

Implementierungsdokumentation

**Definition und Durchführung von  
Messwertverarbeitung  
für den Physikunterricht  
auf Basis eines Raspberry Pis**

**Version 1.0.0**

David Gawron      Stefan Geretschläger      Leon Huck  
Jan Küblbeck      Linus Ruhnke

10. August 2019

# Inhaltsverzeichnis

<b>1</b>	<b>Ziel der Implementierungsdokumentation</b>	<b>3</b>
<b>2</b>	<b>Ausarbeitungsstand der Abnahmekriterien</b>	<b>4</b>
2.1	Entwurf von Messkonfigurationen . . . . .	4
2.2	Handhabung von Bausteinprototypen . . . . .	5
2.3	Gewährleisten von Persistenz . . . . .	5
2.4	Bereitstellung vorgefertigter Teile . . . . .	6
2.5	Handhabung von Messläufen . . . . .	6
2.6	Benutzbarkeit der GUI . . . . .	6
2.7	Abgrenzungskriterien . . . . .	7
<b>3</b>	<b>Umsetzung des Entwurfs</b>	<b>8</b>
3.1	Model . . . . .	9
3.1.1	Ersetzen des Entwurfsmuster Erbauer durch eine Zuständigkeitskette	9
3.2	View . . . . .	11
3.2.1	GUI-Paket . . . . .	11
3.2.2	Menu-Paket . . . . .	11
3.2.3	Configuration-Paket . . . . .	12
3.2.4	BlockProperties-Paket . . . . .	13
3.2.5	Exception-Paket . . . . .	13
3.2.6	HelpAndOption-Paket . . . . .	13
3.2.7	Model-Interface-Paket . . . . .	13
3.2.8	Controller-Interface-Paket . . . . .	13
3.3	Controller . . . . .	14
3.4	Backend . . . . .	14
3.5	Cache . . . . .	14
3.6	File-Service . . . . .	14
<b>4</b>	<b>Realer Implementierungsablauf</b>	<b>14</b>
<b>5</b>	<b>Anhang</b>	<b>14</b>
<b>6</b>	<b>Glossar</b>	<b>16</b>

## **1 Ziel der Implementierungsdokumentation**

## 2 Ausarbeitungsstand der Abnahmekriterien

Im Folgenden werden die Muss-, Soll- und Wunschkriterien aus dem Pflichtenheft herangezogen, das in der ersten Phase des Projekts entstanden ist. Es findet eine Bestandsaufnahme statt, inwieweit die Kriterien erfüllt sind.

Falls das Softwareprodukt ein Musskriterium nicht wie im Pflichtenheft beschrieben aufweist, so führt dieses Dokument detailliert die Ursachen und Gründe hierfür auf. Falls das Softwareprodukt ein Sollkriterium nicht wie im Pflichtenheft beschrieben aufweist, so beschreibt dieses Dokument zwar nicht in jedem Detail, aber hinreichend informativ die Ursachen und Gründe hierfür. Nicht umgesetzte Wunschkriterien werden lediglich benannt, aber nicht hinterfragt.

### 2.1 Entwurf von Messkonfigurationen

Es fand ein Fallback statt. Die Messkonfigurationen werden nicht wie gewünscht graphisch durch ein Drag- and Drop Feld erstellt, sondern müssen textuell eingegeben werden. Dadurch verändert sich auch die Betrachtung, wie und ob die folgenden Kriterien überhaupt erfüllt werden können.

MK 1 Das Musskriterium „Hinzufügen eines Bausteins aus dem Prototypen-Feld zu der Messkonfiguration“ ist TODO

MK 2 Das Musskriterium „Anpassen von wichtigen funktionalen Bausteineigenschaften“ ist TODO

MK 3 Das Musskriterium „Löschen eines Bausteins aus der Messkonfiguration“ ist erfüllt, da der Benutzer die Textuelle Repräsentation eines Bausteins aus der Messkonfiguration entfernen kann.

MK 4 Das Musskriterium „Erstellen einer Verbindung“ ist umgesetzt. Der Benutzer kann ein Kanaltupel der Liste an Verbindungen hinzufügen und somit eine Verbindung der Messkonfiguration hinzu fügen.

MK 5 Das Musskriterium „Löschen einer Verbindung“ ist erfüllt. Der Benutzer kann eine Verbindung aus der Liste der Verbindungen löschen, in dem er das entsprechende Kanaltupel löscht.

SK 1 Das Sollkriterium „Undo-Redo-Funktion“ ist nicht umgesetzt. Die Messkonfiguration wird textuell erstellt und der Editor unterstützt keine Undo-Redo-Funktion.

WK 1 Das Wunschkriterium „Hinzufügen, Bearbeiten und Löschen von ergänzender Informationen zu der Messkonfiguration durch den Benutzer“ ist nicht umgesetzt.

## 2.2 Handhabung von Bausteinprototypen

SK 2 Der Benutzer ist in der Lage die Eigenschaften der Bausteinprototypen einzusehen. Jedoch ist die Ansicht auf das Anzeigen der *todo* beschränkt. Der Grund hierfür ist die Anbindung von dem Model an die GUI. Dadurch ist es aktuell nur möglich mit den allgemeinen Bausteinprototyp-Informationen zu arbeiten. Dementsprechend ist das Anzeigen der speziellen Eigenschaften, der Bausteinprototypen, nicht möglich.

SK 3 Das Kopieren der Bausteinprototypen ist möglich. Auch das Anpassen der allgemeinen Eigenschaften ist möglich. Jedoch können keine generischen Bausteinprototypen erstellt werden. Dafür wäre die Erstellung einer allgemeinen Vorlage nötig gewesen. Diese Erweiterung hätte dem Nutzer jedoch keine weitere Funktionalität geboten. Aus diesem Grund haben wir uns dazu entschieden die Funktion erst in einer eventuellen Erweiterung der Anwendung zu integrieren.

WK 2 Die Verwendung von erweiternder Software, über eine Schnittstelle, ist nicht mehr vorgesehen. Externe Software kann weiterhin zur Erstellung von Yaml-Dateien genutzt werden. Die so entstandenen Yaml-Dateien können über die Lade-Funktion der Anwendung aufgerufen und anschließend verwendet werden.

SK 4 Die Anwendung unterscheidet in ihrer jetzigen Form nicht zwischen Benutzerdefinierten- und System-Bausteinen. Deshalb ist es auch hier nur möglich die allgemeinen Eigenschaften der Bausteinprototypen zu ändern.

SK 5

## 2.3 Gewährleisten von Persistenz

MK 6 Die verwendeten Bausteine und ihre Anordnung, also die Messkonfiguration, kann in einer Datei gespeichert werden.

MK 7 Messkonfigurationen können aus Dateien geladen werden.

SK 6

SK 7

## 2.4 Bereitstellung vorgefertigter Teile

## 2.5 Handhabung von Messläufen

## 2.6 Benutzbarkeit der GUI

**SK 11** Die Funktionalität Aktionen per Drag-and-Drop durchzuführen ist in der aktuellen Version der Anwendung nicht implementiert. Die Gründe hierfür sind die lange Einarbeitungszeit in ein externes Editor-Programm, welches ermöglicht Objekte über Drag-and-Drop zu bewegen. Dies würde das Auswählen eines Editor-Programms beinhalten, die Einarbeitungszeit, die Implementierung und Integration dieses Programms beinhalten. Durch ein zu spätes Festlegen auf ein Editor-Programm, *JHotDraw* hat die Zeit für die Einarbeitung, Implementierung und Integration gefehlt. Deswegen haben wir uns aus Zeitgründen dagegen entschieden ein Editor-Programm zu implementieren. Leider entfällt dadurch eine grundlegende Funktionalität unserer Anwendung, welche bereits fest vorgesehen war und die Benutzung der Anwendung vereinfachen und verbessern würde. Als Weiterentwicklung hat dieses Kriterium eine hohe Priorität.

**SK 12** Die Anwendung enthält ein Hilfe-Fenster, welche dem Benutzer eine kurze Beschreibung der Funktionalität der Anwendung und Information über die Anwendung bieten. Die Informationen zu den GUI-Elementen lassen sich jedoch nur aus dem Hilfe Fenster-Text auslesen und nicht interaktiv über Bedienung der GUI-Elemente.

**SK 13** Die Anwendung bietet dem Benutzer Information über Fehler-Rückmeldungen über ein Fehlerfenster. Vor Fehlverhalten wird nicht gewarnt sondern nur reaktionär auf Fehler reagiert. Die Implementierung des Vorwarnen vor Fehlverhalten hätte sich durch konstante Analyse der Benutzereingaben als zeitintensiv und komplex erwiesen und wurde deswegen nicht umgesetzt.

**WK 6** Die Anwendung ist in deutscher Sprache. Das Sprach-Paket lässt sich in der jetzigen Version nicht ändern.

**WK 7** Die Festlegung auf ein Konfigurationsfeld, welche eine Konfiguration in schriftlicher Form erstellen lässt führt dazu, dass es keine visuelle Repräsentation von Bausteinen und deren Ein- und Ausgänge gibt.

**WK 8** Die Anwendung legt kein festes Farbschema fest und somit ist das Farbschema nicht anpassbar. Die Implementierung ist vorgesehen, aber war zeitlich nicht umsetzbar.

**WK 9** Die Anwendung legt eine feste Schriftgröße fest, die sich durch die feste Implementierung der GUI-Tools festlegt und ist somit nicht änderbar. Die Implementierung ist vorgesehen, aber nicht umgesetzt.

## **2.7 Abgrenzungskriterien**

### 3 Umsetzung des Entwurfs

Während der Entwurfsphase wurden sowohl UML-Klassendiagramme als auch UML-Sequenzdiagramme erstellt. Zusammen mit der textuellen Beschreibungen der zu erstellenden Software-Elemente bildeten diese die Basis für die Produktion des Quellcodes während der Implementierungsphase.

In aller Regel lassen sich abstrakte Entwurfsinhalte während der Implementierung nicht in allen Details exakt umsetzen, was verschiedene Gründe haben kann. Bisweilen entpuppt sich auch eine andere Umsetzung als vorteilhafter. Die folgenden Abschnitte halten für jedes Softwaremodul die Abweichungen der Implementierung gegenüber den im Entwurf beschriebenen Strukturen fest. Des Weiteren enthalten sie die Gründe für diese Abweichungen.



## 3.1 Model

### 3.1.1 Ersetzen des Entwurfsmuster Erbauer durch eine Zuständigkeitskette

Das Paket „Model.BuildingBlockBuilder“ im Entwurf wurde durch das Paket „model.block“ ersetzt. Das dort verwendete Entwurfsmuster Erbauer erfüllte nicht die notwendige Anforderung, dass der Benutzer leicht eigene Versionen von Bausteinen in die Anwendung einfügen konnte. Darum wurde der Erbauer durch eine Zuständigkeitskette ersetzt. Hier gibt es keine Methode für jeden Baustein im Director, sondern es gibt nur eine Anzahl von Bearbeitern, die einen Block eines Types erstellen. Wenn also der Benutzer eine eigene Transformation erstellen will, kann er die .yaml Datei einer bereits vorhandenen Transformation kopieren und einige Parameter (außer Typ und subtyp) verändern. Die resultierende Transformation wird dann von der Anwendung als eine erkannt und kann dann auch dort verwendet werden. Dadurch entfallen alle folgenden Klassen des Entwurfs:

Builder

TransformationBuilder

RepresentationBuilder

XYRepresentationBuilder

TableRepresentationBuilder

SensorBuilder

VirtualSensorBuilder

PhysicalSensorBuilder

SnakeYamlParser

java.util.hashmap

sowie all diese öffentlichen Methoden in der Director Klasse:

createSensorFromYaml

constructTransformation

`constructXYRepresentation`

`constructNTimeRepresentation`

`constructDS18B20TemperatureSensor`

`constructBMPx80PressureSensor`

`constructINA219CurrentAndVoltageSensor`

`constructMMA8451Accelerometer`

`constructTransformation`

Statt dessen wurden folgende Klassen hinzugefügt:

`GeneralBlockKvProcessor`

`KvProcessor`

`SensorKvProcessor`

`PhysicalSensorKvProcessor`

`VirtualSensorKvProcessor`

`RepresentationKvProcessor`

`TableRepresentationKvProcessor`

`XYRepresentationKvProcessor`

`TransformationKvProcessor`

und die Methode `constructBuildingBlock` zum Director und zu jedem Bearbeitern die Methode `processKvPair` hinzugefügt. Dabei unterscheiden sich die Methoden der einzelnen Bearbeitern zwar nicht im Namen, aber in ihrer Funktion. Jeder Bearbeiter leitet entweder die Anfrage weiter oder erstellt einen Blocktyp und gibt ihn zurück.

## 3.2 View

### 3.2.1 GUI-Paket

**MainWindow** Die zentrale Instanz des GUI-Paketes, die Klasse MainWindow wurde durch Methoden zum Starten der Anwendung erweitert und fungiert somit als Initialisierungs-Modul unserer Anwendung. Dadurch wurde die Klasse um Attribute erweitert, welche für das Starten der Anwendung notwendig sind. Durch die Entscheidung für Swing als Framework-Tool, erbt die Klasse von der Klasse JFrame. Dadurch muss die Klasse durch Methoden zur Erzeugung des Anwendungsfenster erweitert werden. Ebenfalls werden alle anderen GUI-Objekte in dieser Klasse zu dem Anwendungsfenster hinzugefügt.

Erweitert wurde das GUI-Paket um die Klassen DataVisualisation, ConfigurationEditor und Editor.

**ConfigurationEditor** Die Klasse ConfigurationEditor stellt das Textfeld zum Schreiben von Messkonfigurationen dar. Durch die Entscheidung gegen ein Drag-and-Drop Editor stellt diese Klasse nun die Instanz dar, mit welcher der Benutzer eine Messkonfiguration entwirft.

**DataVisualisation** Die Klasse DataVisualisation stellt das Textfeld dar, in welcher Messdaten eines Messlaufs präsentiert werden.

**Editor** Die Klasse Editor benutzt das Framework JHotDraw, um einen Drag-and-Drop-Editor zu erstellen. In der jetzigen Version der Anwendung wird dieser Editor nicht eingebunden und nicht benutzt und liefert ebenfalls nicht die gewünschte Funktionalität. Für eine Weiterentwicklung ist der Editor jedoch ein grundlegender Baustein, um die Anwendung benutzerfreundlicher zu machen.

### 3.2.2 Menu-Paket

Das Menü-Paket wurde durch diverse Veränderungen am Entwurf, wie durch Bereitstellung von Framework- Klassen und Methoden vereinfacht, in dem mehrere Klassen zusammengefasst wurden.

**PrototypeField** In unserer Anwendung umfasst die Klasse PrototypeField ebenfalls die Klassen SensorBlockField, TransformationBlockField und RepresentationBlockField. Die Einzelnen Menüs der Bausteine werden als JTabbedPane, also als TabFenster in dem Übermenü PrototypeField und enthalten die am Anfang initiali-

sierten Bausteine. Ebenfalls lassen sich hier über Drücken des "BearbeitenKnopfes" die Eigenschaften der Bausteine einsehen.

Dadurch entfallen in unserer Implementierung die Klassen:

**SensorBlockField** Enthalten in Klasse PrototypeField und somit entfällt eine separate Implementierung.

**TransformationBlockField** Enthalten in Klasse PrototypeField und somit entfällt eine separate Implementierung.

**RepresentationBlockField** Enthalten in Klasse PrototypeField und somit entfällt eine separate Implementierung.

**FieldHandler** Eine separate Implementierung von Handler-Klassen entfällt durch das Swing-Framework durch die Benutzung von vordefinierten Listener-Klassen.

Die Funktionalität der Buttons wurde in der Implementierung beibehalten, jedoch die Klassen zusammengefasst, da das Framework hierfür bereits Funktionalität vorgibt.

**ButtonField** Die Klasse ButtonField umfasst zusammen mit der Klasse MeasurementButtonField in unserer Implementierung Teile des Button-Paket und die Funktionalität. Die Klassen Button und deren Unterklassen sind somit als JButton-Attribut in dem ButtonField enthalten.

**MeasurementButtonField** Die Klasse MeasurementButtonField enthält die anderen Teile des Button-Pakets, welche den Messlauf beeinflussen, d.h Pause, Resume, Reset und SaveMeasurementData. Ebenfalls sind diese Knöpfe als Attribut der MeasurementButtonKlasse wieder zu finden.

Dadurch entfällt das komplette Button-Paket.

### 3.2.3 Configuration-Paket

Durch die Entscheidung Konfigurationen schriftlich aufzubauen und keinen Drag-and-Drop-Editor zu benutzen entfallen die visuellen Representationen der Bausteine, der Verbindung, der Ein- und Ausgänge, sowie die zugehörigen Handler. Damit erfüllt das Configuration-Paket in der Implementierungsversion keine Funktionalität.

### **3.2.4 BlockProperties-Paket**

### **3.2.5 Exception-Paket**

Das Exception-Paket wurde fast vollständig aus dem Entwurf übernommen. Einzelne Methoden entfallen durch das Framework, wie z.B. `closeAll()` in Klasse `ExceptionWindowManager` oder `close()` in `ExceptionWindow`. Ebenfalls wurde in der Klasse `ConnectionExceptionWindow` die Methode `changeWireColor()` entfernt, da durch unsere Entscheidung das Projekt von visuellen Komponenten zu kapseln, die Verbindung nicht mehr zu einer Instanz der Klasse `Wire` zugeordnet werden kann.

### **3.2.6 HelpAndOption-Paket**

Das HelpAndOption-Paket wurde insofern verändert, dass die Interfaces `HelpWindowHandler` und `OptionsWindowHandler` aufgrund der Framework-Entscheidung entfallen. Die Klassen `HelpWindow` und `OptionsWindow` wurden durch Attribute und Methoden zur Anpassen an das Framework erweitert, damit die Fenster als unabhängige Instanzen angezeigt werden. Das `OptionWindow` wurde durch weitere Optionen erweitert, d.h. System-Optionen, Fenster-Optionen, Messlauf-Optionen und Raspberry-Pi-Optionen.

### **3.2.7 Model-Interface-Paket**

Das Interface `ViewDirectoryInterface` wurde nach Entwurfsvorgaben implementiert. Einzelne Methoden finden in der jetzigen Version der Anwendung keine Anwendung, sind aber für eine mögliche Weiterentwicklung notwendig, um bestimmte Funktionalitäten zu bieten.

### **3.2.8 Controller-Interface-Paket**

Das Controller-Interface-Paket wurde nach Entwurfsvorgaben implementiert. Einzelne Methoden aller Interfaces finden in der jetzigen Version der Anwendung keine Anwendung, sind aber für eine mögliche Weiterentwicklung notwendig, um bestimmte Funktionalitäten zu bieten. Bei einer Weiterentwicklung um ein Konfiguration-Editor, welcher Drag-And-Drop unterstützt, werden die Interfaces `IBlockAction` und `IConnectionAction` eine wesentliche Rolle zur Verbindung der zwei Module spielen.

### 3.3 Controller

### 3.4 Backend

### 3.5 Cache

### 3.6 File-Service

**CsvService** Die Klasse CsvService wurde in der Implementierung durch die Methoden zum Umbenennen und Exportieren von CsvDateien erweitert. Die im Entwurf gewollte Funktionalität wird durch die Implementierung umgesetzt.

**PngService** Die Funktionalität der Klasse PngService wurde wie im Entwurf beschrieben umgesetzt. Sie wird jedoch durch die Umorganisation der Projektfunktionalität und Projektstruktur nicht verwendet.

**YamlService** Die Klasse YamlService wurde nach der Entwurfsstruktur umgesetzt. Durch Anpassung an SnakeYaml wurden jedoch einige Parameter und Argumente verändert.

## 4 Realer Implementierungsablauf

Dieser Abschnitt führt auf, inwieweit der tatsächliche zeitliche Implementierungsablauf vom geplanten Ablauf abgewichen ist, und beschreibt die Ursachen und Gründe für diese Abweichungen. Abhängigkeiten zwischen den Implementierungsschritten und kritische Pfade stehen hierbei besonders im Fokus.

Von Abweichungen betroffene Softwareelemente werden nicht im Einzelnen aufgeführt, sondern es werden lediglich in Bezug auf die Abweichungsgründe die Gruppen der betroffenen Softwareelemente benannt.

## 5 Anhang

[illegible]

## 6 Glossar

**JHotDraw** JHotDraw ist ein Open-Source, Java-basiertes Framework zur Erstellung von grafischen Editoren. Durch die einfachere Unterstützung von Drag-and-Dop, als komplexere Frameworks eine gute Alternative..

**todo** todo.