzt beinhaltet das Adapter Pattern einen *Adaptee* so wie den *Adapter* selbst. Der *Adaptee* definiert eine bestehende Schnittstelle, welche vom *Adapter* adaptiert werden muss.

Der *Client* ruft die gewünschte Operation auf einer *Adapter*-Instanz auf, welche anschließend die gewünschten *Adaptee*-Operation ausführt.

2.2.2 Strategie Design Pattern

Zweck des Strategy (Strategie) Patterns ist es, eine Familie von einzelnen gekapselten und austauschbaren Algorithmen zu schaffen. Dieses Patern ermöglicht eine variable und vom Client unabhängige Nutzung des Algorithmus.
Das Pattern kommt zum Einsatz, wenn eine Reihe von zusammenhängenden Klassen sich nur in Ihrem Verhalten unterscheiden, verschiedene Varianten eines Algorithmus erfordert werden, der Client keine Kenntnis von den vom Algorithmus verwendeten Daten haben soll, oder eine Klasse verschiedene Verhaltensweisen aufweist.

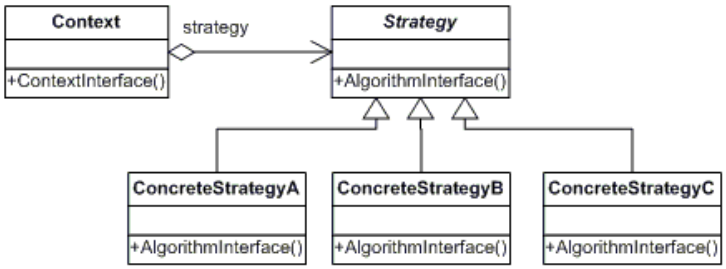

Abbildung 2.2: Strategie Pattern Struktur [5]

Das Design Pattern besteht aus den folgenden Teilnehmern. Die *Strategy*, welche eine gemeinsame Schnittstelle für die verwendeten Algorithmen deklariert. Einer oder mehreren *ConcreteStrategy*, welche die Implementierung der Algorithmen oder Klassen ist, so wie dem *Context*, welcher mit einer *ConcreteStrategy* ausgestattet wird. Des Weiteren besitzt der *Context* eine Referenz auf das *Strategy* Objekt.[5, S.383 ff]
Über den *Context* kann anschließend zur Laufzeit des Programmes die benötigten *ConcreteStrategy* geladen und ausgeführt werden. Ein konkretes Beispiel hierzu wird im Buch Head First Design Patterns [6] behandelt, was den nutzen dieses Patterns nochmal verdeutlicht.

2.3 Datenbanken

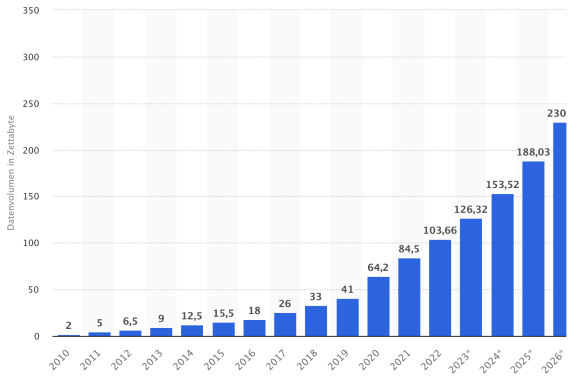

Abbildung 2.3: Volumen der weltweit generierten Daten bis 2027 [7]

Weltweit wurden im Jahr 2022 Daten im Umfang von 103.66 Zettabyte erfasst. Diese Zahl wird sich laut Statistik 2.3 bis zum Jahr 2026 verdoppelt haben. Angesicht dieser Zahlen, sind Datenbanken aus der heutigen Zeit nicht wegzudenken. Sie bieten eine Möglichkeit, große Mengen an Daten strukturiert abzuspeichern und anschließend auszuwerten.
Hierbei werden Datenbanken grundsätzlich in zwei Kategorien unterteilt. Relationale Datenbank und "Nicht relationale Datenbanken". Unterschiede der Datenbankarten machen sich in der Sprache zum Auswerten der DB, ihrer Skalierbarkeit, der Struktur, der Eigenschaften und der Unterstützung durch die Community bemerkbar.

2.3.1 SQL - Structured Query Language

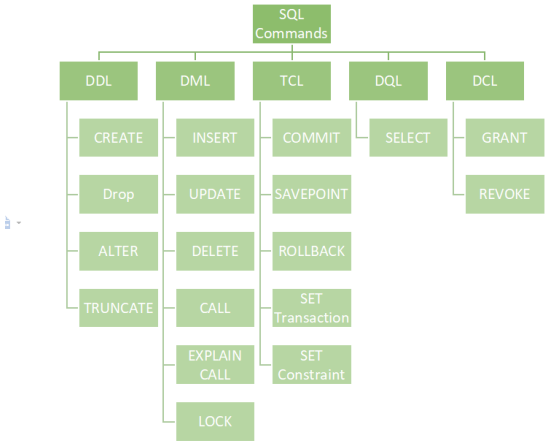

Abbildung 2.4: SQL Befehls Kategorien [8]

IBM-Forscher Edgar F. Codd definierte 1969 ein Datenbankmodell für Relationale Datenbanken. Auf Grundlagen seiner Forschung begann, in den folgenden Jahren, die Entwicklung der Sprache Structured Englisch Query Language (SEQUEL). Codds Modell für basiert auf der Zuordnung von Schlüsseln. Nach einigen Überarbeitungen der Implementierung wurde diese anschließend in Structured Query Language (SQL) umbenannt.
SQL ermöglicht insbesondere die Speicherung, Bearbeitung so wie eine Abfrage von Daten in einer Datenbank. Mithilfe des Prinzips der Schlüssel können Datensätze miteinander verknüpft werden. Somit kann einem Benutzernamen beispielsweise ein echter Name, eine Telefonnummer und eine E-Mail Adresse zugewiesen werden.
Die besondere Eigenschaft von SQL ist das Konzept von Arrays. Relationale Datenbanken bestehen aus Arrays, welche sich mithilfe von verschiedenen Befehlen erzeugen und

bearbeiten. [9]
SQL bietet eine Reihe von Befehlen, welche die Interaktion mit der Datenbank ermöglichen. Diese können grundsätzlich in 5 Kategorien eingeteilt werden (siehe Abb. 2.4). Die wichtigsten Befehle sind dabei *INSERT*, *UPDATE* und *DELEAT*, mit welchen sich Datensätze schreiben und bearbeiten lassen. Zudem der kommt der *SELECT* Befehl, welcher das Auslesen von Datensätzen ermöglicht. Um die Tabellenstruktur der Datenbank zu bearbeiten kommen die Befehle *CREATE* und *DROP* zum Einsatz. [8]
Natürlich bietet die Programmiersprache eine weitaus komplexere Syntax, um Datensätze sortiert auswerten zu können. Eine vollständige Dokumentation der Sprache findet sich auf der w3school Webseite [10].

2.4 MTBF und Reliability

Es gibt viele Ursachen, welche zu einem Ausfall elektronischer Komponenten in einem System, führen können. Laut dem technischen Bericht [11] ist in 50% der Fälle die Temperatur der Komponenten für einen Ausfall verantwortlich. Dies liegt an den unterschiedlichen thermischen Ausdehnungskoeffizienten des Materials auf der Platine haben. Durch die unterschiedliche Ausdehnung der Bauteile und der Platine selbst, kommt es zu hohen Belastungen der Lötstellen. Während sich dieser Zyklus wiederholt, können Risse in den Verbindungen entstehen und ausbreiten. Diese können anschließend zu einem Bruch im elektrischen Stromkreis führen. [12]

Die in Abbildung 2.5 abgebildete Bathtub-Kurve ist ein Konzept, welches zur Beschreibung der Lebensdauer von elektronischer Komponenten verwendet wird. Dabei kann die Lebenszeit in drei Abschnitte unterteilt werden. Die Bathtub-Kurve beschreibt eine mittlere Betriebsdauer zwischen Ausfällen. Sie weist drei Betriebsphasen auf. In der ersten Phase, bekannt als *Infant Mortali-*

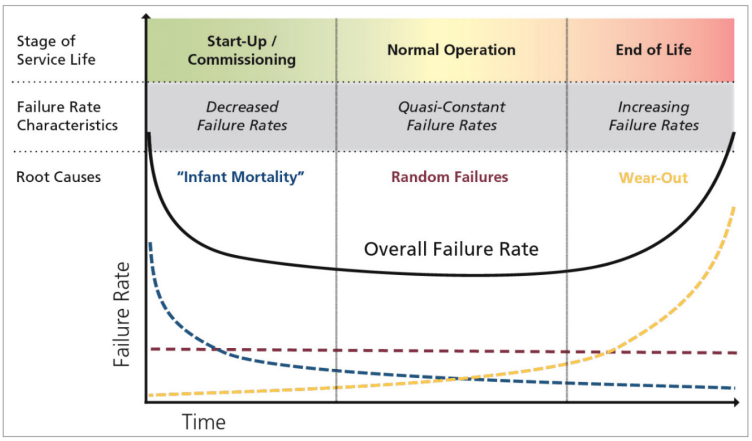

Abbildung 2.5: Bathtub Curve [12]

ty, kommt es durch Konstruktions-, Produktions- und Werkstoffmängel häufig gleich zu Beginn des Betriebs zu Fehlern und Ausfällen. Geräte, die von diesen Problemen nicht betroffen sind, laufen meist zuverlässig durch die zweite Phase der Kurve, bekannt als *Random Failures*. Hierbei kommt es nur deutlich seltener und vereinzelt zu Ausfällen. Zum Ende der Lebensdauer kommt es, in der *Wear-Out* Phase, durch Alterung und Verschleiß wieder vermehrt zu Ausfällen. [12]
Der Mean Time Between Failures (MTBF) ist dabei eine statistische Kennzahl, die den durchschnittlichen Zeitraum in Stunden angibt, der zwischen zwei aufeinanderfolgenden Ausfällen einer bestimmten Komponente, eines Systems oder eines Produkts verstrichen is. Dieser weist zudem eine Temperaturabhängigkeit auf. Beispielsweise bei Kapazitäten kann im Durchschnitt gesagt werden: *Eine Erhöhung der Betriebstemperatur um 10°C, führt zu einer Halbierung der Lebenserwartung.* Ein MTBF von 100h sagt also aus, dass ein System im Durchschnitt, 100h laufen wird, bevor es zu einem Fehler kommen wird.[13]
Die Zuverlässigkeit (Reliability) eines Gerätes hingegen ist als die Wahrscheinlichkeit definiert, mit der ein System seine beabsichtigten Funktionen für einen festgelegten Zeitraum erfüllen wird. Hat ein System bei 100h eine Zuverlässigkeit von 0.8, so besteht eine 80% Wahrscheinlichkeit, dass das System nach 100h noch funktioniert.[13]
Die Zuverlässigkeit eines Systems kann über den MTBF mit Formel 2.1 berechnet werden. Dabei ist zu beachten, dass man die Temperaturabhängigkeiten des Systems beachtet.

$$R(t) = e^{-\frac{t}{\text{mtbf}}} \tag{2.1}$$

2.5 FuzzyLogic

Fuzzy Logic, erstmals in den 1960er Jahren von Lotfi Zadeh an der University of California entwickelt, stellt einen innovativen Ansatz der Datenverarbeitung dar, der auf Wahrheitsgraden basiert. Im Gegensatz zur herkömmlichen Booleschen Logik, die sich auf binäre Zustände von 1 oder 0 bzw. wahr oder falsch stützt, zeichnet sich die Fuzzy Logic durch ihre Fähigkeit aus, die Vielschichtigkeit von Zwischenzuständen zu berücksichtigen.[15]

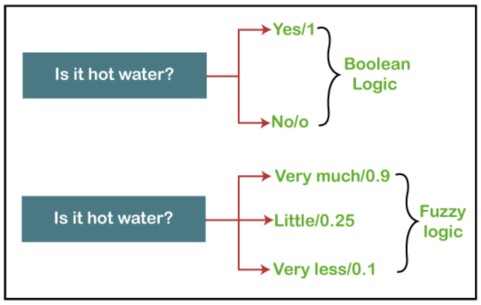

Abbildung 2.6: Vergleich von Fuzzy Logic zu Boolescher Logik [14]

Das zentrale Merkmal der Fuzzy-Logik besteht darin, unpräzise Argumentationsweisen zu modellieren, die eine bedeutende Rolle in der bemerkenswerten Fähigkeit des Menschen spielen, unter Bedingungen der Ungewissheit und Ungenauigkeit rationale Entscheidungen zu treffen (Siehe Abbildung 2.6). Diese Fähigkeit basiert auf unserem Vermögen, aus einem Wissensbestand, der ungenau, unvollständig oder nicht völlig zuverlässig ist, ungefähre Antworten auf Fragen abzuleiten. Anders als in klassischen logischen Systemen strebt die Fuzzy Logic danach, die Grauzonen zwischen klaren Kategorien zu erfassen und somit eine flexiblere und menschlichere Art der Datenverarbeitung zu ermöglichen. Dieser Ansatz hat Anwendungen in verschiedenen Bereichen gefunden, darunter Steuerungssysteme, künstliche Intelligenz, Entscheidungsfindung und mehr. [16]

2.5.1 Architektur eines Fuzzy Logic Systems

Ein Fuzzy Logic Systems kann in vier Module unterteilt werden. Jede dieser Komponenten spielt dabei eine entscheidende Rolle für das gesamte System. Abbildung 2.8 zeigt den Aufbau eines Fuzzy Logic Systems auf.
Die Umwandlung der Systemeingänge ist Aufgabe des *Fuzzification* Moduls. Dabei werden exakten Werte in sogenannte Fuzzy-Sets umgewandelt. Die *Inference Engine* bestimmt anschließend den Grad der Übereinstimmung des aktuellen Fuzzy-Sets in Bezug auf jede Regel und trifft eine Entscheidung darüber, welche Regeln gemäß dem Eingangsfeld ausgelöst werden sollen. Durch eine Kombination der ausgelösten Regeln wird anschließend eine Steuerungsaktion formuliert. Das *Defuzzification* Model wird im Anschluss dazu verwendet, aus den durch die *Interence Engine* erhaltenen Fuzzy-Setz, exakte Ausgangswerte zu erhalten. Die *Rule Base* umfasst eine Sammlung von Regeln und den "*IF-THEN* Bedingungen, welche im Vorfeld definiert und dem System bereitgestellt werden. Diese werden verwendet, um das Entscheidungssystem auf Grundlage von linguistischen Variablen zu steuern. [14]

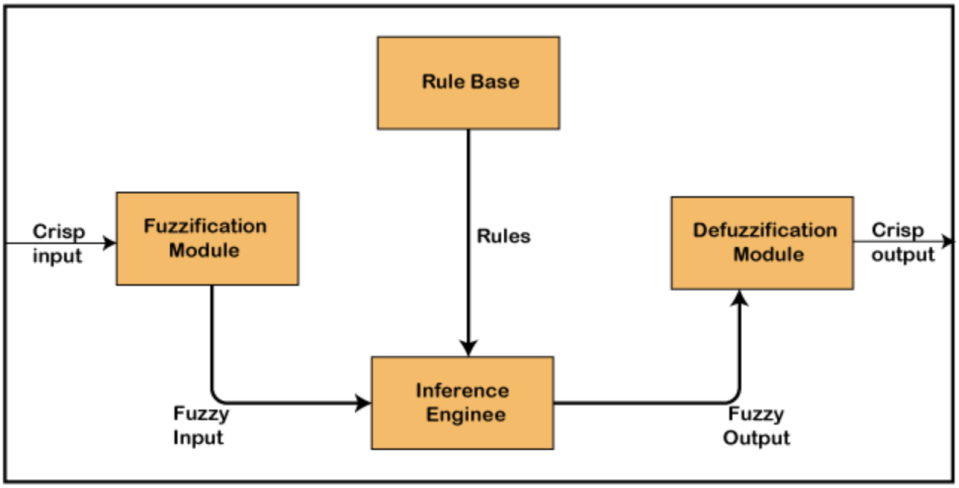

Abbildung 2.7: Architektur eines Fuzzy Logic Systems [14]

2.6 Grafana

Grafana ist eine Open-Source-Software zur plattformübergreifenden Analyse von Metriken und Daten. Als Webservice gehostet, ermöglicht es die Visualisierung von Daten mithilfe von vorgefertigten Vorlagen für Tabellen und Graphen. Diese Software unterstützt verschiedene Datenquellen und kann problemlos direkt mit einer relationalen Datenbanken verbunden werden. Grafana erleichtert somit die präzise Darstellung und Interpretation von Informationen durch benutzerfreundliche Visualisierungstools. [17]

Abbildung 2.8: Grafana Dashboard Beispiel [17]

3 Architektur Konzept

In diesem Kapitel wird die Struktur und Architektur der Hardware-Health-Monitroing Lösung erläutert. Die Architektur wird dabei zur Übersichltichkeit in drei Teilkomponenten unterteilt.
Abschnitt 3.1 befasst sich dabei mit der Datenerfassung. Hier wird das Konzept zum Plattform übergreifenden Auslesen der Hardwaresensorik und der Schpeicherung der erfasten Messdaten beleuchtet. In Abschnitt **??** wir anschließend das Modell zur bewertung des Systemzustnads erläutert. Das Konzept zur Visualisierung der Daten wird anschließend in **??** beleuchtet. Zuletzt werden die Teilsysteme in Abschnitt **??** zu einem Gesamtsystem zusammengeführt.

3.1 Datenerfassung

Die Herausforderung beim Auslesen der Daten liegt darin, die plattformspezifischen Informationen in einem allgemeinen Datenmodell zu konsolidieren. Hierbei stammen die Daten aus verschiedenen Schnittstellen. Die Architektur muss in der Lage sein, sämtliche verfügbaren Sensordaten plattformunabhängig auszulesen, darunter beispielsweise Temperaturen und CPU-Auslastung. Zusätzlich sollte sie die Integration weiterer plattformspezifischer Hardwarekonfigurationen und Schnittstellen ermöglichen, ohne eine grundlegende Neustrukturierung des bestehenden Codes zu erfordern.
Das Speichern der ausgelesenen Messdaten soll in einem einheitlichen Format erfolgen, das unabhängig von der Hardwarekonfiguration der Zielplatform ist. Zudem ist eine klare und sinnhafte Struktur der Datenbank wichtig, da diese Performant und Skalierbar sein muss.

3.1.1 Entwurf einer Architektur zum Auslesen der Systemhardware

Die Objekte und Klassen, die für das Auslesen der Systemhardware verwendet werden, sind im Verzeichnis *HM.HWServices* abgelegt. Dieses Verzeichnis enthält alles, was benötigt wird, um die Hardware der Zielplatformen auszulesen.
Das UML Diagramm in Abbildung 3.1 veranschaulicht den Aufbau von *HM.HWServices*.

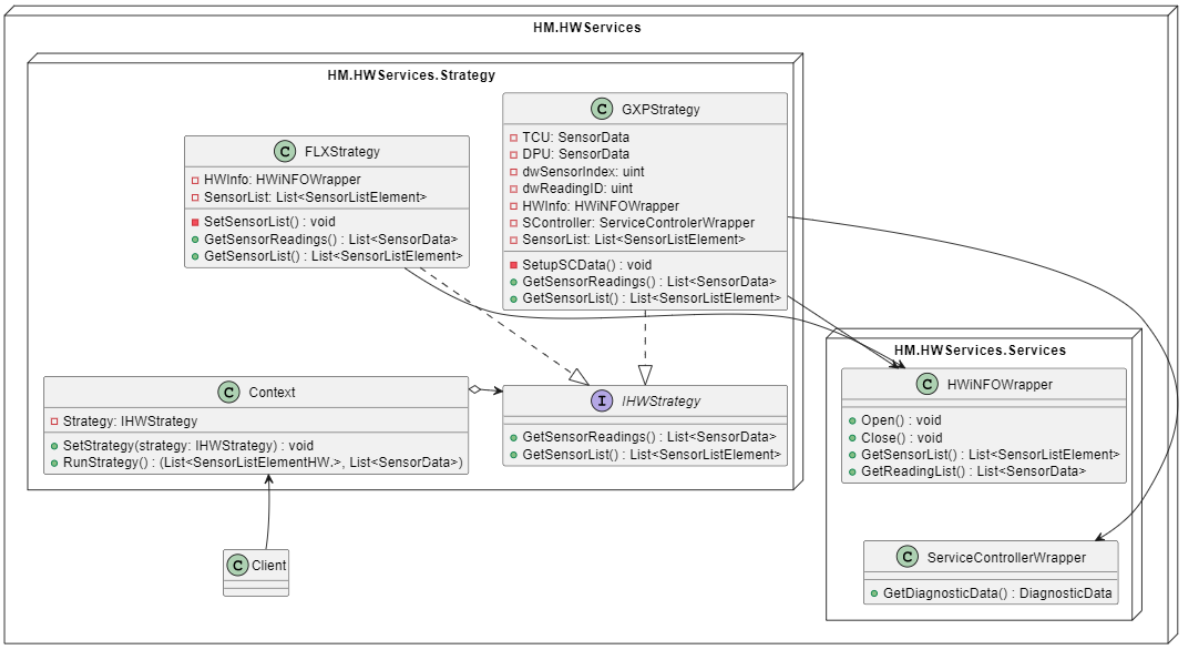

Abbildung 3.1: Architektur der HM.HWServices für das Auslesen der Hardware

Die Architektur der *HM.HWServices* wird nach dem, in Abschnitt 2.2.2 beschriebnen Strategie Muster ausgelegt. Die Komponenten des Strategie Musters sind im Verzeichnis *HM.HWServices.Strategy* organisiert.
Durch den *Context* kann die gewünschte Strategie ausgewählt und aufgerufen werden. Die Client-Anwendung interagiert dabei ausschließlich mit dem *Context*, der wiederum die gewünschten Funktionen der ausgewählten Strategieklasse aufruft.
Da sich das Auswerten von VisuNet FLX und GXP voneinander unterscheidet, soll für jede Platform eine eigene Strategieklasse verantwortlich sein. Jede dieser Strategieklasse wird hierbei von dem *IHWStrategy* Interface abgeleitet. Dieses definiert die Struktur der Klassen, sodas die Funktion der ausgewählten Klasse im *Context* ausgeführt werden kann, ohne die spezifische Strategieklasse genau zu kennen. Anschließend kann beim Start des Programms entschieden werden, welche dieser Strategien angewendet werden soll.

Muss das Programm in Zukunft um eine neue Hardwarekonfiguration einer Platform erweitert werden, kann dies über das hinzufügen einer weiteren Strategieklasse realisiert werden.
Zum Auslesen der Sensordaten stehen zunächst zwei Schnittstellen zurverfügung. Zum einen liefert über die in Abschnitt 2.1.2 beschriebene Shared Memory Funktion der HWiNFO Software Messwerte aller an das Mainboard angeschlossenen Sensoren. Die VisuNet FLX Platform bedarf keiner weiteren Schnittstellen zum lesen von Sensordaten.
Um die Temperatursensoren in DPU (Display Unit) und TCU (Thin Client Unit) der VisuNet GXP Plattform aus zu lesen, wird eine Weitere Bibliothek verwerwendet, welche die Komunikation mit dem in der Platform verbauten Servicecontroller ermöglicht.
Für beide Schnittstellen wurde eine Wrapperklasse (Siehe Abschnitt 2.2.1) konzipiert, welche die wesentlichen Funktionen in einer übersichtlichen Klasse bereitstellen und die Datenstruktur der Schnittstelle adaptieren. Die Wrapperklassen werden im Verzeichnis *HM.HWServices.Wrapper* organisiert. Die Wrapperklassen sind in Abbildung 3.1 zu sehen.

Adaption der Schnittstelle

Beim Auslesen der Sensorik über die *HWiNFOWrapper* Klasse, können zwei Listen ausgelesen werden. Zum einen werden alle verfügbaren Sensoren der Platform in einer Liste von *SensorListElement* abgelegt. Zum anderen wird eine Liste des Typen *SensorData* erzeugt. In dieser ist jeder Sensor mit dem aktuellen Messwertwert hinterlegt. Die Datentypen hierzu sind in Abbildung 3.2 abgebildet.

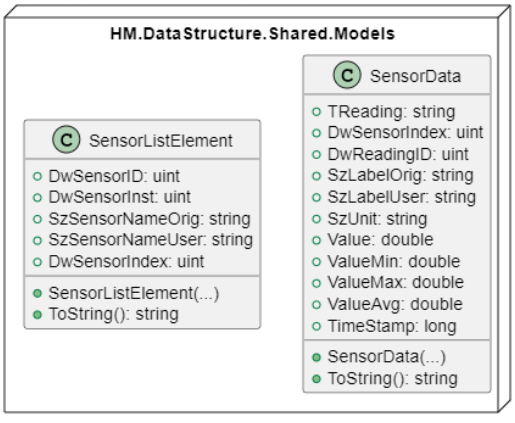

Abbildung 3.2: Datenstruktur zum zwischenspeichern der Sensordaten

Strategie Konzept

Ziel der Strategieklassen ist wie zuvor bereits erwähnt die Kapselung der Algorithmen zum Auslesen der Zielplatformen. Eine Strategie soll daher folgende daten zurück liefern: Zum einen soll eine Aufzählung aller Sensoren im System erstellt werden, und in einer Liste von *SensorListElement* hinterlegt werden. Zum anderen soll ein aktueller auszug der Sensorwerte erstellt werden können. Hierzu sollen die Werte der einzelnen Sensoren in einer Liste von *SensorData* gespeichert werden.
Da die *HWiNFOWrapper* Schnittstelle alle verfügbraren Sensoren der VisuNet FLX Platform abdekt, ist die Strategie zum Auslesen dieser Platform recht Simpel. Dabei soll über einen Funktionsaufruf eine Liste mit den Aktuellen messwerten der Sensoren erstellt und zurückgegeben werden.

3.1.2 Entwurf eines Datenbankmodells zum Speichern der Messwerte

Literaturverzeichnis

[1] M. Gütermann, „Condition-based Monitoring für industrielle PC's“, Pepperl + Fuchs SE, Techn. Ber., 2023.

[2] V. Mostowoj, „Entwicklung einer Architektur für die Erfassung, Auswertung und Speicherung von Sensordaten zur Umsetzung eines System Health Monitorings“, 2023.

[3] „SQLite“. (2023), Adresse: %7Bhttps://www.sqlite.org/index.html%7D (besucht am 11. 08. 2023).

[4] „HWiNFO Diagnostic Software“. (2023), Adresse: %7Bhttps://www.hwinfo.com/%7D (besucht am 16. 08. 2023).

[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns - Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. mitp Verlags GmbH & Co.KG, 2015, ISBN: 978-3-8266-9700-5.

[6] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, *Head First Design Patterns - A Brain Friendly Guide*. O'Reilly Media, Incorporated, 2004, ISBN: 978-0-5960-0712-6.

[7] „Volumen der jährlich generierten/replizierten digitalen Datenmenge weltweit von 2010 bis 2022 und Prognose bis 2027“. (2023), Adresse: %7Bhttps://datascientest.com/de/sql-alles-uber-die-datenbanksprache#:~:text=SQL%20oder%20%E2%80%9EStructured%20Query%20Language,darin%20enthaltenen%20Daten%20zu%20verwalten.%7D (besucht am 11. 08. 2023).

[8] „SQL | DDL, DQL, DML, DCL and TCL Commands“. (2023), Adresse: %7Bhttps://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/%7D (besucht am 11. 08. 2023).

[9] „SQL – Alles über die Datenbanksprache“. (2023), Adresse: %7Bhttps://datascientest.com/de/sql-alles-uber-die-datenbanksprache#:~:text=SQL%20oder%20%E2%80%9EStructured%20Query%20Language,darin%20enthaltenen%20Daten%20zu%20verwalten.%7D (besucht am 11. 08. 2023).

[10] „SQL Tutorial“. (2023), Adresse: %7Bhhttps://www.w3schools.com/sql/%7D (besucht am 11. 08. 2023).

[11] H. C. Fortna, „Avionics Integrity Program, Technical Report ASD-TR-84-5030“, Wright Patterson Air Force Base, Techn. Ber., 1984.

[12] E. D. A. AMIR RUBIN, „LIFE EXPECTANCY OF ELECTRONIC EQUIPMENT POST-LOSS“, *AREPA*, 2020.

[13] „Understanding MTBF and Reliability“. (2020), Adresse: %7Bhttps://relyence.com/2020/05/27/understanding-mtbf-reliability/#:~:text=The%20key%20difference%20is%20that,functioning%20at%20a%20certain%20time.&text=In%20this%20equation%3A,that%20you%20are%20interested%20in%7D (besucht am 14. 08. 2023).

[14] „Fuzzy Logic - Introduction“. (2023), Adresse: %7Bhttps://www.geeksforgeeks.org/fuzzy-logic-introduction/%7D (besucht am 16. 08. 2023).

[15] „DEFINITION Fuzzy Logic“. (2023), Adresse: %7Bhttps://www.techtarget.com/searchenterpriseai/definition/fuzzy-logic#:~:text=Fuzzy%20logic%20is%20an%20approach,at%20Berkeley%20in%20the%201960s.%7D (besucht am 16. 08. 2023).

[16] Lofti A. Zadeh, „Fuzzy logic - he logic underlying approximate, rather than exact, modes of reasoning- is finding applications that range from process control to medical diagnosis.“, Magisterarb., University of California, Berkeley, 1988.

[17] „Grafana Labs Dashbords“. (2023), Adresse: %7Bhttps://grafana.com/%7D (besucht am 16. 08. 2023).