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

18.09.2023

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

Hyper
link fixen

Abbildungsverzeichnis

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

TCU Thin Client Unit

DPU Display Unit

HTTP Hypertext Transfer Protocol

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 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 Erfahrung 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.

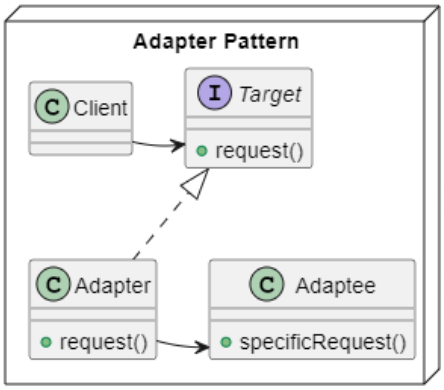

Abbildung 2.1.: Adapter Pattern Struktur [5]

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.

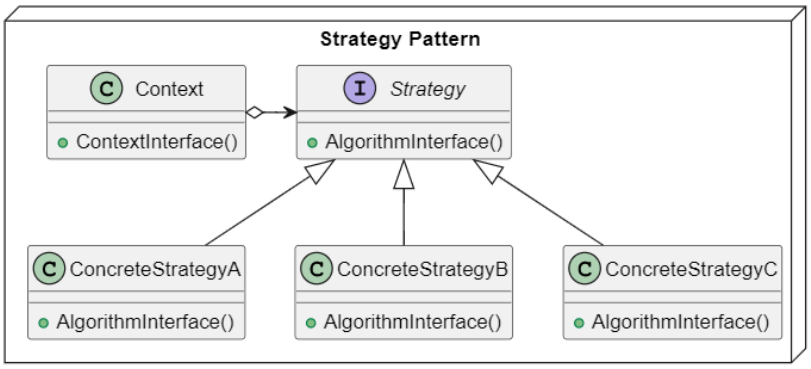

Abbildung 2.2.: Strategie Pattern Struktur [5]

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

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.

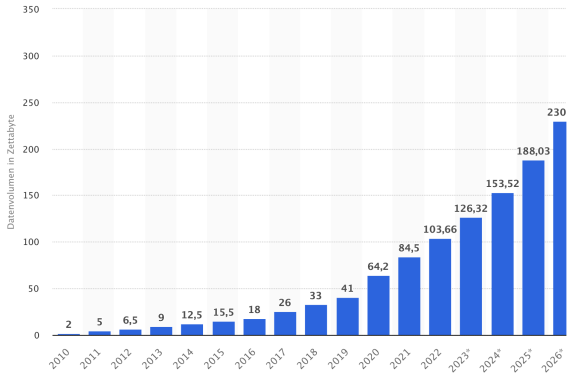

Abbildung 2.3.: Volumen der weltweit generierten Daten bis 2027 [7]

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

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.

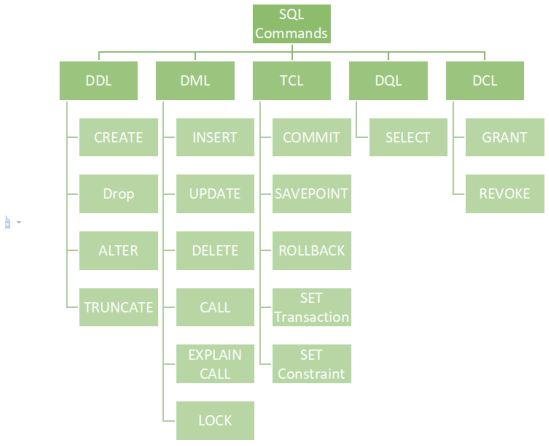

Abbildung 2.4.: SQL Befehls Kategorien [8]

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

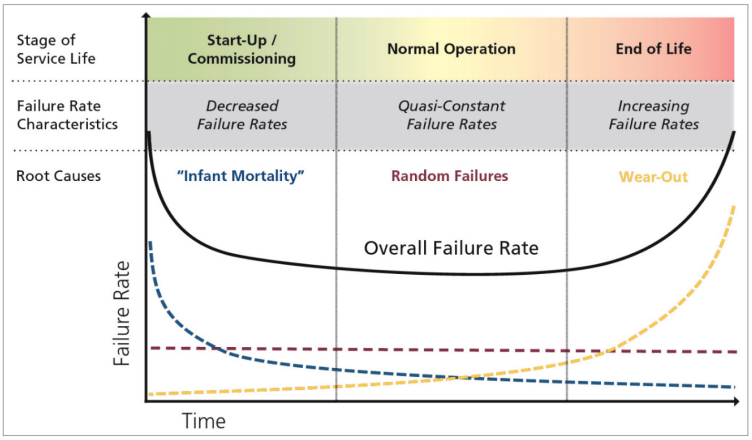

Abbildung 2.5.: Bathtub Curve [12]

beschreibt eine mittlere Betriebsdauer zwischen Ausfällen. Sie weist drei Betriebsphasen auf. In der ersten Phase, bekannt als *Infant Mortality*, 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 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]

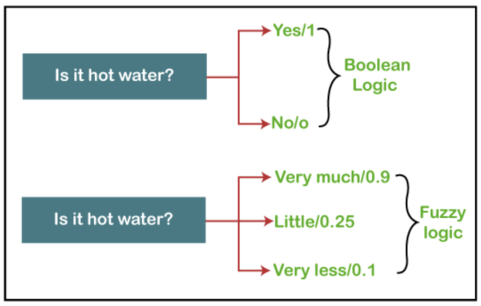

Abbildung 2.6.: Vergleich von Fuzzy Logic zu Boolescher Logik [14]

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]

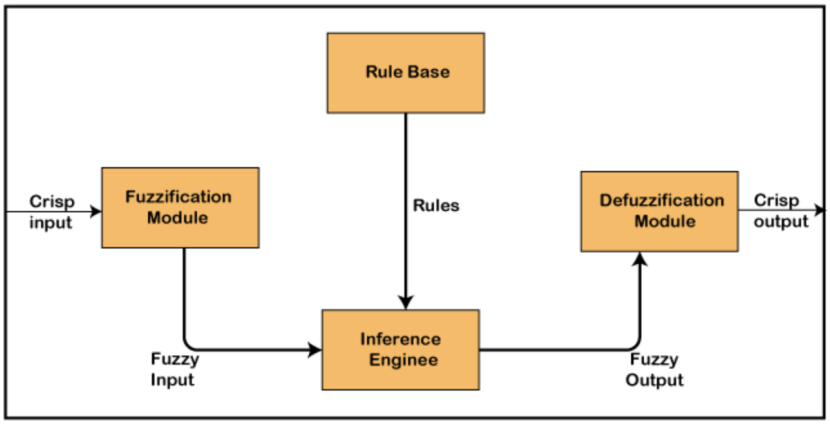

Abbildung 2.7.: Architektur eines Fuzzy Logic Systems [14]

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 Übersichtlichkeit in drei Teilkomponenten unterteilt.
Abschnitt 3.1 befasst sich dabei mit der Datenerfassung. Hier wird das Konzept zur Plattform übergreifenden Auslesen der Hardwaresensorik und der Speicherung der erfassten Messdaten beleuchtet. In Abschnitt 3.2 wir anschließend das Modell zur Bewertung des Systemzustands erläutert. Die Zusammenführung der einzelnen Teilsysteme wird in Abschnitt 3.3 beschrieben. Zuletzt wird das Konzept zur Visualisierung der Daten in 3.4 beleuchtet.

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 Zielplattform 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 Zielplattformen auszulesen.
Das UML-Diagramm in Abbildung 3.1 veranschaulicht den Aufbau von *HM.HWServices*.

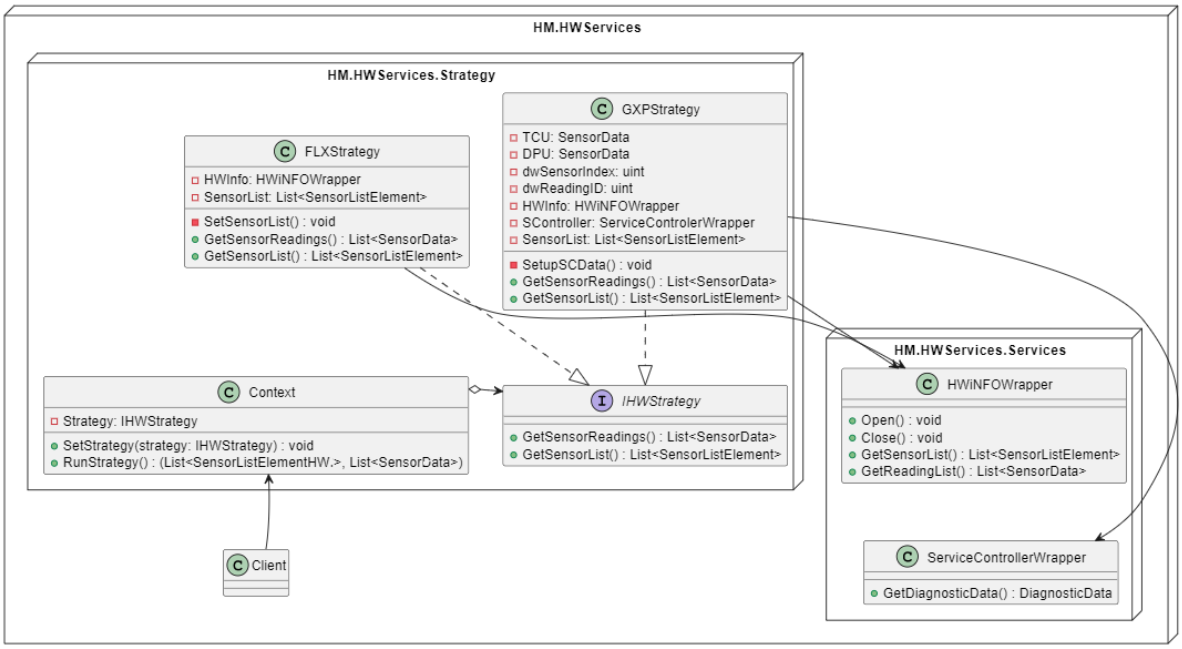

Abbildung 3.1.: Architektur der HM.HWServices für das Auslesen der Hardware

Die Architektur der *HM.HWServices* wird nach dem, in Abschnitt 2.2.2 beschrieben Strategie Muster ausgelegt. Die Komponenten des Strategiemusters 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 Plattform eine eigene Strategieklasse verantwortlich sein. Jede dieser Strategieklasse wird hierbei von dem *IHWStrategy* Interface abgeleitet. Dieses definiert die Struktur der Klassen, sodass 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 Plattform erweitert werden, kann dies über das Hinzufügen einer weiteren Strategieklasse realisiert werden.
Zum Auslesen der Sensordaten stehen zunächst zwei Schnittstellen zur Verfügung. 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 Plattform bedarf keiner weiteren Schnittstellen zum Lesen von Sensordaten.
Um die Temperatursensoren in Display Unit (DPU) und Thin Client Unit (TCU) der VisuNet GXP Plattform auszulesen, wird eine weitere Bibliothek verwendet, welche die Kommunikation mit dem in der Plattform 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 Plattform 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.

Mit UML der Wrapper erweitern

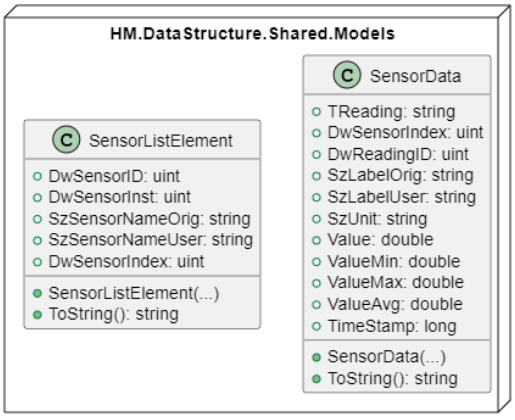

Abbildung 3.2.: Datenstruktur zum Zwischenspeichern der Sensordaten

Strategie Konzept

Ziel der Strategieklassen ist wie zuvor bereits erwähnt die Kapselung der Algorithmen zum Auslesen der Zielplattformen. Eine Strategie soll daher folgende Daten zurückliefern: Zum einen soll eine Aufzählung aller Sensoren im System erstellt werden, und in einer Liste von *SensorListElement* hinterlegt werden. (Siehe Tabelle 3.1)

DwSensorID	DwSensorInst	SzSensorNameOrig	SzSensorNameUser	DwSensorIndex
0	0	System: ARBOR ELIT-3	System: ARBOR ELIT-3922-3965U-PE	0
0	0	CPU [#0]: Intel Celeron	CPU [#0]: Intel Celeron 3965U	1
0	0	CPU [#0]: Intel Celeron	CPU [#0]: Intel Celeron 3965U: DTS	2
0	0	CPU [#0]: Intel Celeron	CPU [#0]: Intel Celeron 3965U: Enhanced	3
0	0	CPU [#0]: Intel Celeron	CPU [#0]: Intel Celeron 3965U: C-State Residency	4
0	0	Memory Timings	Memory Timings	5
0	0	CPU [#0]: Intel Celeron	CPU [#0]: Intel Celeron 3965U: Performance Limit Reasons	6
0	0	Fintek F81866	Fintek F81866	7
0	0	Intel PCH	Intel PCH	8
0	0	S.M.A.R.T.: M.2 (S80) 3	S.M.A.R.T.: M.2 (S80) 3TE7 (2CA12302100030003)	9
0	0	Drive: M.2 (S80) 3TE7 (	Drive: M.2 (S80) 3TE7 (2CA12302100030003)	10
0	0	Network: Intel I210AT (	Network: Intel I210AT Copper (Springville) Network Adapter	11
0	0	Network: Intel I210AT (	Network: Intel I210AT Copper (Springville) Network Adapter	12
0	0	Windows Hardware Er	Windows Hardware Errors (WHEA)	13

Tabelle 3.1.: Beispiel einer Liste bestehend aus *SensorListElement*

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. (Siehe Tabelle 3.2)

TReading	DwSensorIndex	DwReadingID	SzLabelOrig	SzLabelUser	SzUnit	Value	ValueMin	ValueMax	ValueAvg	TimeStamp
SENSOR_TYPE_USAGE	1	117440515	Total CPU Usage	Total CPU Usage	%	55,47	1,56	60,94	41,31	1692817646064
SENSOR_TYPE_TEMP	2	16777470	CPU Package	CPU Package	°C	46,00	42,00	48,00	44.75	1692817646064
SENSOR_TYPE_TEMP	7	16777217	Temperature 1	Temperature 1	°C	42,00	41,00	42,00	41,17	1692817646064
SENSOR_TYPE_OTHER	9	134217728	Drive Failure	Drive Failure	Yes/No	0,00	0,00	0,00	0,00	1692817646064
SENSOR_TYPE_USAGE	1	117440515	Total CPU Usage	Total CPU Usage	%	61,72	1,56	61,72	44,06	1692817649079
SENSOR_TYPE_TEMP	2	16777470	CPU Package	CPU Package	°C	46,00	42,00	48,00	44,93	1692817649079
SENSOR_TYPE_TEMP	7	16777217	Temperature 1	Temperature 1	°C	42,00	41,00	42,00	41,29	1692817649079
SENSOR_TYPE_OTHER	9	134217728	Drive Failure	Drive Failure	Yes/No	0,00	0,00	0,00	0,00	1692817649079
SENSOR_TYPE_USAGE	1	117440515	Total CPU Usage	Total CPU Usage	%	53,91	1,56	61,72	44,72	1692817652094
SENSOR_TYPE_TEMP	2	16777470	CPU Package	CPU Package	°C	48,00	42,00	48,00	45,13	1692817652094
SENSOR_TYPE_TEMP	7	16777217	Temperature 1	Temperature 1	°C	42,00	41,00	42,00	41,33	1692817652094
SENSOR_TYPE_OTHER	9	134217728	Drive Failure	Drive Failure	Yes/No	0,00	0,00	0,00	0,00	1692817652094
SENSOR_TYPE_USAGE	1	117440515	Total CPU Usage	Total CPU Usage	%	62,11	1,56	62,11	46,57	1692817655109
SENSOR_TYPE_TEMP	2	16777470	CPU Package	CPU Package	°C	46,00	42,00	48,00	45,24	1692817655109
SENSOR_TYPE_TEMP	7	16777217	Temperature 1	Temperature 1	°C	41,00	41,00	42,00	41,29	1692817655109
SENSOR_TYPE_OTHER	9	134217728	Drive Failure	Drive Failure	Yes/No	0,00	0,00	0,00	0,00	1692817655109
SENSOR_TYPE_USAGE	1	117440515	Total CPU Usage	Total CPU Usage	%	5,15	1,56	62,11	47,27	1692817658124
SENSOR_TYPE_TEMP	2	16777470	CPU Package	CPU Package	°C	47,00	42,00	48,00	45,33	1692817658124
SENSOR_TYPE_TEMP	7	16777217	Temperature 1	Temperature 1	°C	42,00	41,00	42,00	41,33	1692817658124
SENSOR_TYPE_OTHER	9	134217728	Drive Failure	Drive Failure	Yes/No	0,00	0,00	0,00	0,00	1692817658124
SENSOR_TYPE_USAGE	1	117440515	Total CPU Usage	Total CPU Usage	%	66,02	1,56	66,02	48,91	1692817661139
SENSOR_TYPE_TEMP	2	16777470	CPU Package	CPU Package	°C	46,00	42,00	48,00	45.45	1692817661139
SENSOR_TYPE_TEMP	7	16777217	Temperature 1	Temperature 1	°C	41,00	41,00	42,00	41.3	1692817661139
SENSOR_TYPE_OTHER	9	134217728	Drive Failure	Drive Failure	Yes/No	0,00	0,00	0,00	0,00	1692817661139

Tabelle 3.2.: Beispiel einer Liste bestehend aus *SensorData*

Da die *HWiNFOWrapper* Schnittstelle alle verfügbaren Sensoren der VisuNet FLX Plattform abdeckt, ist die Strategie zum Auslesen dieser Plattform recht simpel. Dabei soll über einen Funktionsaufruf eine Liste mit den aktuellen Messwerten der Sensoren erstellt und zurückgegeben werden.
Der *ServiceControllerWrapper* liefert, neben den Werten des *HWiNFOWrapper*, DPU und TCU Temperaturen der VisuNet GXP Plattform. Die Datensätze der beiden Schnittstellen müssen daher in der *GXPStrategy* kombiniert werden, sodass wie auch bei der *FLX-Strategy* jeweils eine Liste der Sensoren und eine Liste mit den Messwerten erstellt wird (Siehe Tabelle 3.1 und 3.2).
Mithilfe der Strategieklassen kann somit für jegliche Hardwarekonfiguration einer Plattform, eine gezielte Strategie erstellt werden, welche lediglich zwei Listen zurückliefert. Dabei ist der Anwendung selbst egal, welche und wie viel Schnittstellen verwendet werden.

3.1.2. Entwurf eines Datenbankmodells zum Speichern der Messwerte

Im vorherigen Abschnitt 3.1.1 wurde eine Architektur zum Auslesen der plattformunabhängigen Sensoren beschrieben. Resultierend daraus steht dem Programm eine Schnittstelle bereit, welche eine Reihe von Messdaten zur Verfügung stellt. Ziel des Datenbankmodells ist es, die bereitgestellten Messdaten in einem strukturierten und übersichtlichen Format abzuspeichern.

Zuordnung der Datensätze

Aus Tabelle 3.2 geht hervor, dass die Datensätze der SensorData Tabelle grundsätzlich in 3 Kategorien unterteilt werden können. Anhand der Spalte *SzUnit* können die Daten in drei Kategorien Temperatur (°C), Auslastung (%) und Events (Yes/No) unterteilt werden. Bei weiterer Betrachtung der Datensätze wird zudem ersichtlich, dass die Spalten *TReading*, *DwSensorIndex*, *DwReadingID*, *SzLabelOrig*, *SzLabelUser* und *SzUnit* bei jedem Sensor wiederholen.
Einhergehend aus dieser Beobachtung wurde die in Abbildung 3.3 abgebildete Struktur

der Datenbank erstellt.

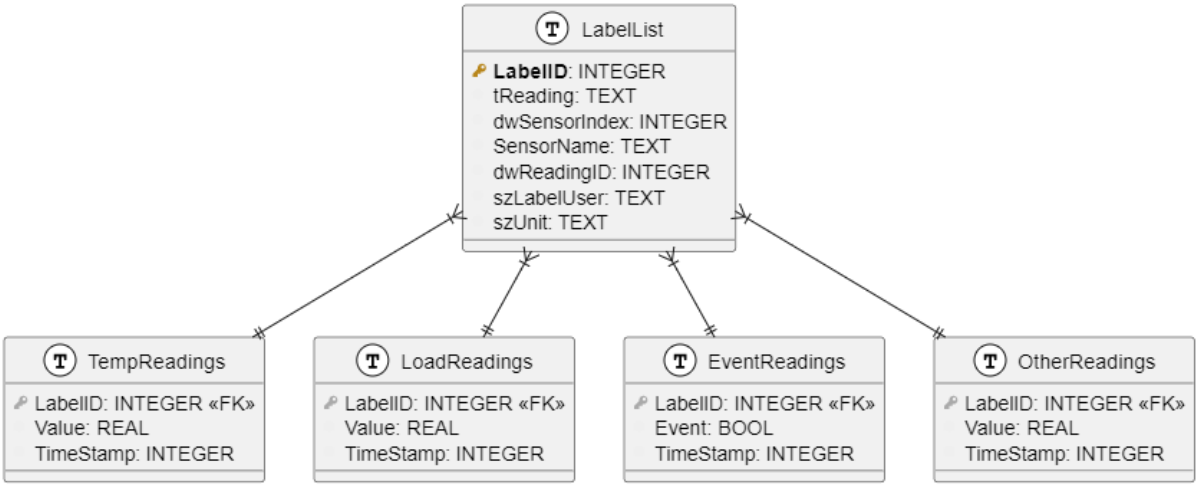

Abbildung 3.3.: Datenbankmodell des Hardware-Health-Monitorings

Die Datensätze der Tabellen 3.1 und 3.2 werden in 5 Datenbanktabellen unterteilt. Dabei werden die Felder *TReading*, *DwSensorIndex*, *DwReadingID*, *SzLabel* und *SzUnit* in die Tabelle *LabelList* ausgelagert. Hinzu kommt das Feld *SensorName*, welches aus Tabelle 3.1 stammt. Über das Feld *LabelID*, welches als primär Schlüssel konfiguriert wird. Über diesen Schlüssel können nun die eigentlichen Messwerte, den Sensoren zugeordnet werden. Die Messwerte werden dabei sortiert nach *SzUnit* in vier Tabellen geschrieben. *TempTabel* für Temperaturen in °C, *LoadReadings* für Auslastung in %, *EventTabel* für Events in 1/0 und *OtherReadings* für alle andere Einheiten. Jede der vier Tabellen verfügt über ein Feld *Value*, *TimeStamp* und einem zugehörigen Schlüssel *LabelID*.
Möchte man eine Reihe von Messwerten eines spezifischen Sensors erhalten, muss man zunächst den primär Schlüssel des Sensors aus der *LabelList* Tabelle ermitteln. Anschließend kann man die Werte der entsprechenden Tabelle erhalten.

Erweiterung der Datenbank

Zusätzlich zu den ausgelesenen Messwerten müssen auch die Ergebnisse der Systembewertung in der Datenbank abgelegt werden. Hierzu werden zunächst die Tabelle *SystemStatus* und *SystemMTBF* erstellt. Diese sind identisch und unterscheiden sich lediglich in der Bedeutung ihrer Werte. Der ermittelte Systemzustand wird in die Feldern *Status*, *Score* und *Value* geschrieben. Die Felder *TimeStampStart* und *TimeStampEnd* grenzen das Intervall ein, in welchem der Systemzustand ermittelt wurde. Des Weiteren werden die Tabellen

HistorySystemStatus und *SystemReliability* erzeugt. In diese werden die historischen Werte des Systems, so wie dessen Verlässlichkeit gespeichert. (Siehe Abbildung 3.4)

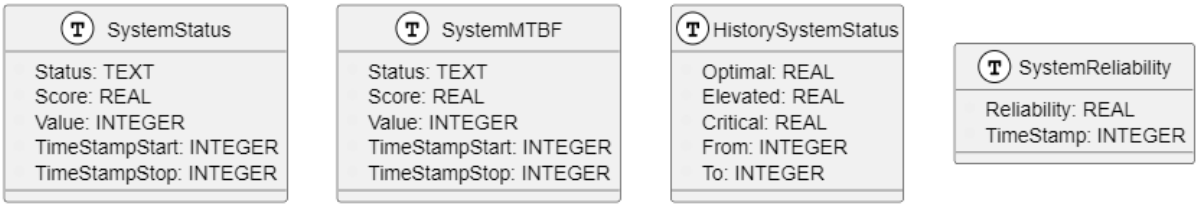

Abbildung 3.4.: Datenbankmodellerweiterung des Hardware-Health-Monitorings

Aufbau der Schnittstellen

Um die komplexe Syntax der SQLite Engine zu kapsel, wird eine Wrapperklasse benötigt, welche die wesentlichen Datenbankaufrufe handhabt. Neben den Befehlen zum Erzeugen der Datenbank, bietet die *SQLiteWrapper* Klasse auch die benötigten Funktionen um Datensätze von und in die Datenbank zu laden. Die Klasse wird in dem Verzeichnis organisiert (Siehe Abbildung 3.5).

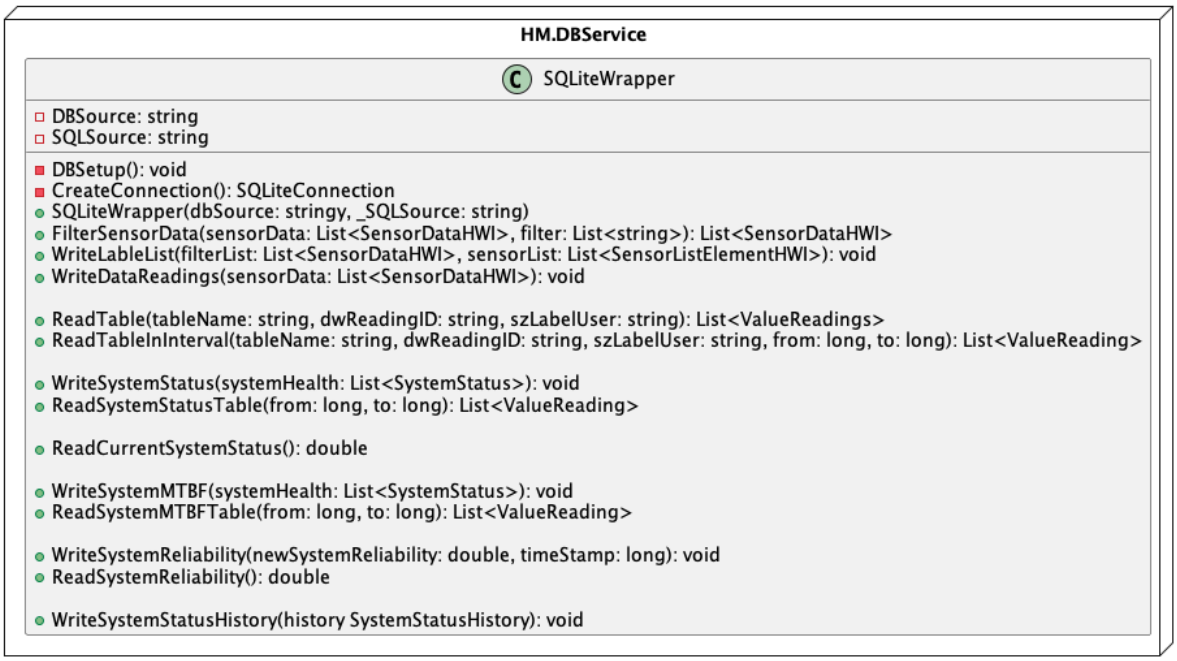

Abbildung 3.5.: Architektur des HM.DBService Verzeichnis

Des Weiteren wird das *HM.DataStrucutre.Shared.Models* Verzeichnis um die *ValueReadings* Klasse erweitert. Diese besteht aus den Parametern *Value* und *TimeStamp*. Sie dient als Schnittstelle, über welche einzelne Datensätze aus der Datenbank im Programm zwischengespeichert werden können. (Siehe Abbildung 3.6)

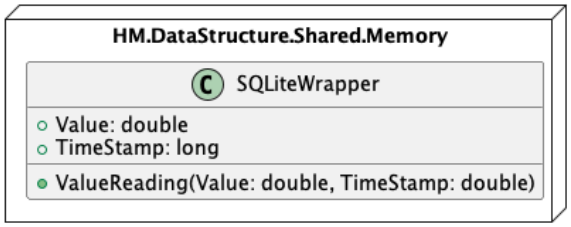

Abbildung 3.6.: Schnittstelle zum Auslesen einzelner Datensätze

3.2. Datenverarbeitung

Die Bewertung des Systemzustands erfolgt über die Ermittelung von drei zentralen Kennwerten. Zum einen muss der aktuelle Systemzustand ermittelt werden. Dieser ist eine Momentaufnahme, die innerhalb der letzten Minute erfasst wird. Er bildet das Kernelement der Bewertung. Alle weiteren Kennwerte bauen auf dem aktuellen Systemzustand auf. Im weiteren Verlauf wird dieser als *SytemStatus* betitelt. Des Weiteren wird aus den Aufzeichnungen des *SystesmStatus* der sogenannte *HistorySystemStatus* ermittelt. Dieser liefert eine historische Einsicht in das Nutzungsverhalten des Systems über eine gewisse Zeit. Zuletzt kann über den *HistorySystemStatus* die, in Abschnitt 2.4 behandelten, Systemzuverlässigkeit errechnet werden.

3.2.1. Health Status Definition

Abbildung 3.7.

Der Betrieb der Zielplattform wird zunächst in drei Zustände unterteilt. Sind alle Temperatur und die CPU Auslastung in der Norm, so läuft die Plattform im *Optimal* Bereich. Steigen Temperaturen oder CPU Auslastung, so steigt der Status auf *Elevated*. Erreicht die Temperatur nun Grenzwerte, die nicht überschritten werden dürfen, begibt sich der SystemStatus in den *Critical* Bereich.
Der *Systemstatus* wird daher in die drei Zustände *Optimal*, *Elevated* und *Critical* unterteilt. Der SystemStatus kann, wie in Abbildung 3.7, nach dem Ampelprinzip visualisiert.
Die Algorithmen zur Ermittlung dieser Werte sind unabhängig von der Hardwarekonfiguration der verschiedenen Zielplattformen. Lediglich die verwendeten Daten, die den Algorithmen zur Verfügung gestellt werden, unterscheiden sich für jede Plattform. In den folgenden Abschnitten werden die Algorithmen auf die VisuNet FLX Plattform ausgelegt. Dabei wird die Architektur so ausgelegt, dass sie für alle anderen Plattformen verwendbar ist.

Hardwarekonfiguration der VisuNet FLX Platform

Damit die Ergebnisse der Bewertungsalgorithmen möglichst genau sind, ist es wichtig, mit den richtigen Werten zu rechnen. Zum einen müssen die richtigen Temperaturgrenzen für das System festgelegt werden.

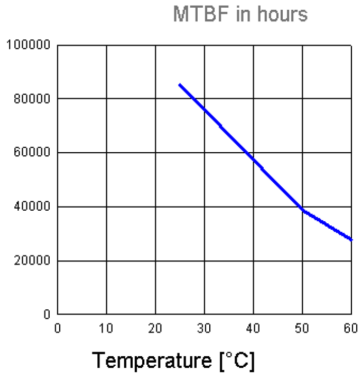

Abbildung 3.8.: MTBF Vorhersage für die VisuNet FLX Plattform

Die aus Anhang A entnommene Abbildung 3.8 stellt den MTBF, der VisuNet FLX Plattform, in Abhängigkeit zur Betriebstemperatur des Gerätes. Aus dieser werden drei Intervalle für den Betrieb des Gerätes festgelegt. Von 0°C bis 45°C läuft das Gerät im *Optimal* Betrieb. Dabei wird für dieses Intervall eine MTBF von 60000h angenommen. Im zweiten Intervall von 45°C bis 63°C läuft das Gerät im *Elevated* Betrieb. Der angenommene MTBF für dieses Intervall ist 40000h. Bei allem über 63°C läuft das Gerät im *Critical* betrieb, mit einem MTBF 30000h.

Um die MTBF Vorhersage aus Abbildung 3.8 mit den Temperaturen der Hardware verwenden zu können, müssen hierfür die richtigen Sensoren verwendet werden. Abbildung 3.9 zeigt die Sensorverteilung auf der VisuNet FLX Plattform. Da CPU und Spannungsregulator (siehe Abb. 3.9 *VR VCC Temperature* und *CPU Temperature*) zwei signifikante Hitzequellen sind, wird der Sensor *Temperatur 1* zur Bestimmung der Betriebstemperatur gewählt.

Temperatu 1 typo korregie- ren

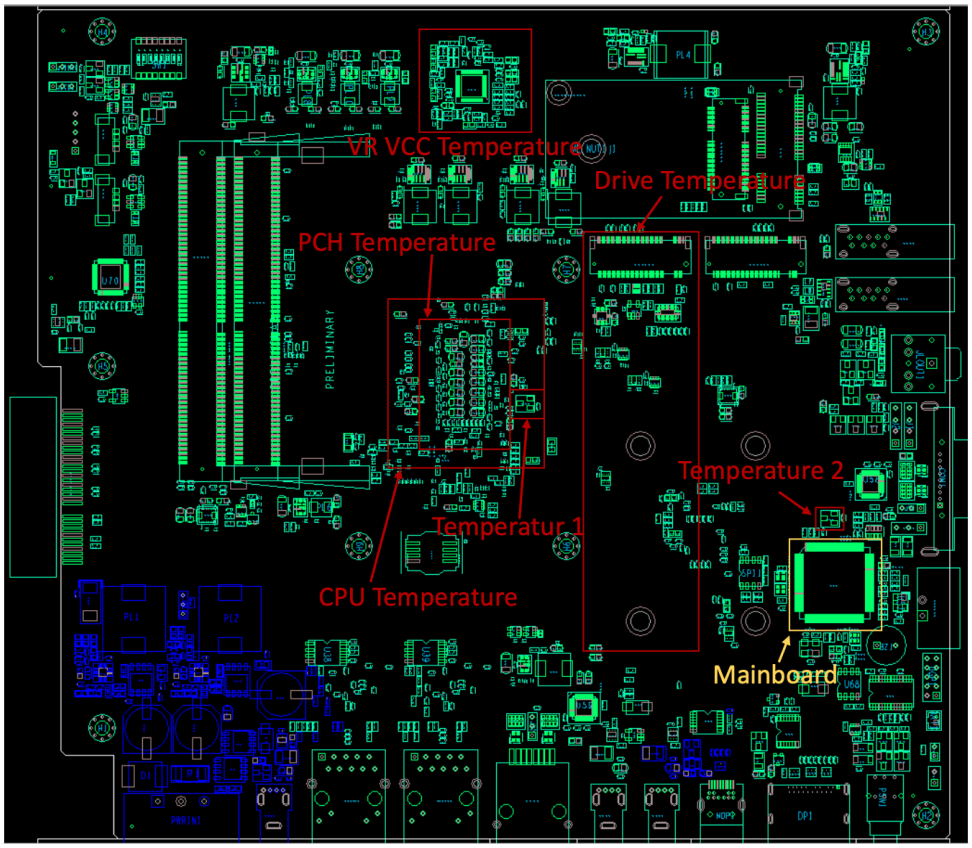

Abbildung 3.9.: Sensorverteilung auf der Mainboard Platine der VisuNet FLX Plattform

3.2.2. Auslegung einer FuzzyLogic zur Ermittelung des SystemStatus

Der *SystemStatus* wird mithilfe des, in Abschnitt 2.5 beschriebenen, FuzzyLogic Verfahren ermittelt. Das System soll dabei so ausgelegt werden, dass der *SystemStatus* von Temperatur und CPU-Last abhängt.
Im ersten Schritt müssen daher zunächst die linguistischen Variablen, welche die Ein- und Ausgänge der FuzzyEngine repräsentieren, definiert werden. Zudem müssen auch die passenden Zugehörigkeitsfunktionen definiert werden. Diese sind mathematische Funktionen, welche angeben, wie stark ein Wert zu einem bestimmten linguistischen Begriff gehört. [14]

Definition der linguistischen Variablen und der Zugehörigkeitsfunktionen

Die Temperaturen werden der linguistischen Variable *Temperature* zugeordnet. Zudem werden drei Zugehörigkeitsfunktionen definiert. Diese sind in Abbildung 3.10 abgebildet. Dabei werden die Grenzen der Funktionen an die in Abschnitt 3.2.1 aufgestellten Temperaturintervalle angepasst. Es werden insgesamt drei trapezförmige Funktionen erstellt. Die Funktionen werden mit den Begriffen *Low*, *Medium* und *High* bezeichnet.

richtiges bild reinmachen

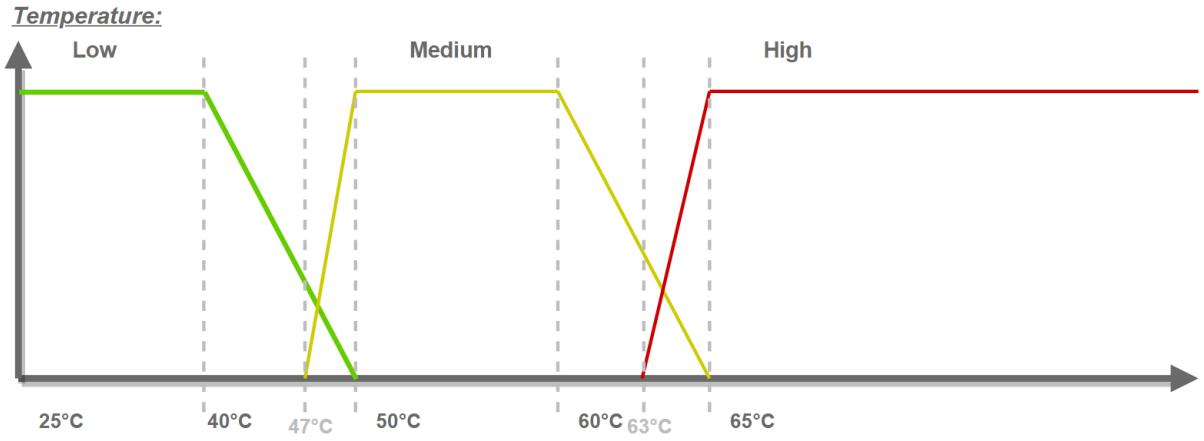

Abbildung 3.10.: Zugehörigkeitsfunktionen der Temperature Variable

Die Auslastung der CPU wird der linguistischen Variable *CPULoad* zugeordnet. Wie auch bei den Zugehörigkeitsfunktionen der Temperatur werden hierfür ebenfalls drei trapezför-

mige Funktionen aufgestellt. Diese werden ebenfalls mit den Begriffen *Low*, *Medium* und *High* bezeichnet. Die Zugehörigkeitsfunktionen Funktionen der Variable *CPULoad* werden in Abbildung 3.11 dargestellt.

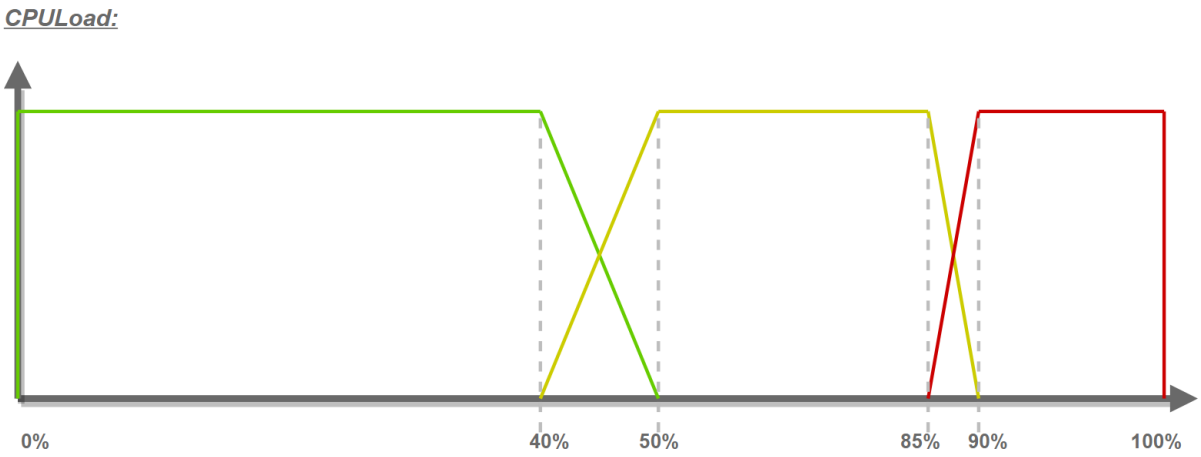

Abbildung 3.11.: Zugehörigkeitsfunktionen der CPULast Variable

Der Ausgang der Engine wird als die linguistische Variable *SystemStatus* deklariert. Hierzu werden drei Rechteckfunktionen *Optimal*, *Elevated* und *Critial* erzeugt. Das Ergebnis wird in % angegeben, wobei jede Funktion genau ein Drittel darstellt. Die Funktionen der *SystemStaus* Variable sind in Abbildung 3.12 dargestellt.

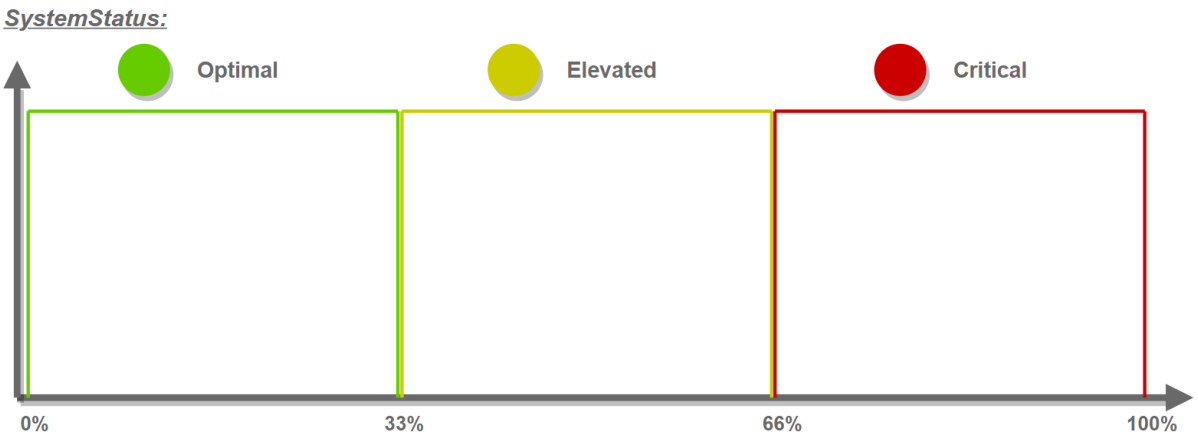

Abbildung 3.12.: Zugehörigkeitsfunktionen der SystemStatus Variable

Definition der Regeln

Im zweiten Schritt beim Auslegen eines FuzzyLogic Systems muss das Regelwerk festgelegt werden. Die Regeln eines solchen Systems sind dabei verbal gefasst und drücken die Beziehung zwischen den linguistischen Variablen der Ein- und Ausgänge aus.[18]
Die Ausgangsvariablen *SystemStatus* wird in Abhängigkeit von den Eingangsvariablen *Temperature* und *CPLoad* gesetzt. Dabei werden die möglichen Zustände der Variable, die in Abbildung 3.12 gezeigt werden, wie folgt definiert.

1. *SystemStatus: Optimal*
   Das System operiert im *Optimal* Bereich, solange die Temperatur des Systems in den Wertebereich der Zugehörigkeitsfunktion *Low* fällt. Zudem muss die Auslastung der CPU in im Wertebereich Zugehörigkeitsfunktion *Low* oder *Medium* liefen.

2. *SystemStatus: Elevated*
   Das System operiert im *Elevated* bereich, sobald die Temperatur des Systems in den Wertebereich der Zugehörigkeitsfunktion *Medium* oder die Auslastung der CPU in den Wertebereich der Zugehörigkeitsfunktion *High* fällt.

3. *SystemStatus: Critical*
   Das System operiert im *Critical* bereich, wenn die Temperatur des Systems in den Wertebereich der Zugehörigkeitsfunktion *High* fällt. Dies bleibt unabhängig von der CPU Auslastung des Systems.

Aus dieser Zustandsdefinition kann somit das, in Abschnitt 3.13 gezeigte, Regelwerk zur Ermittelung des Systemzustands abgeleitet werden.

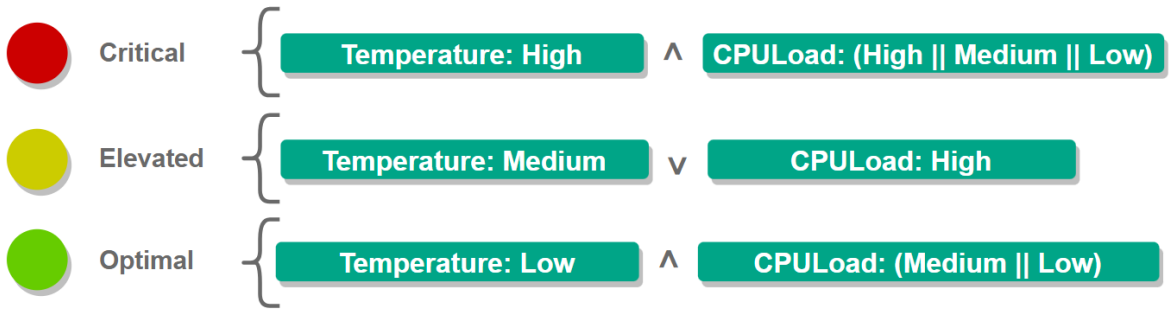

Abbildung 3.13.: Regelwerk zum Ermitteln des Systemstauts

Architektur

Da die Basis der FuzzyLogic für jede Zielplattform dieselbe ist, wird zunächst über das *IFuzzyLogic* Interface, die Schnittstelle zur FuzzyEngin definiert. Hier raus wird anschließend die abstrakte Basisklasse *FuzzyLogic* abgeleitet werden. Diese beinhaltet die verwendeten linguistischen Variablen, die jeweiligen Zugehörigkeitsfunktionen, so wie die Methoden zur Berechnung der Ergebnisse.
Aus der *FuzzyLogic* Klasse können anschließend weitere Klassen abgeleitet werden, in denen die gewünschten Zugehörigkeitsfunktionen für die jeweiligen Zielplattformen angelegt werden können. Das in der Basisklasse definierte Regelwerk kann bei Bedarf in der abgeleiteten Klasse überschrieben werden. Die Struktur des *HM.ScoringModel* Verzeichnisses wird in Abbildung 3.14 visualisiert.

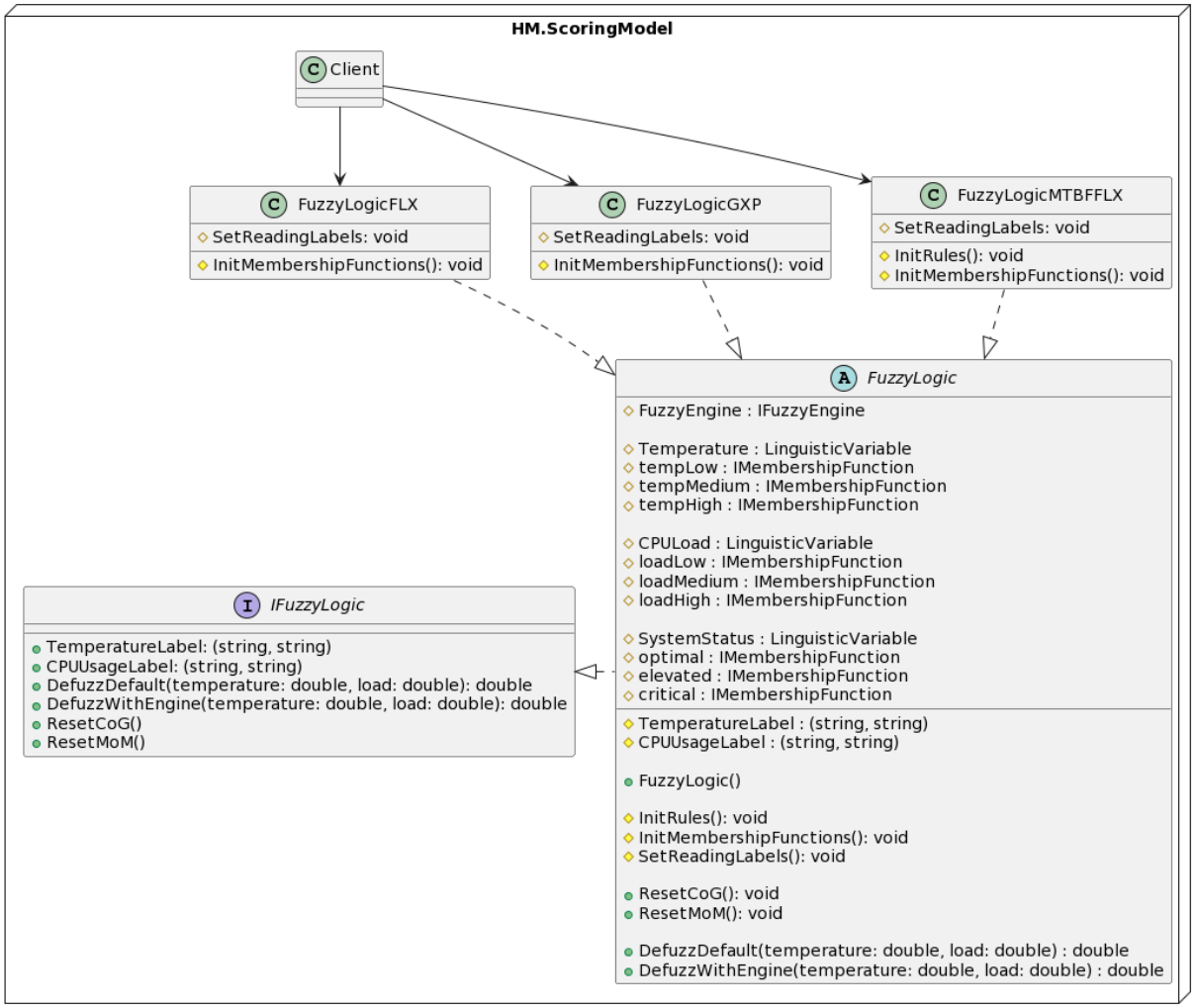

Abbildung 3.14.: Architektur des HM.ScoringModel für die Ermittelung des SystemStatus

3.2.3. Ermittelung der Systemstatus Historie und der Systemzuverlässigkeit

Um die Systemstatus Historie des Systemes zu berechnen, werden die Aufzeichnungen des Systemstatus benötigt. Diese müssen im gewünschten Intervall, beispielsweise der letzten 24h, gegeben sein. Die Idee besteht im Wesentlichen daraus, die Zeit zwischen zwei Systemzuständen, für jeden Zustand *Optimal*, *Elevated* und *Critacal* zu addieren. Somit kommt man auf die Zeiten, in dem das System im jeweiligen Zustand verbracht hat. Anschließend können die einzelnen Zeiten durch die gesamte Zeit des Intervalls geteilt werden, um auf die prozentualen Anteile der jeweiligen Zustände zu kommen.
Hierzu visualisiert das in Abbildung 3.15 visualisierte Ablaufdiagramm, die Ermittlung der Systemzustands Historie. Die Systemstatus Historie wird in der *HistorySystemStatus* Tabelle der Datenbank abgespeichert. (Siehe Abbildung 3.4)

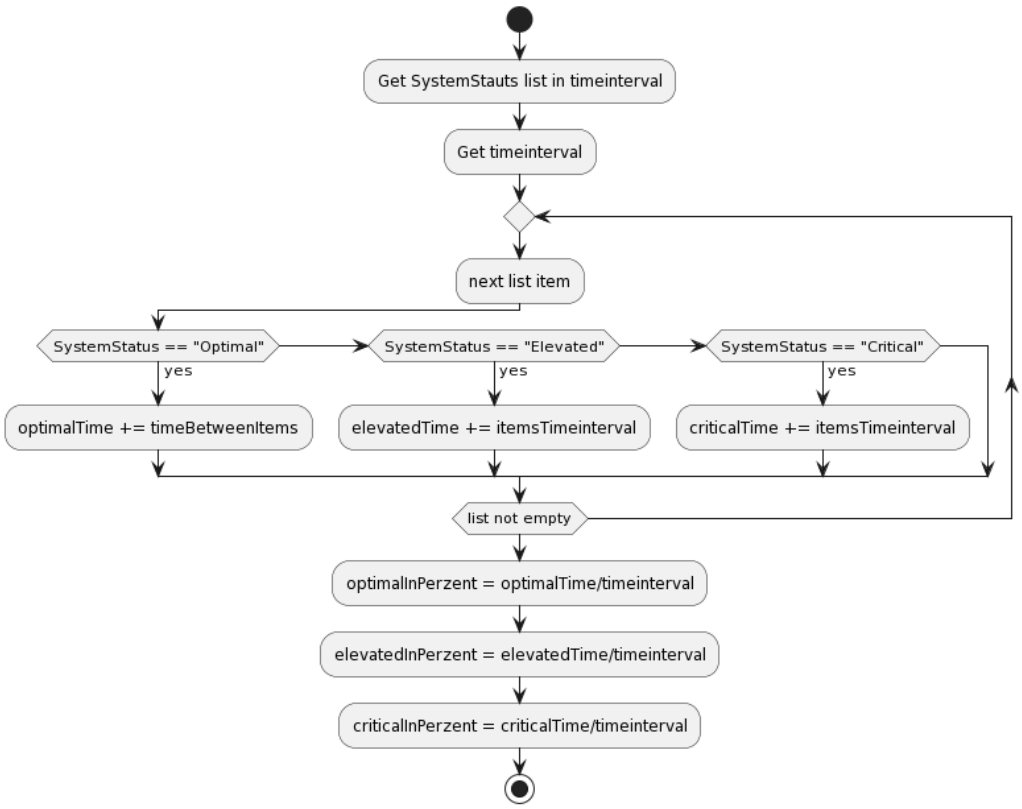

Abbildung 3.15.: Algorithmus zur Berechnung der Systemstatus Historie

Über die in Abschnitt 2.4 beschrieben Formel 2.1 kann die Reliability (Zuverlässigkeit) des Systems berechnet werden. Hierzu wird zum einen die Laufzeit des Systems, zum anderen der temperaturabhängige MTBF des Systems benötigt. Jedoch kann diese Formel nicht direkt angewendet werden. Zum einen gibt es keine direkte Aufzeichnung über die gesamte Laufzeit des Systems. Zum anderen variiert die Betriebstemperatur des Systems, sodass man für verschiedene Betriebstemperaturen verschiedene MTBF Werte verwenden muss.
Abhilfe hierzu schaft der Algorithmus zur Kalkulation der Systemstatus Historie. Die Idee hinter diesem Ansatz besteht darin, die unterschiedlichen Betriebstemperaturen mithilfe der FuzzyLogic Klasse und einem angepassten Regelwerk aufzuzeichnen. Anschließend kann hierzu die Systemstatus Historie berechnet werden. Das Ergebnis liefert somit eine genaue Aussage darüber, wie lange das System in welchem Temperaturbereich gearbeitet hat. Somit bekommt man zum einen die Systemlaufzeit, zum anderen den zugehörigen temperaturabhängigen MTBF zur Berechnung der Systemzuverlässigkeit.
Um bei der Berechnung der Systemzuverlässigkeit nicht jedes Mal alle Werten zu benutzen, wird diese zudem nur für eine kleine Zeitspanne, beispielsweise alle 24h, erneut durchgeführt. Das Ergebnis kann anschließend mit dem Ergebnis der vorherigen Berechnung verrechnet werden, um auf die aktuelle Systemzuverlässigkeit zu kommen.
Um die Berechnung durchführen zu können, muss daher zunächst das Regelwerk der FuzzyEngin angepasst werden. Hierzu wird die in Abbildung 2.8 abgebildete Klasse *FuzzyLogicMTBFFLX* mit dem in Abbildung 3.16 abgebildeten Regelwerk angelegt. Der Ausgang *SystemStatus* so wie der Eingang *Temperature*, so wie die korrespondierenden Zugehörigkeitsfunktionen bleiben dabei unverändert. (Siehe Abbildung 3.12 und 3.10). Eingang *CPULoad* fällt weg. Der Zustand des Eingangs wird eins zu eins auf den Ausgang übersetzt. Abbildung 3.16 veranschaulicht das angepasste Regelwerk der *SuzzyLogicMTBF* Klasse.

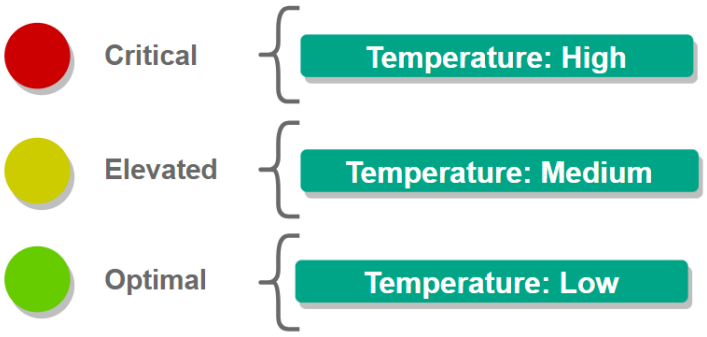

Abbildung 3.16.: Regelwerk SystemMTBF

Das zuvor beschriebene Verfahren zur Ermittelung der Systemzuverlässigkeit wird in Abbildung 3.17 visualisiert.

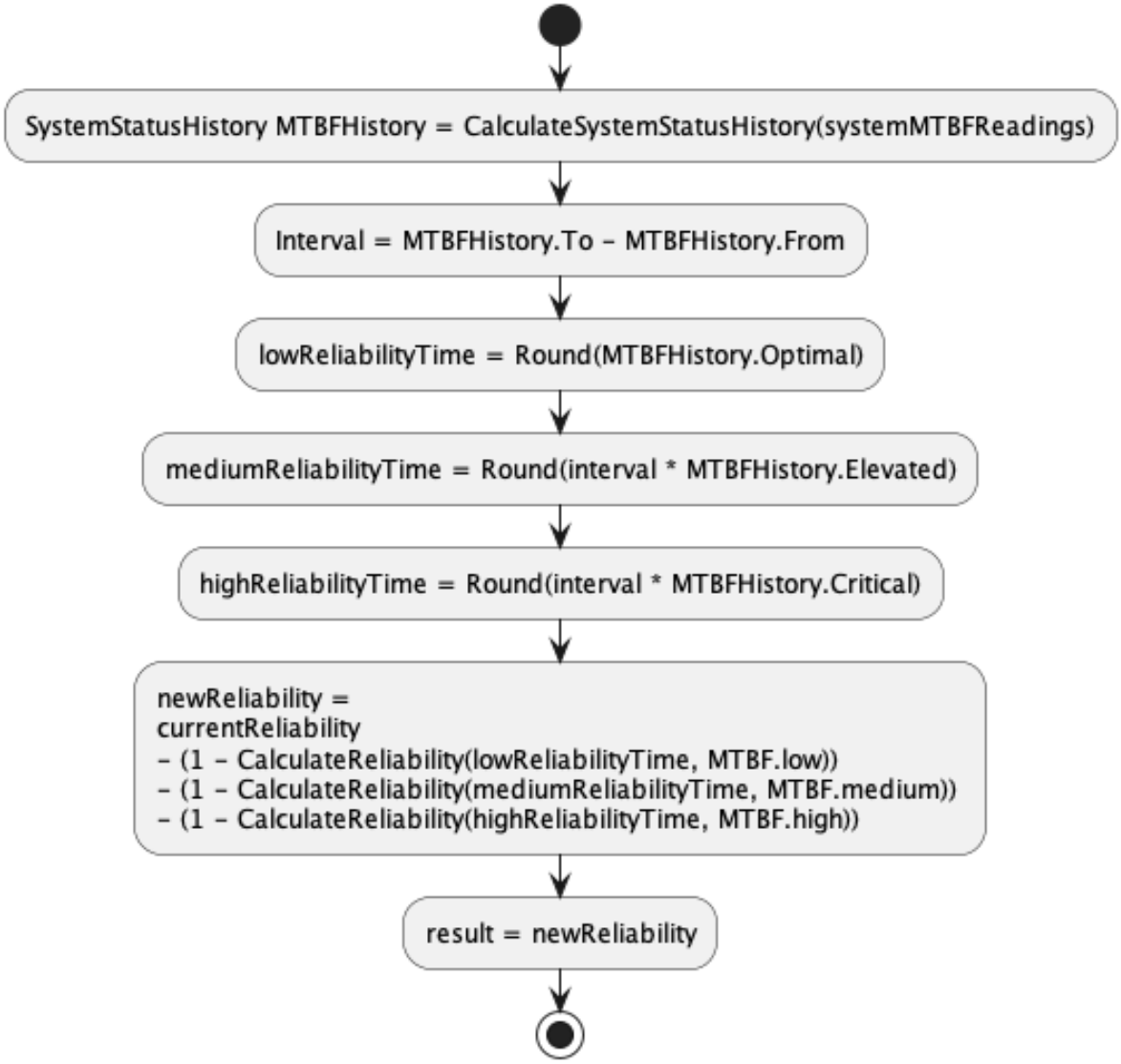

Abbildung 3.17.: Algorithmus zur Berechnung der Systemzuverlässigkeit

Klassenarchitektur

Da die Algorithmen zum Berechnen der Systemstatus Historie so wie der Reliability des Systems unabhängig von Hardwarekonfiguration sind, wird das Verzeichnis *HM.ScoringModel*

um die, in Abbildung 3.18 abgebildete, *StatusHistory* Klasse erweitert. Dieses bietet zwei Funktionen, mit welchen die Systemzuverlässigkeit so wie die Systemstatus Historie berechnet werden kann. Da lediglich die temperaturabhängigen MTBF Werte der Zielplattformen sich unterscheiden können, sollen diese beim Aufruf der Funktion *CalculateCurrentReliability* übergeben.

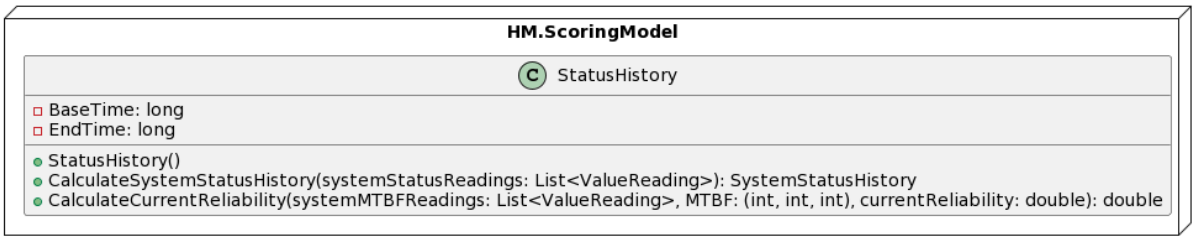

Abbildung 3.18.: Wrapperklasse zur Berechnung von Reliability und der Systemstatus Historie

3.3. Entwurf eines Task Scheduling Verfahrens

Um die in Abschnitt 3.1 und 3.2 beschriebenen Teilsysteme zu einem lauffähigen Programm zusammenzuführen, wird ein Konzept benötigt, das es erlaubt, einzelne Aufgaben in regelmäßigen Intervallen auszuführen. Dieses Konzept kann durch die Verwendung eines TaskSchedulers realisiert werden, der die zeitliche Planung und Ausführung der einzelnen Aufgaben gemäß den festgelegten Intervallen steuern und verwalten kann. Hierzu werden unter dem *HM.TimerBasedScheduler* Verzeichnis folgende Klassen angelegt. (Siehe Abbildung 3.19)

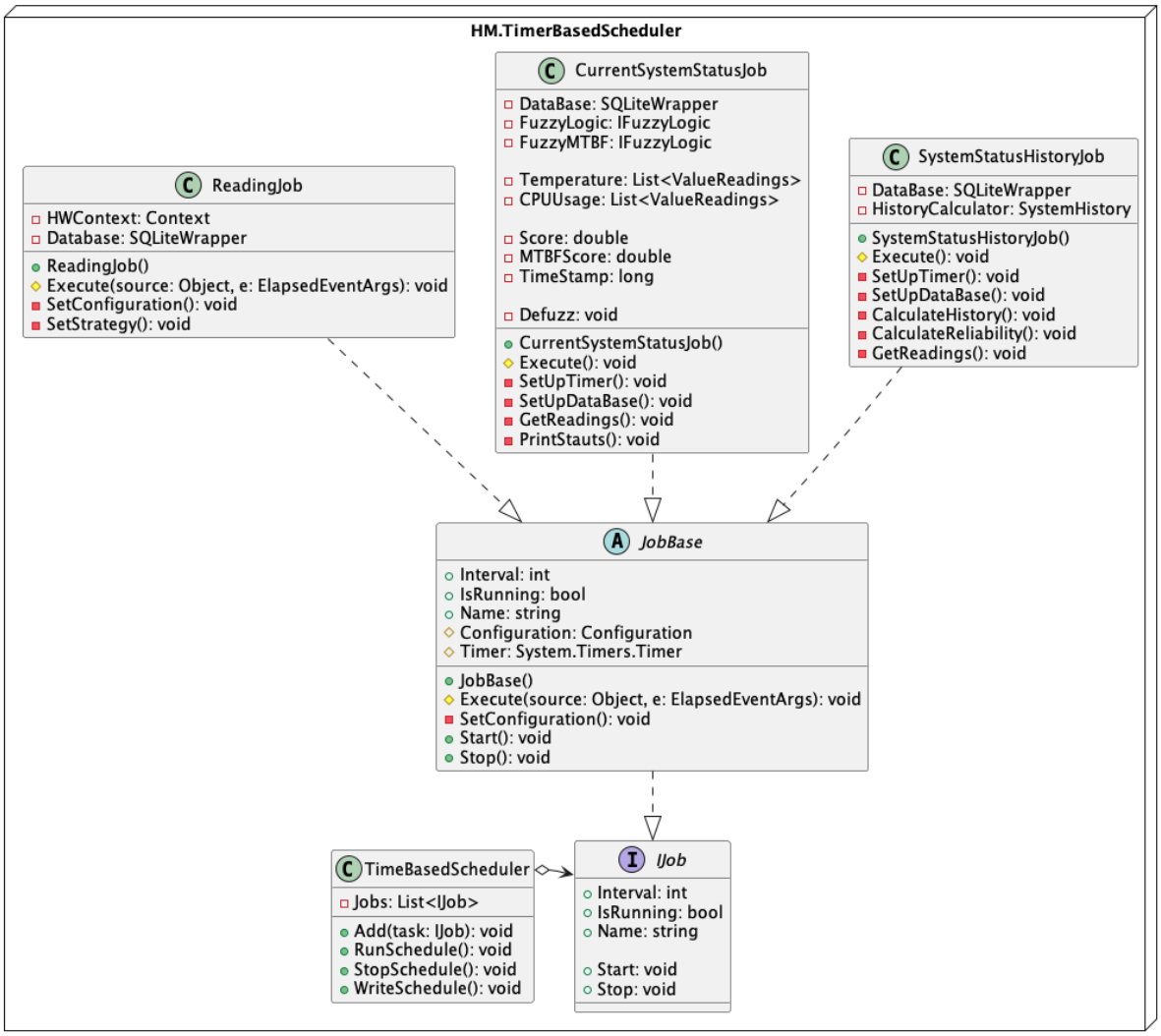

Abbildung 3.19.: Architektur des Taskscheduler

Die Basis des Schedulers ist die *TimerBasedScheduler* Klasse. In dieser können verschiedene Aufgaben zum Zeitplan hinzugefügt werden. Zudem kann der Scheduler in dieser Klasse gestartet und gestoppt werden. Um zur Laufzeit des Programms eine Übersicht über die laufenden Aufgaben zu haben, wird der *TimerBasedScheduler* Klasse eine Funktion zur Visualisierung des Zeitplans in der Konsole hinzugefügt.
Die Schnittstelle zu einer bestimmten Aufgabe, im Weiteren als Job bezeichnet, wird über das *IJob* Interface definiert. Dieses definiert die von der *TimerBasedScheduler* Klasse verwendeten Funktionen und Variablen. Aus diesem wird anschließend die abstrakte *JobBase* Klasse abgeleitet. Die *JobBase* Klasse liefert alle Basisfunktionen und Variablen, welche in jedem Job verwendet werden. Die Verwendung einer Basisklasse verhindert Codeduplikation und erleichtert durch eine vordefinierte Struktur das Programmieren weiterer spezifischen Jobs.
Im letzten Schritt wird eine Reihe von Jobs abgeleitet, welche die in gewünschten Aufgaben ausführen. Hierzu werden drei Klassen aus der *JobBase* Klasse abgeleitet.

- *ReadingJob*
  Aufgabe der *ReadingJob* Klasse ist die Auswertung der Hardwarespezifischen Sensoren. Hierzu bedient sich die Klasse dem in Abschnitt 3.1.1 beschriebenen *HM.HWServices* und dessen Strategieklassen. Zudem verwendete diese Klasse die, in Abschnitt 3.1.2 beschriebene, Datenbank Schnittstelle zum Schreiben der Messwerte.

- *CurrentSystemStatusJob*
  Aufgabe der *CurrentSystemStatusJob* Klasse ist die Ermittelung des aktuellen Systemstatus. Hierzu verwendet die Klasse das, in Abschnitt 3.2.2 beschriebene, *HM.ScoringModel* Verzeichnis und dessen *FuzzyLogic* Klassen. Des Weiteren bedient sich auch diese Klasse der in Abschnitt 3.1.2 beschriebene Datenbank Schnittstelle.

- *SystemStatusHistoryJob*
  Aufgabe der *SystemStatusHistoryJob* Klasse ist die Ermittelung der Systemstatus Historie so wie der Systemzuverlässigkeit. Hierzu verwendet die Klasse das, in Abschnitt 3.2.3 beschriebene *HM.ScoringModel* Verzeichnis und dessen *StatusHistory* Klasse. Des Weiteren bedient sich auch diese Klasse der in Abschnitt 3.1.2 beschriebene Datenbank Schnittstelle. Des Weiteren bedient sich auch diese Klasse der in Abschnitt 3.1.2 beschriebene Datenbank Schnittstelle.

3.4. Datenvisualisierung

Die Visualisierung der aufgezeichneten Messwerte, so wie der ermittelten Zustandsbewertung des Systems soll über die Open-Source-Software Grafana erfolgen. Die Visualisierung der Daten soll hierbei nicht an die Zielplattform gebunden sein und von jedem beliebigen Rechner Abrufbar sein. Desweiteren soll die möglichkeit geboten sein, die Werte verschiedener Platformen in einem gemeinsamen Dashboard anzeigen zu lassen.
Das Konzept für eine Platformunabhängige Visualisierung der Hardware-Health-Monitoring Daten wird in Abbildung 3.20. Hierbei werden die Daten der eizelnen Zielplattform einem Zentralen Server über eine REST API in der Anwendung bereitgestellt. Der Grafana Service wird anschlißend von einem seperaten Server dem anwender bereitgestellt.

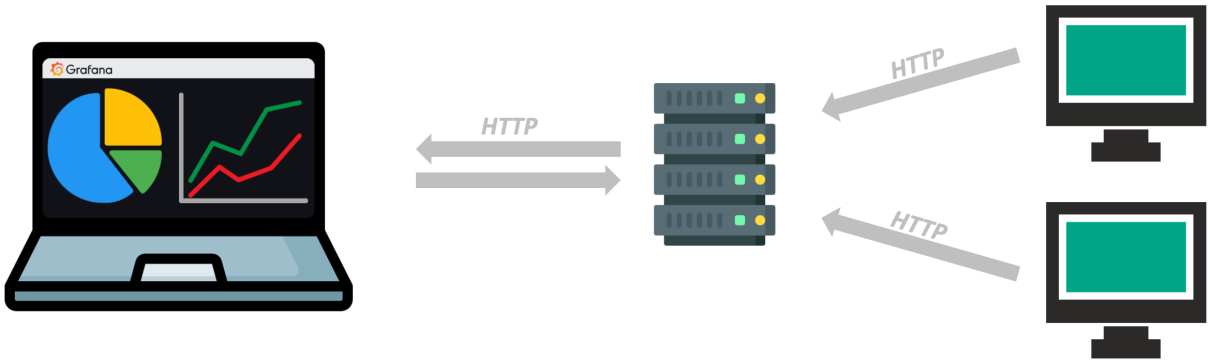

Abbildung 3.20.: Konzept zur Platformunabhängigen Datenvisualisierung

3.4.1. Architektur der REST API

Die in Grafana verwendete Datenquelle *SimpleJSON* diktiert die Struktur der API vor. In der Dokumentation der Datenquelle [19] werden die URL Pfade so wie Body der HTTP Anfragen erläutert.
Hierbei können, nach Abbildung 3.21, die HTTP Anfragen an drei URL Pfade gesendet werden. Die URL mit dem Ende „/“ wird zum Testen der Verbindung genutzt. Anfragen an die URL „*/search*“ liefern eine Liste an vorhandenen Datensätze zurück. Diese wird genutzt um einen bestimmten Datensatz mit dem nächsten Aufruf der API anzufragen. Über eine Anfragen an „*/query*“ können die gewünschten Datensätze erhalten werden.

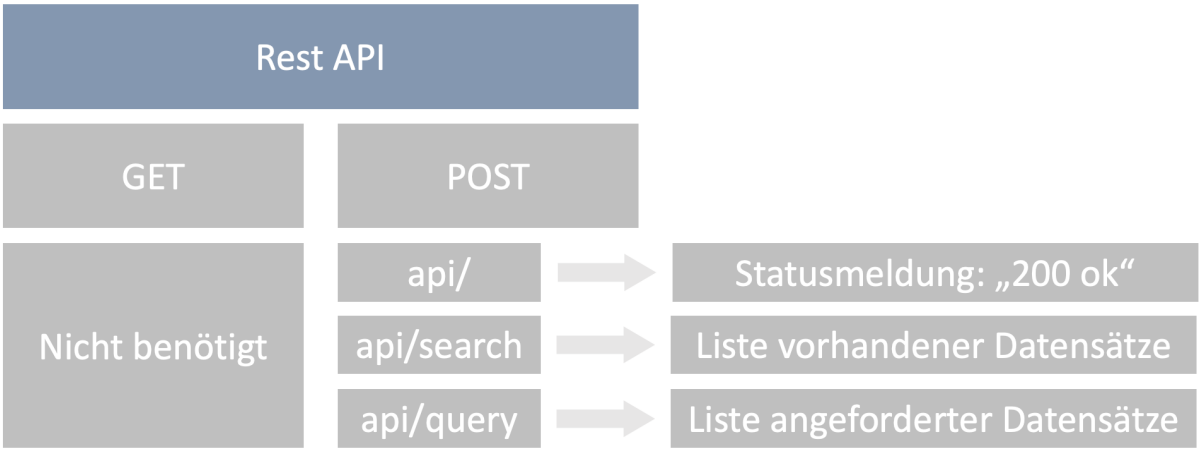

Abbildung 3.21.: Visualisierung der HTTP Anfragen an die API

Da die Anfragen von Grafana in zufälligen intervallen gestellt werden, muss die API zu einem beliebigen zeitpunkt auf diese reagieren können. Um wärend der behandlung einer Anfrage, nicht das gesamte Program lahm zu legen, wird die API des Hardware Health Monitors in einem seperaten Thread gestartet und läuft unabhängig vom in Abschnitt 3.3 behandelten Taskscheduler.
Die API wird in der Architektur der gesamten Anwendung von einer einfachen Klasse, welche unter dem Verzeichnis *HM.API* organisiert ist, representiert. Abbildung 3.22 zeigt zudem die verwendeten Resoursen der Schnittstelle.

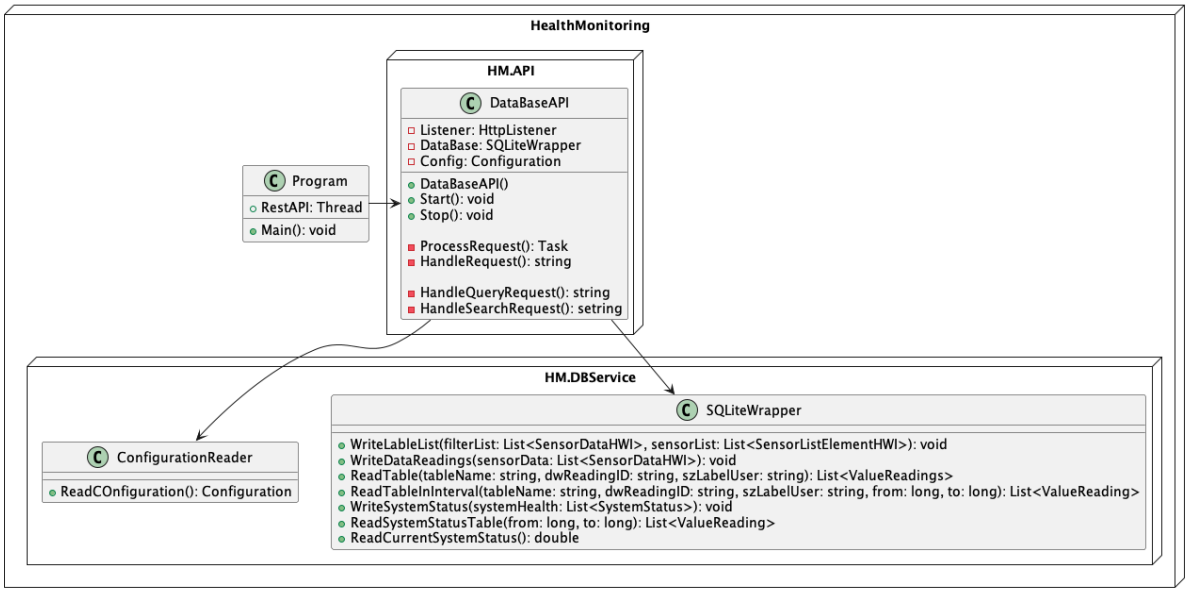

Abbildung 3.22.: Arichtektur der API

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).

[18] „LabVIEW PID and Fuzzy Logic Toolkit API Reference“. (2023), Adresse: %7Bhttps://www.ni.com/docs/de-DE/bundle/labview-pid-and-fuzzy-logic-toolkit-api-ref/page/lvpidmain/creating_rulebase.html%7D (besucht am 25. 08. 2023).

[19] „Simple JSON Datasource - a generic backend datasource“. (2023), Adresse: %7Bhttps://grafana.com/grafana/plugins/grafana-simple-json-datasource/%7D (besucht am 20. 08. 2023).

ARBOR®
Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

Model Name: ELIT-3922

Prediction equipment / method: CARE® BQR Ver 2.2 / Bellcore TR-332 Issue 6

Prediction result:

MTBF	25DC	85047.8	hours
	60DC	27602.2	hours
Failure rate	25DC	11758.09	FITs
	60DC	36228.99	FITs

Environmental conditions:

- GB (Ground, Fixed, Controlled)
- Temperature : 60 degree C
- device quality level: 1
- Stress %: 50%

*Explanatory notes
MTBF: Mean time between failures
FITs: Failures in 10^9 hours

Vince	***Vince***	***Mike Chen***
QA Manager	***Check by***	***Written by***

Rev: 0 *Page 1*

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

CARE(R) BQR , Ver.2.2

MTBF Components and Blocks Report

Project: FMB-i90Z2-3965U-PE , Condition: 25DC
File: FMB-i90Z2-3965U-PE_25DC_MC.htm , Date: 07/25/23 , Time: 10:26:01

	MTBF(HRS)	FR (F/10^6HRS)
Predicted	85047.76	11.7581

Level	Blk#	Family	RefDes	PART Number	Fail.Rate	Stress	Env	Temp	Source	Qty
0.....	1	Block	SYSTEM	SCDB-147G	11.7581	-	GB	25.0	B332I6	1
.1.....	1	Connector	BAT1	1*2	0.0006	-	-	25.0	B332I6	1
.1.....	1	Miscellaneous	BAT2	BR2450A	0.4500	-	-	25.0	B332I6	1
.1.....	1	Capacitor	BC1	0.1uf-C	0.0027	V=50.0%	-	25.0	B332I6	223
.1.....	1	Inductive	BP1	BEAD60ohm	0.0159	-	-	25.0	B332I6	4
.1.....	1	Inductive	BZ1	BUZZER	0.0159	-	-	25.0	B332I6	1
.1.....	1	Capacitor	C1	1000pf-C	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C11	1000pf-C	0.0027	V=50.0%	-	25.0	B332I6	4
.1.....	1	Capacitor	C114	20pf-C	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C116	22pf-C	0.0027	V=50.0%	-	25.0	B332I6	3
.1.....	1	Capacitor	C118	47uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	16
.1.....	1	Capacitor	C13	6800pf-C	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C134	2.2uf-C/25V	0.0027	V=50.0%	-	25.0	B332I6	12
.1.....	1	Capacitor	C138	0.022uf-C/25V	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C143	0.022uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C145	0.039uf-C/16V	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C151	10uf-C/10V	0.0027	V=50.0%	-	25.0	B332I6	13
.1.....	1	Capacitor	C162	22uf-C/16V	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C166	22uf/6.3V	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C168	0.01uf-C	0.0027	V=50.0%	-	25.0	B332I6	14
.1.....	1	Capacitor	C17	1uf-C/25V	0.0027	V=50.0%	-	25.0	B332I6	10

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Capacitor	C2	0.1uf-C	0.0027	V=50.0%	-	25.0	B332I6	4
.1.....	1	Capacitor	C213	3300pf-C	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C216	2.2uf-C/16V	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C227	10pf-C/50	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C242	1uf-C	0.0027	V=50.0%	-	25.0	B332I6	5
.1.....	1	Capacitor	C250	4.7uf-C/25V	0.0027	V=50.0%	-	25.0	B332I6	3
.1.....	1	Capacitor	C31	150pf-C	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C32	22uf-C/10V	0.0027	V=50.0%	-	25.0	B332I6	6
.1.....	1	Capacitor	C36	4.7uf-C/10V	0.0027	V=50.0%	-	25.0	B332I6	3
.1.....	1	Capacitor	C37	2200pf-C	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C4	4.7uf-C/50V	0.0027	V=50.0%	-	25.0	B332I6	15
.1.....	1	Capacitor	C42	33pf-C	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C45	10uf-C/10V	0.0027	V=50.0%	-	25.0	B332I6	11
.1.....	1	Capacitor	C46	0.01uf-C/10V	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C47	0.22uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	3
.1.....	1	Capacitor	C50	0.22uf-C/16V	0.0027	V=50.0%	-	25.0	B332I6	33
.1.....	1	Capacitor	C58	470pf-C	0.0027	V=50.0%	-	25.0	B332I6	6
.1.....	1	Capacitor	C6	4.7uf-C/16V	0.0027	V=50.0%	-	25.0	B332I6	5
.1.....	1	Capacitor	C62	220pf-C	0.0027	V=50.0%	-	25.0	B332I6	3
.1.....	1	Capacitor	C65	1500pf-C	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C75	560pf-C	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C77	82pf-C	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C79	27pf-C	0.0027	V=50.0%	-	25.0	B332I6	7
.1.....	1	Capacitor	C80	100pf-C	0.0027	V=50.0%	-	25.0	B332I6	5
.1.....	1	Capacitor	C81	68pf-C	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	C84	330pf-C	0.0027	V=50.0%	-	25.0	B332I6	4
.1.....	1	Capacitor	C88	56pf-C	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	C9	10uf-C/25V	0.0027	V=50.0%	-	25.0	B332I6	26
.1.....	1	Capacitor	CA1	0.1uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	41
.1.....	1	Capacitor	CB1	10uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	121
.1.....	1	Capacitor	CC1	1uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	98

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Capacitor	CD1	22uf-C/6.3V	0.0213	V=50.0%	-	25.0	B332I6	96
.1.....	1	Capacitor	CD98	22uf-C/10V	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	CE1	150uf-A/35V	0.0213	V=50.0%	-	25.0	B332I6	3
.1.....	1	Capacitor	CM1	1uf-C/16V	0.0027	V=50.0%	-	25.0	B332I6	81
.1.....	1	Connector	CN1	1*4PIN	0.0011	-	-	25.0	B332I6	2
.1.....	1	Connector	CN2	1*4PIN	0.0011	-	-	25.0	B332I6	1
.1.....	1	Connector	COM1	DSUB	0.0043	-	-	25.0	B332I6	1
.1.....	1	Connector	COM2	RJ45	0.0005	-	-	25.0	B332I6	1
.1.....	1	Capacitor	CP1	100pf-8P4C	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	CP2	180pf-8P4C	0.0027	V=50.0%	-	25.0	B332I6	4
.1.....	1	IC	CPU1	Intel Celeron 3965U	0.0249	-	-	25.0	B332I6	1
.1.....	1	Capacitor	CS1	100uf-C/16V	0.0027	V=50.0%	-	25.0	B332I6	1
.1.....	1	Capacitor	CS14	220uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	6
.1.....	1	Capacitor	CS6	220uf-C/6.3V	0.0027	V=50.0%	-	25.0	B332I6	2
.1.....	1	Capacitor	CS8	270uf-C/2V	0.0027	V=50.0%	-	25.0	B332I6	5
.1.....	1	Diode	D1	5.0SMDJ36CA	0.0204	P=50.0%	-	25.0	B332I6	1
.1.....	1	Diode	D10	BAT54A	0.0204	P=50.0%	-	25.0	B332I6	1
.1.....	1	Diode	D11	BAT54C	0.0179	If=50.0%	-	25.0	B332I6	1
.1.....	1	Diode	D12	BAT54HT1G	0.0179	If=50.0%	-	25.0	B332I6	1
.1.....	1	Diode	D13	S9314-101B	0.0179	If=50.0%	-	25.0	B332I6	2
.1.....	1	Diode	D2	SK34A	0.0179	If=50.0%	-	25.0	B332I6	1
.1.....	1	Diode	D3	BDC400201	0.0204	P=50.0%	-	25.0	B332I6	4
.1.....	1	Diode	D8	ZENER MMSZ5230A	0.0204	P=50.0%	-	25.0	B332I6	2
.1.....	1	Connector	DGP1	2*5PIN	0.0028	-	-	25.0	B332I6	1
.1.....	1	Connector	DIMM1	DIMM	0.0410	-	-	25.0	B332I6	1
.1.....	1	Connector	DIMM2	DIMM	0.0410	-	-	25.0	B332I6	1
.1.....	1	Connector	DP1	DISPLAY	0.0057	-	-	25.0	B332I6	1
.1.....	1	Diode	ESD1	ESD	0.0179	If=50.0%	-	25.0	B332I6	4
.1.....	1	Diode	ESD11	AZ5125	0.0179	If=50.0%	-	25.0	B332I6	3
.1.....	1	Diode	ESD14	AZ1065	0.0179	If=50.0%	-	25.0	B332I6	2
.1.....	1	Diode	ESD16	AZC199-04S	0.0179	If=50.0%	-	25.0	B332I6	6

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Diode	ESD5	ESD	0.0179	If=50.0%	-	25.0	B332I6	6
.1.....	1	Fuse	F1	FUSE	0.0150	-	-	25.0	B332I6	2
.1.....	1	Fuse	F3	FUSE	0.0150	-	-	25.0	B332I6	2
.1.....	1	Inductive	FB1	BEAD30ohm	0.0159	-	-	25.0	B332I6	18
.1.....	1	Inductive	FB18	BEAD60ohm	0.0159	-	-	25.0	B332I6	2
.1.....	1	Inductive	FB20	BEAD50ohm	0.0159	-	-	25.0	B332I6	2
.1.....	1	Inductive	FB39	BEAD600ohm	0.0159	-	-	25.0	B332I6	2
.1.....	1	Inductive	FB9	BEAD120ohm	0.0159	-	-	25.0	B332I6	2
.1.....	1	Connector	INV1	1*5PIN	0.0014	-	-	25.0	B332I6	1
.1.....	1	Connector	J1	1*4PIN	0.0011	-	-	25.0	B332I6	2
.1.....	1	Connector	JBAT1	1*2	0.0006	-	-	25.0	B332I6	1
.1.....	1	Connector	JCP1	1*3PIN	0.0213	-	-	25.0	B332I6	1
.1.....	1	Connector	JCP1-J	1*2PIN	0.0006	-	-	25.0	B332I6	1
.1.....	1	Connector	JLOUT1	AUDIO	0.0026	-	-	25.0	B332I6	1
.1.....	1	Connector	JME-J	1*2PIN	0.0006	-	-	25.0	B332I6	1
.1.....	1	Connector	JSW2	1*2	0.0006	-	-	25.0	B332I6	1
.1.....	1	Connector	JV1-J	1*2PIN	0.0006	-	-	25.0	B332I6	4
.1.....	1	Connector	JVINV1	1*3PIN	0.0213	-	-	25.0	B332I6	5
.1.....	1	Resistor	L1	90ohm	0.0011	P=50.0%	-	25.0	B332I6	10
.1.....	1	Resistor	L4	90ohm	0.0011	P=50.0%	-	25.0	B332I6	4
.1.....	1	IC	LAN1	WGI210IT	0.0198	-	-	25.0	B332I6	2
.1.....	1	Connector	LANCN1	RJ45	0.0005	-	-	25.0	B332I6	2
.1.....	1	Connector	LVDS1	2*15PIN	0.0085	-	-	25.0	B332I6	1
.1.....	1	Connector	MC1	MINI	0.0148	-	-	25.0	B332I6	1
.1.....	1	Connector	MDP1	MiNi-DP	0.0057	-	-	25.0	B332I6	1
.1.....	1	Connector	MEKEY1	NGFF	0.0213	-	-	25.0	B332I6	1
.1.....	1	Connector	MMKEY1	NGFF	0.0213	-	-	25.0	B332I6	1
.1.....	1	Inductive	PL1	10uH	0.0159	-	-	25.0	B332I6	2
.1.....	1	Inductive	PL11	0.12uH-PW	0.0431	-	-	25.0	B332I6	3
.1.....	1	Inductive	PL14	0.24uH-PW	0.0431	-	-	25.0	B332I6	1
.1.....	1	Inductive	PL15	5.6uH-PW	0.0431	-	-	25.0	B332I6	1

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Inductive	PL16	22uH	0.0431	-	-	25.0	B332I6	1
.1.....	1	Inductive	PL17	10uH	0.0159	-	-	25.0	B332I6	1
.1.....	1	Inductive	PL3	1uH	0.0159	-	-	25.0	B332I6	3
.1.....	1	Inductive	PL4	1uH	0.0159	-	-	25.0	B332I6	1
.1.....	1	Inductive	PL6	1uH	0.0159	-	-	25.0	B332I6	4
.1.....	1	Switch	PSW1	6PIN DIP	0.0213	Icont=50.0%	-	25.0	B332I6	1
.1.....	1	Connector	PWRIN1	Terminal	0.0285	-	-	25.0	B332I6	1
.1.....	1	Transistor	Q1	PDC6976X-5	0.0398	P=50.0%	-	25.0	B332I6	4
.1.....	1	Transistor	Q10	2N7002KDW	0.0398	P=50.0%	-	25.0	B332I6	7
.1.....	1	Transistor	Q11	2N7002K	0.0398	P=50.0%	-	25.0	B332I6	7
.1.....	1	Transistor	Q15	MDV1522URH	0.0398	P=50.0%	-	25.0	B332I6	2
.1.....	1	Transistor	Q19	PJT138K	0.0398	P=50.0%	-	25.0	B332I6	3
.1.....	1	Transistor	Q20	BSS138BKW	0.0398	P=50.0%	-	25.0	B332I6	5
.1.....	1	Transistor	Q28	LP3401LT1G	0.0398	P=50.0%	-	25.0	B332I6	2
.1.....	1	Transistor	Q33	PDN3916S	0.0398	P=50.0%	-	25.0	B332I6	2
.1.....	1	Transistor	Q34	2N7002KW	0.0398	P=50.0%	-	25.0	B332I6	2
.1.....	1	IC	Q40	MMBT3906-HF	0.0192	-	-	25.0	B332I6	6
.1.....	1	Transistor	Q47	IRLML6402TRPBF	0.0398	P=50.0%	-	25.0	B332I6	2
.1.....	1	Transistor	Q5	PMBT3904	0.0398	P=50.0%	-	25.0	B332I6	5
.1.....	1	Transistor	Q6	MDU1722VRH	0.0398	P=50.0%	-	25.0	B332I6	4
.1.....	1	Transistor	Q9	MDV3605URH	0.0398	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R1	0.008ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R101	61.9K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R102	330ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R106	499ohm	0.0011	P=50.0%	-	25.0	B332I6	4
.1.....	1	Resistor	R11	60.4K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R112	150K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R115	10ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R117	4.7ohm	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R147	3.83K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R148	34K	0.0011	P=50.0%	-	25.0	B332I6	1

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Resistor	R149	10ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R150	71.5K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R151	34.8K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R152	115K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R153	243ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R157	100ohm	0.0011	P=50.0%	-	25.0	B332I6	7
.1.....	1	Resistor	R158	45.3ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R16	100K	0.0011	P=50.0%	-	25.0	B332I6	10
.1.....	1	Resistor	R160	49.9ohm	0.0011	P=50.0%	-	25.0	B332I6	8
.1.....	1	Resistor	R162	4.87K	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R166	619ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R167	80.6K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R168	1.3K	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R169	806ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R170	4.02K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R171	200ohm	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R172	140ohm	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R173	453ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R174	1.78K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R175	30.9K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R176	78.7K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R177	39.2K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R178	124K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R179	15.4K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R18	1.24ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R180	16.9K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R181	29.4K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R182	35.7K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R183	2M	0.0068	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R184	255K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R185	4.12K	0.0011	P=50.0%	-	25.0	B332I6	2

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Resistor	R187	49.9K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R188	249ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R190	357ohm	0.0016	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R192	15K	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R193	12.1K	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R194	33.2K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R2	10ohm	0.0011	P=50.0%	-	25.0	B332I6	4
.1.....	1	Resistor	R20	49.9K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R204	4.7K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R205	21K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R206	2.2K	0.0011	P=50.0%	-	25.0	B332I6	5
.1.....	1	Resistor	R21	499ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R213	24.9ohm	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R214	1K	0.0011	P=50.0%	-	25.0	B332I6	15
.1.....	1	Resistor	R221	51ohm	0.0011	P=50.0%	-	25.0	B332I6	4
.1.....	1	Resistor	R23	7.5K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R24	390ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R244	121ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R245	80.6ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R247	470ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R248	8.2K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R25	470K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R258	22ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R26	200K	0.0011	P=50.0%	-	25.0	B332I6	10
.1.....	1	Resistor	R261	33ohm	0.0011	P=50.0%	-	25.0	B332I6	9
.1.....	1	Resistor	R266	2.2K	0.0011	P=50.0%	-	25.0	B332I6	4
.1.....	1	Resistor	R273	60.4ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R276	1M	0.0011	P=50.0%	-	25.0	B332I6	6
.1.....	1	Resistor	R28	2K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R287	113ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R295	1K	0.0011	P=50.0%	-	25.0	B332I6	6

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Resistor	R297	10M	0.0068	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R298	2.7K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R305	56.2ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R306	220ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R320	0ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R323	0.01ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R324	240ohm	0.0011	P=50.0%	-	25.0	B332I6	22
.1.....	1	Resistor	R340	2.2ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R372	47K	0.0080	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R374	150ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R378	3.3K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R379	4.99K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R380	33ohm	0.0011	P=50.0%	-	25.0	B332I6	14
.1.....	1	Resistor	R391	330ohm	0.0011	P=50.0%	-	25.0	B332I6	4
.1.....	1	Resistor	R399	33K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R4	1M	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R43	7.5K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R433	4.7K	0.0011	P=50.0%	-	25.0	B332I6	10
.1.....	1	Resistor	R44	390ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R45	499K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R450	5.11K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R451	47K	0.0080	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R46	100K	0.0011	P=50.0%	-	25.0	B332I6	16
.1.....	1	Resistor	R461	75ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R463	22K	0.0080	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R467	4.7K	0.0011	P=50.0%	-	25.0	B332I6	9
.1.....	1	Resistor	R47	20K	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R48	1K	0.0011	P=50.0%	-	25.0	B332I6	19
.1.....	1	Resistor	R485	24.9K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R487	42.2K	0.0011	P=50.0%	-	25.0	B332I6	4
.1.....	1	Resistor	R488	154K	0.0011	P=50.0%	-	25.0	B332I6	2

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Resistor	R49	3.3K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R498	10ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R499	43ohm	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R5	0ohm	0.0011	P=50.0%	-	25.0	B332I6	18
.1.....	1	Resistor	R50	2.7K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R519	20K	0.0011	P=50.0%	-	25.0	B332I6	5
.1.....	1	Resistor	R52	60.4K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R522	2K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R53	7.87K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R54	51K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R55	330K	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	R56	100ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R562	150K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R576	2.2K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R585	560K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R587	2.2K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R590	220K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R6	10K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R60	90.9K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R601	681ohm	0.0016	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R604	510K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R607	121K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R608	2.2ohm	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R610	210K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R612	40.2k	0.0016	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R614	47.5K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R617	63.4K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R621	68.1K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R65	6.49K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R69	133K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R7	24.3K	0.0011	P=50.0%	-	25.0	B332I6	1

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	Resistor	R71	3.57K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R72	2.87K	0.0011	P=50.0%	-	25.0	B332I6	1
.1.....	1	Resistor	R73	3.3ohm	0.0011	P=50.0%	-	25.0	B332I6	6
.1.....	1	Resistor	R74	100K	0.0011	P=50.0%	-	25.0	B332I6	12
.1.....	1	Resistor	R77	5.1ohm	0.0011	P=50.0%	-	25.0	B332I6	6
.1.....	1	Resistor	R78	20K	0.0011	P=50.0%	-	25.0	B332I6	10
.1.....	1	Resistor	R8	1K	0.0011	P=50.0%	-	25.0	B332I6	5
.1.....	1	Resistor	R9	100K	0.0011	P=50.0%	-	25.0	B332I6	2
.1.....	1	Resistor	R91	20K	0.0011	P=50.0%	-	25.0	B332I6	5
.1.....	1	Resistor	R97	100ohm	0.0011	P=50.0%	-	25.0	B332I6	15
.1.....	1	Resistor	RA1	10K	0.0011	P=50.0%	-	25.0	B332I6	50
.1.....	1	Resistor	RB1	10K	0.0011	P=50.0%	-	25.0	B332I6	90
.1.....	1	Resistor	RS1	4.7K	0.0011	P=50.0%	-	25.0	B332I6	3
.1.....	1	Resistor	RY1	0ohm	0.0011	P=50.0%	-	25.0	B332I6	50
.1.....	1	Resistor	RZ1	0ohm	0.0011	P=50.0%	-	25.0	B332I6	165
.1.....	1	IC	SPI1	MX25L512EMI	0.0190	-	-	25.0	B332I6	1
.1.....	1	Switch	SW1	DIP	0.0213	Icont=50.0%	-	25.0	B332I6	1
.1.....	1	OptoDevice	SWOUT1	LED	0.0008	-	-	25.0	B332I6	1
.1.....	1	IC	U1	LT4356HDE	0.0191	-	-	25.0	B332I6	1
.1.....	1	IC	U10	TPS22992RXPR	0.0190	-	-	25.0	B332I6	4
.1.....	1	IC	U12	SN74AUP1G97DCK	0.0189	-	-	25.0	B332I6	2
.1.....	1	IC	U14	SN74LVC1G08DCK	0.0189	-	-	25.0	B332I6	10
.1.....	1	IC	U17	MUN3CAD03	0.0191	-	-	25.0	B332I6	2
.1.....	1	IC	U18	NB685GQ	0.0191	-	-	25.0	B332I6	1
.1.....	1	IC	U2	MIC2127AYML-TR	0.0191	-	-	25.0	B332I6	2
.1.....	1	IC	U23	MP2939GQK-Z	0.0196	-	-	25.0	B332I6	1
.1.....	1	IC	U24	MP86901-CGLT	0.0192	-	-	25.0	B332I6	3
.1.....	1	IC	U27	MP86901-AGQT	0.0191	-	-	25.0	B332I6	1
.1.....	1	IC	U28	SN74AUP1G07DCK	0.0189	-	-	25.0	B332I6	1
.1.....	1	IC	U35	STM811SW16F	0.0189	-	-	25.0	B332I6	2
.1.....	1	IC	U4	MPQ8633AGLE-Z	0.0192	-	-	25.0	B332I6	1

ARBOR® Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023

.1.....	1	IC	U40	SN74CBT3125PWR	0.0191	-	-	25.0	B332I6	2
.1.....	1	IC	U42	ALC888S-VDS-GR	0.0196	-	-	25.0	B332I6	2
.1.....	1	IC	U43	AS78L05RTR-G1	0.0201	-	-	25.0	B332I6	1
.1.....	1	IC	U44	PI3PCIE3212ZBE	0.0192	-	-	25.0	B332I6	1
.1.....	1	IC	U45	SN74LVC1G08DBV	0.0189	-	-	25.0	B332I6	5
.1.....	1	IC	U46	TXS0102DQMR	0.0190	-	-	25.0	B332I6	1
.1.....	1	IC	U48	AP2553AFDC-7	0.0189	-	-	25.0	B332I6	8
.1.....	1	IC	U5	MPQ8633BGLE	0.0192	-	-	25.0	B332I6	1
.1.....	1	IC	U56	F81866D-I	0.0205	-	-	25.0	B332I6	1
.1.....	1	IC	U58	F81439N	0.0195	-	-	25.0	B332I6	2
.1.....	1	IC	U6	MPM3810GQB-Z	0.0191	-	-	25.0	B332I6	1
.1.....	1	IC	U63	ASM1806	0.0201	-	-	25.0	B332I6	1
.1.....	1	IC	U67	SLB9665TT2.0	0.0193	-	-	25.0	B332I6	1
.1.....	1	IC	U68	IDT553S	0.0190	-	-	25.0	B332I6	1
.1.....	1	IC	U7	TPS74801DRC	0.0190	-	-	25.0	B332I6	1
.1.....	1	IC	U70	PTN3460BS	0.0197	-	-	25.0	B332I6	1
.1.....	1	IC	U71	SN74LVC1G126DCK	0.0189	-	-	25.0	B332I6	1
.1.....	1	IC	U72	SN74LVC1G125DCKR	0.0189	-	-	25.0	B332I6	1
.1.....	1	IC	U74	GL852GT-OHY12	0.0193	-	-	25.0	B332I6	1
.1.....	1	IC	U75	MP4560DQ-LF	0.0190	-	-	25.0	B332I6	2
.1.....	1	IC	U9	NB681AGD	0.0191	-	-	25.0	B332I6	5
.1.....	1	Connector	USB1	USB	0.0017	-	-	25.0	B332I6	2
.1.....	1	Connector	USB3	USB	0.0017	-	-	25.0	B332I6	3
.1.....	1	Crystal	X1	32.768KHz	0.0750	-	-	25.0	B332I6	1
.1.....	1	Crystal	X2	24Mhz	0.0750	-	-	25.0	B332I6	1
.1.....	1	Crystal	X3	25MHz	0.0750	-	-	25.0	B332I6	2
.1.....	1	Crystal	X5	48MHz	0.0750	-	-	25.0	B332I6	1
.1.....	1	Crystal	X6	25MHz	0.0750	-	-	25.0	B332I6	1
.1.....	1	Connector	xPCIE1	PCI-E	0.0467	-	-	25.0	B332I6	1
.1.....	1	Crystal	Y3	12MHz	0.0750	-	-	25.0	B332I6	1

End of Report

ARBOR®
Embedded and Network Computing

MTBF Prediction

Date: 07,26,2023