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

18. August 2019

Inhaltsverzeichnis

1 Ziel der Implementierungsdokumentation

Im sogenannten Wasserfallmodell folgte nach der Anforderungsanalyse und dem Entwurf nun die Implementierung der Softwarewaremodule als dritte Entwicklungsphase. Dieses Dokument soll den Entwicklungsstand am Ende dieser Phase sowie den Verlauf der Phase wiedergeben. Hierbei werden das erstellten Pflichtenheft und die zuvor erstellten UML-Diagrammen herangezogen, und der Stand der Entwicklung wird zu den Inhalten dieser Artefakte in Bezug gesetzt.

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 nicht erfüllt. Durch den Fallback zu einem textuellen Messkonfigurationsfeld ist ein direktes Platzieren nicht mehr möglich. Der Benutzer kann zwar anhand der Daten des Bausteinprototypen einen solchen im Konfigurationsfeld erstellen, aber ein direktes Platzieren einer Kopie ist nicht möglich. Die Priorität in der zweiten Hälfte der Implementierungsphase lag darin, eine funktionierende Minimalanwendung zu erstellen, so dass überhaupt ein Prototyp zur Veränderung und Speicherung eines Datenstroms erstellt wird. Die Entscheidung auf den Fallback zurück zu greifen, fiel in der zweiten Hälfte der Implementierungsphase. In der ersten Hälfte wurde viel Zeit damit verbracht ein geeignetes Gui-Framework für eine Drag- and Drop Anwendung zu suchen und sich darin einzuarbeiten. Die Einarbeitungszeit war dabei sehr Zeit fordernd, so dass sich das komplette Projekt verzögerte und drohte zu scheitern.

MK 2 Das Musskriterium „Anpassen von wichtigen funktionalen Bausteineigenschaften“ ist teilweise erfüllt. Der Benutzer kann die Eigenschaften des Bausteins verändern, allerdings ist das Verändern der Position im Konfigurationsfeld nicht wirklich möglich. Der Benutzer kann zwar die Reihenfolge der Bausteine innerhalb der textuellen Konfiguration verändern, allerdings haben die Bausteine im Model keine Positionsattribute und es gibt auch kein Konfigurationsfeld mit Positionskoordinaten. Darum ist dieser Teil des Musskriteriums nicht erfüllt.

MK 3 Das Musskriterium „Löschen eines Bausteins aus der Messkonfiguration“ ist teilweise erfüllt, da der Benutzer die Textuelle Repräsentation eines Bausteins aus der Messkonfiguration entfernen kann und dieser, nach einem Neuladen, auch so im Model entfernt wird. Allerdings war ursprünglich eine benutzerfreundliche und intuitive Ansteuerung der Bausteine über das graphische Konfigurationsfeld in der Gui angedacht. Da dieses durch den Fallback entfällt, wurde hier eine textuelle Lösung implementiert.

MK 4 Das Musskriterium „Erstellen einer Verbindung“ ist teilweise umgesetzt. Der Benutzer kann ein Kanaltupel der Liste an Verbindungen hinzufügen und somit eine Verbindung der Messkonfiguration hinzu fügen. Da dies aber ursprünglich über das graphische Konfigurationsfeld geschehen sollte, ist hier nur der textuelle Fallback implementiert.

MK 5 Das Musskriterium „Löschen einer Verbindung“ ist teilweise erfüllt. Der Benutzer kann eine Verbindung aus der Liste der Verbindungen löschen, indem er das entsprechende Kanaltupel löscht. Dies entspricht aber auch nicht der ursprünglich gedachten Umsetzung über das graphische Konfigurationsfeld. Darum ist dieses Musskriterum auch nur teilsweise erfüllt.

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 *allgemeinen Bausteinprototyp-Informationen* 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 Vorläge 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 Das Löschen von erstellten Bausteinprototypen ist ermöglicht.

2.3 Gewährleisten von Persistenz

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

MK 7 Das Kriterium ist erfüllt. Messkonfigurationen können aus Dateien geladen werden.

SK 6 Das Kriterium ist nicht erfüllt, da keine neuen Prototypen in der Anwendung erstellt werden können. (Siehe SK 3)

SK 7 Bausteine werden aus Dateien geladen, die jedoch nur außerhalb der Anwendung erstellt werden können.

2.4 Bereitstellung vorgefertigter Teile

MK 8 Die Bausteinprototypen für Sensoren werden direkt aus den Yaml Dateien, die sich in PhyPiDAQ befinden, erstellt.

MK 9 Generische Transformationen können erstellt werden. Die Funktion, die sie realisieren, kann in ihren Eigenschaften angepasst werden.

MK 10 Darstellungsbausteine für einen Messablauf sind nicht vorhanden. Nach einer Neu-Auslegung des Projekts sind nur noch textbasierte Interaktionsmöglichkeiten, für den Benutzer, vorhanden. Der Grund hierfür waren drei Technologiewechsel in der GUI.

1. Der erste Wechsel erfolgte von *Standard Widget Toolkit* (SWT) auf *Graphical Editing Framework* (GEF). SWT bot uns keine Möglichkeit unkompliziert

eine Drag-And-Drop Funktionalität zu integrieren. Wir hielten es daher für sinnvoll das, auf SWT basierende, GEF als Framework zu verwenden. Die Anwendung war als Eclipse-Erweiterung konzipiert. Wir stellten jedoch fest, dass die Verwendung von GEF einen zu hohen Zeitaufwand für die Einarbeitung zur Folge hätte.

2. Der zweite Wechsel erfolgte von GEF auf *JHotDraw*. Dieser Wechsel sollte die Komplexität der Technologie reduzieren. Eine Möglichkeit Drag-And-Drop zu verwenden war weiterhin geplant.

3. Der dritte Wechsel erfolgte von JHotDraw auf *Swing* und *Abstract Window Toolkit* (AWT). Dieser Wechsel ermöglichte eine Umsetzung einer textbasierten Darstellung von der Daten. Dementsprechend mussten die Möglichkeit der GUI beschränkt werden. Hätte diese Umstellung nicht stattgefunden wäre ein hohes Risiko bestanden, dass keine lauffähige Anwendung erstellt werden könnte.

SK 8 Es liegen drei unterschiedliche Messkonfigurationen vor.

WK 3 Es sind keine Spiele, oder vergleichbares, vorhanden.

2.5 Handhabung von Messläufen

Die Messläufe beziehen sich auf die aktuell geladene Messkonfiguration und können nicht nur Messdaten vom Raspberry Pi beziehen, sondern auch in einem Demomodus simulierte Messdaten verwenden. Zum Test-Modus als auch zur zeitlichen Parametrisierung der Messläufe gibt es jeweils eine Registerkarte in den Einstellungen.

MK 11 Der Start eines Messlaufs erfolgt über den letzten der drei Hauptbuttons am oberen Fensterrand. Ist ein Messlauf einmal gestartet, kann der Benutzer in einem gesonderten Feld weitere Buttons zur Steuerung der Messläufe benutzen.

Das Starten, Pausieren, Fortführen und Beenden funktioniert. Auf einen Beenden-Button wurde bewusst verzichtet, da der Pause-Button für den Benutzer den gleichen Effekt hat, und er so stets die Möglichkeit hat, den Messlauf doch noch einmal fortzuführen. Für den Fall, dass der Messlauf atypisches Verhalten aufzeigen sollte, oder falls Einstellungen geändert wurden, gibt es einen im Pflichtenheft nicht beschriebenen Reset-Button.

Dieses Kriterium kann als erfüllt angesehen werden.

MK12 Die Abtastrate der Sensoren kann der Benutzer zum Ende der Implementierungsphase lediglich (zusammen mit anderen Sensor-Metadaten) einsehen. Die Konfiguration der Python-Skripte gestaltete sich schwieriger als erwartet, so dass eine Vereinbarung getroffen wurde, dass auch der Einsatz von Pythonskripten mit fest eincodierter Abtastrate in Ordnung geht. Die Zeitpunkte, zu denen die (nachgelagerten) Bausteine in der Messkonfiguration eine Verarbeitung der Messdaten vornehmen, können aufgrund der Latenzen, die bei der Netzwerkübertragung auftreten können, als zeitlich entkoppelt betrachtet werden. Über die Einstellungen kann der Benutzer ein Zeitraster definieren. Messdaten, die im selben Zeitschlitz abgetastet wurden, werden von FreeJDaq zusammengeführt und einer Messkonfiguration als eine einzige Belegung der Sensor-Kanäle übergeben.

SK9 Die im Kriterium beschriebenen Checks finden erst beim Start eines Messlaufs statt. Backend und Model geben dann jedoch einigermaßen aussagekräftige Fehlermeldungen ab.

SK10 Die verarbeiteten Messdaten können in einer Csv-Datei gespeichert werden. In den Sensorblöcken ist jedoch keine Speicherfunktion implementiert. Sie können dort lediglich geloggt werden.

WK4 Die Sensorbausteine können nicht direkt Testdaten aufnehmen. Simulierte Messdaten können jedoch vom Backend eingespielt werden.

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-Programm beinhalten, die Einarbeitungszeit, die Implemtierung und Integration dieses Programm beinhalten. Durch ein zu spätes Festlegen auf ein Editor-Programm, JHotDraw hat die Zeit für die Einarbeitung, Implementierung und Integrierung 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. Aus diesen Gründen ist dieses Sollkritierium nicht erfüllt.

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. Daher ist dieses Sollkriterium teilweise erfüllt.

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. Daher ist dieses Sollkriterium nur teilweise 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

Alle Abgrenzungskriterien wurden eingehalten.

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 Model Initialisierung

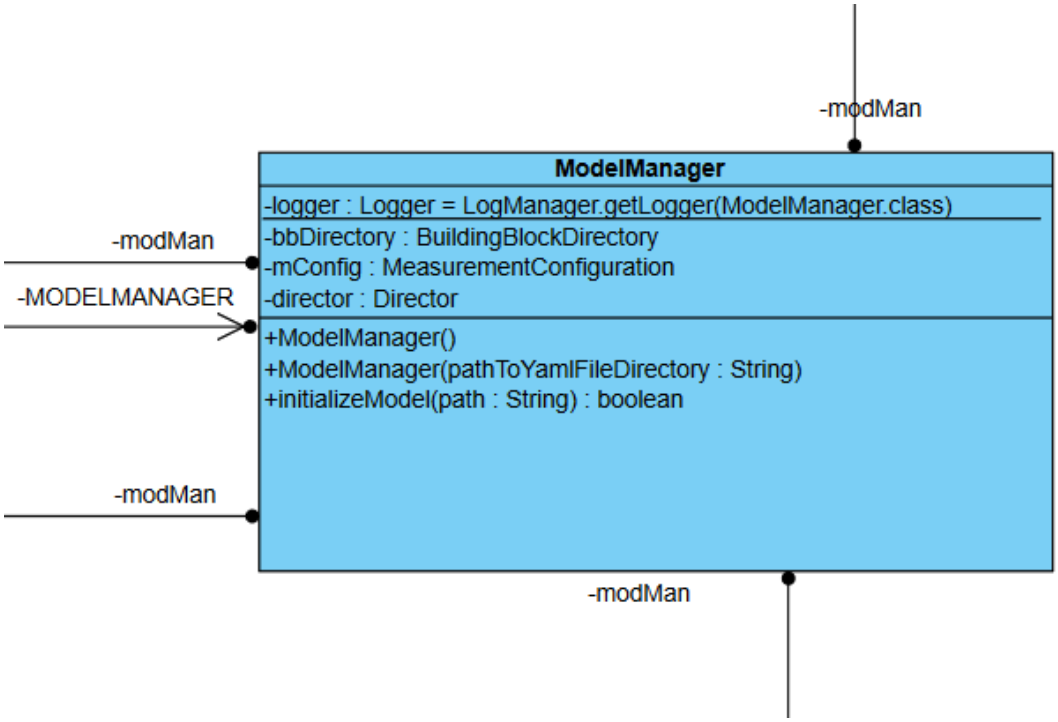

Abbildung 1: Die Klasse ModelManager

Der Modelmanager initialisiert das Model anhand eines Pfades zu den Bausteinprototypdateien, vergleiche Abbildung 1. Die privaten Methoden des Entwurfs createNewMeasurementConfiguration, createTransformationPrototypesFromYaml, createRepresentationPrototypesFromYaml, fetchSensorPrototypesFromBackedn und pushNetworkConfigToBackend wurden an anderer Stelle oder gar nicht implementiert, da ihre Funktionalität an dieser Stelle oder überhaupt nicht gebraucht wurden.

Wichtig für die Initialisierung ist auch die BuildingBlockDirectory Klasse (Abbildung 2), der alle erstellten Prototypen sowie die Bausteine der Messkonfiguration speichert. Außerdem implementiert diese Klasse auch eine Schnittstelle für die Gui. Die Methoden addConfigConnection und removeConfigConnection wurden nicht implementiert, da diese Funktion über den Controller realisiert wurde. Der BuildingblockDirectory dient für die Gui nur als Nachschlageort, um Daten und Objekte zu erhalten. Die Verwaltung der Messkonfiguration geschieht über den Controller und die Messkonfigurationsklasse. Hinzugefügt wurde die Methode subscribeUpdateEvents und deleteConfig.

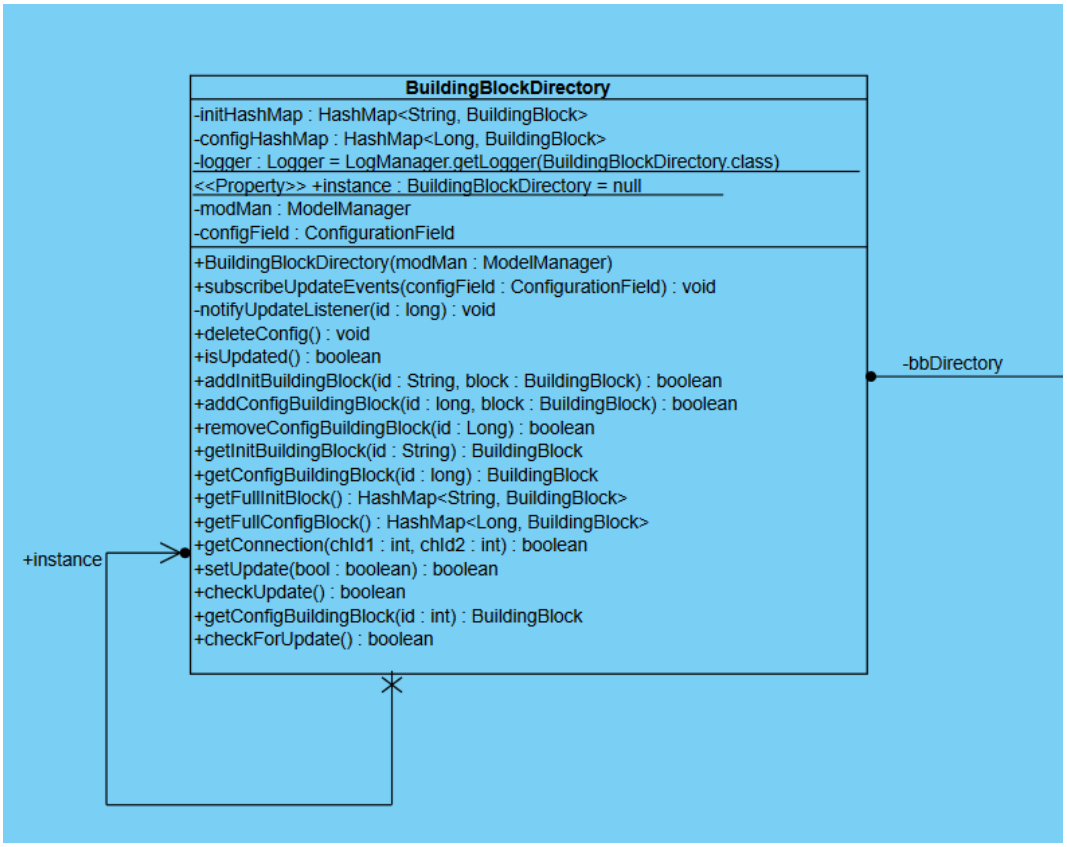

Abbildung 2: Die Klasse ModelManager

3.1.2 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 Benutzter leicht eigene Versionen von Bausteinen in die Anwendung einfügen konnte. Darum wurde der Erbauer durch eine Zuständigkeitskette ersetzt, siehe Abbildung 3 . 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 Benutzter 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

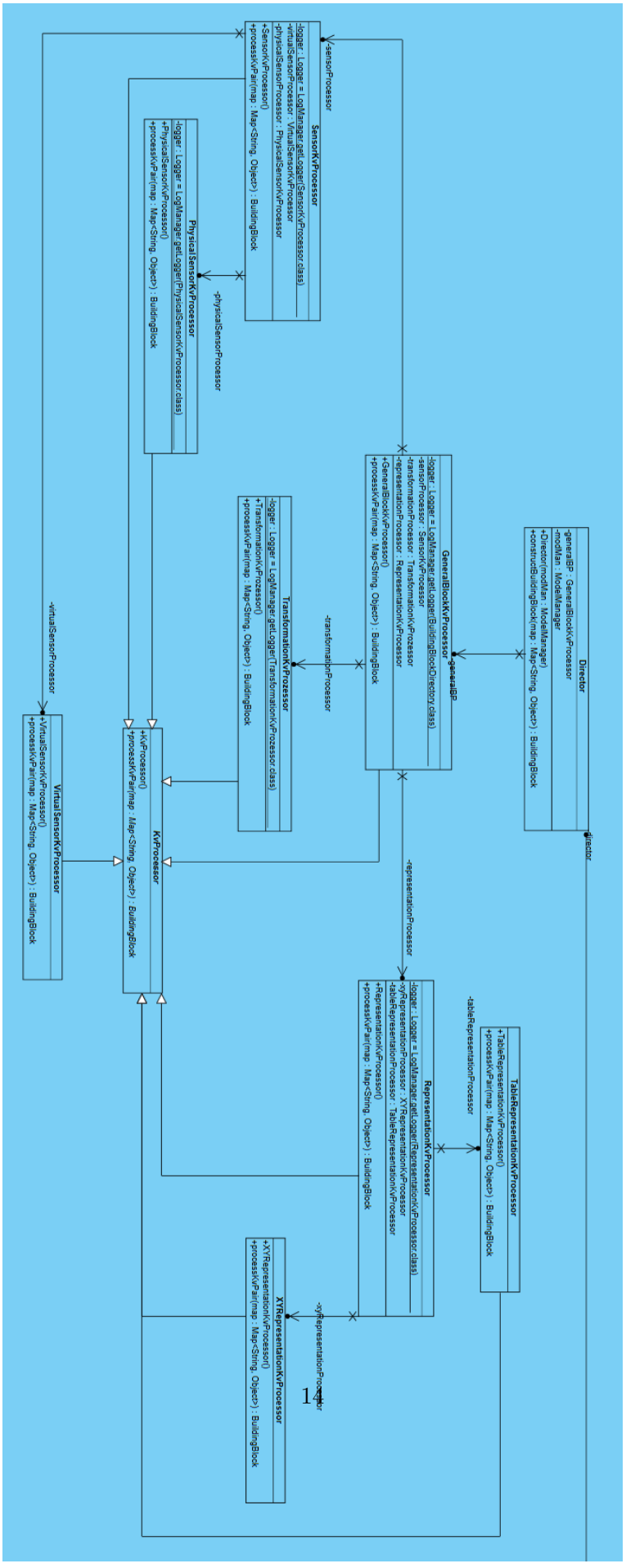

Abbildung 3: Aufbau der Zuständigkeitskette

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.

3.1.3 Auswirkungen des Fallbacks auf die Bausteinklassen

Nach der Hälfte der Implementierungszeit haben wir uns entschlossen die graphische Drag- and Drop-Funktion aufzugeben, um einen funktionierenden Prototypen entwickeln zu können, der einen Datenfluss zwischen allen Modulen der Anwendung realisiert. Wir haben die graphische Benutzeroberfläche stark vereinfacht und die Bearbeitung der Messkonfiguration erfolgt nun textuell. Dadurch sind viele Klassen und Methoden im

Model nicht mehr notwendig oder wurden für die neuen Funktionen angepasst. Die Benutzbarkeit und Benutzerfreundlichkeit wird durch den Fallback zwar reduziert, aber ein funktionierender Prototyp hatte mehr Priorität.

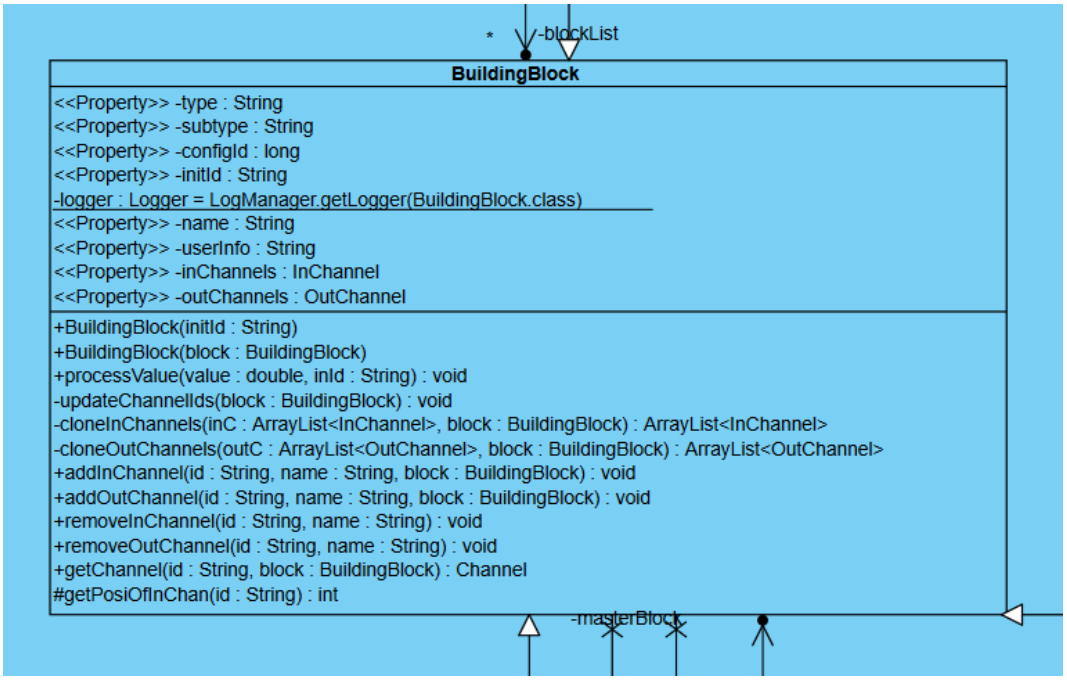

Abbildung 4: Die Klasse BuildingBlock

Die Klasse BuidingBlock aus dem Paket model.core ist nun nicht mehr abstrakt und ist in Abbildung 4 zu sehen. Sie hat außerdem die Methode processValue erhalten um die Benutzung unspezifizierter Bausteine zu erkennen. Außerdem sind die Klassen helpMessage und yamlRepresentation, die eine Assoziation zu der Klasse BuildingBlock haben nicht implementiert. Die yaml Repräsentation wird durch den Fileservice realisiert.

Die Entwurfspakete model.sensorLogic , model.transformationLogic und model.representationLogic wurden zu den Implementierung-Paketen model.sensor (Abbildung 5), model.transformation (Abbildung 6) und model.representation (Abbildung 7) umbenannt. In den drei Paketen wurden alle Entwurfsklassen beibehalten. Allerdings haben die Klassen VirtualSensor, PhysicalSensor, TableRepresentation und XYRepresentation keine Funktion, außer um möglichen zukünftigen Bausteinen eine Grundlage zur erweiterten Kategorisierung zu bieten. Diese vier Klassen haben ihre explizite Methode processValue verloren, da sich diese von Implementierung der jeweiligen Oberklasse nicht unterscheidet. Außerdem sind die drei Klassen Sensor, Transformation und Representation in der Implementierung nicht mehr abstrakt. Alle Bausteinklassen haben außerdem einen Copy-Constructor erhalten um tiefe Kopien anlegen zu können.

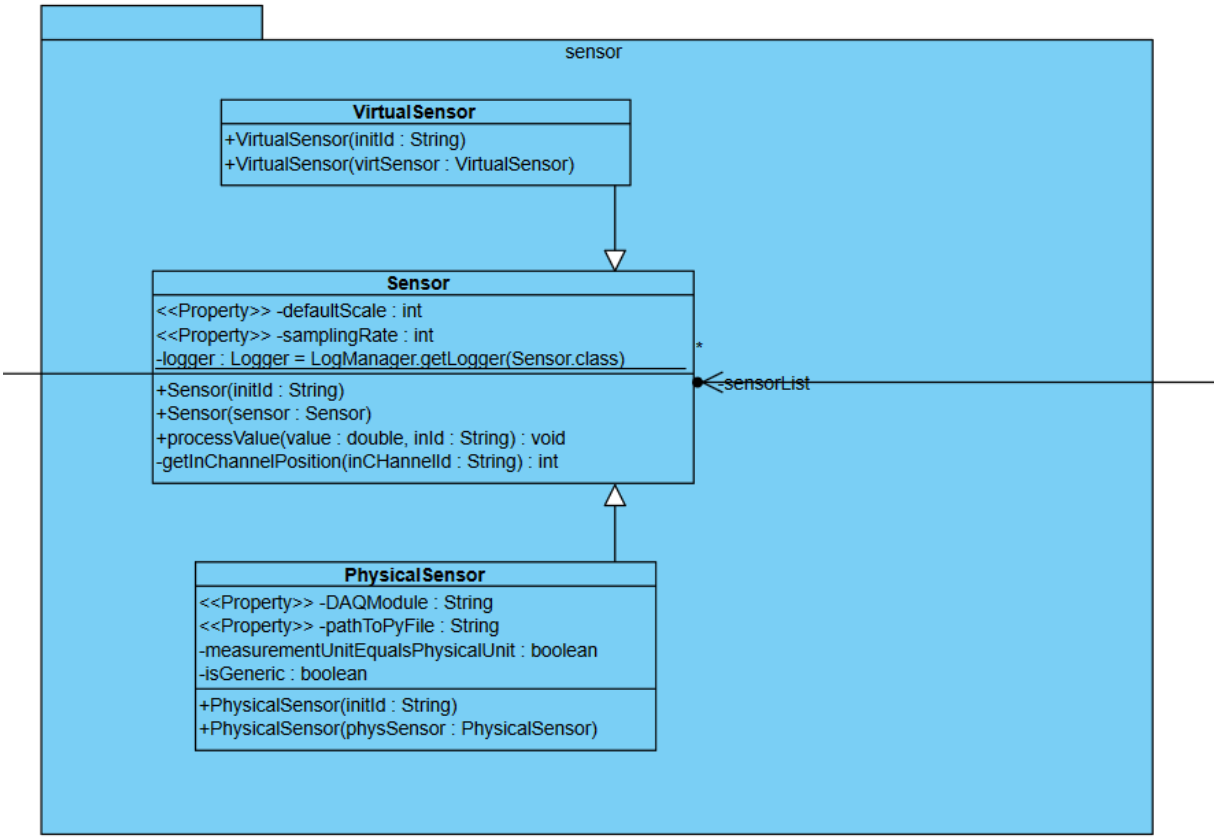

Abbildung 5: Aufbau des Sensor-Paket

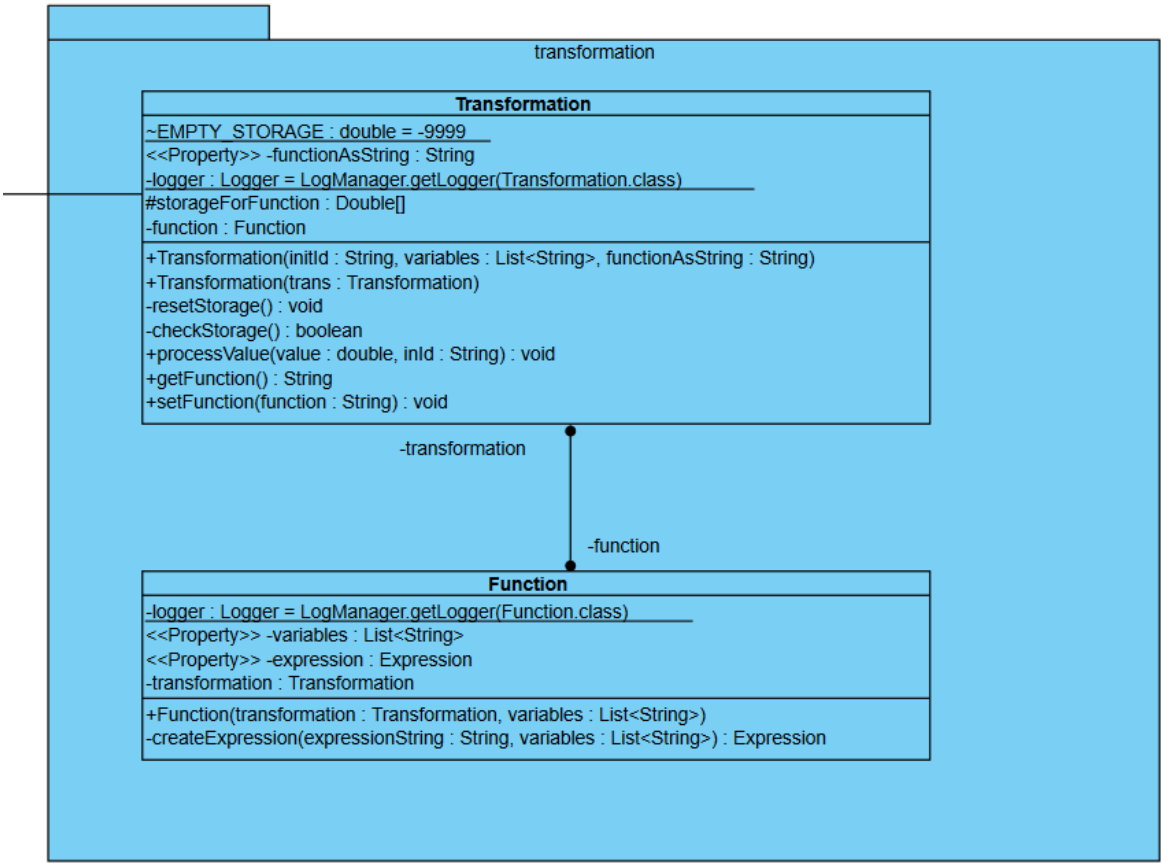

Abbildung 6: Aufbau des Transformation-Paket

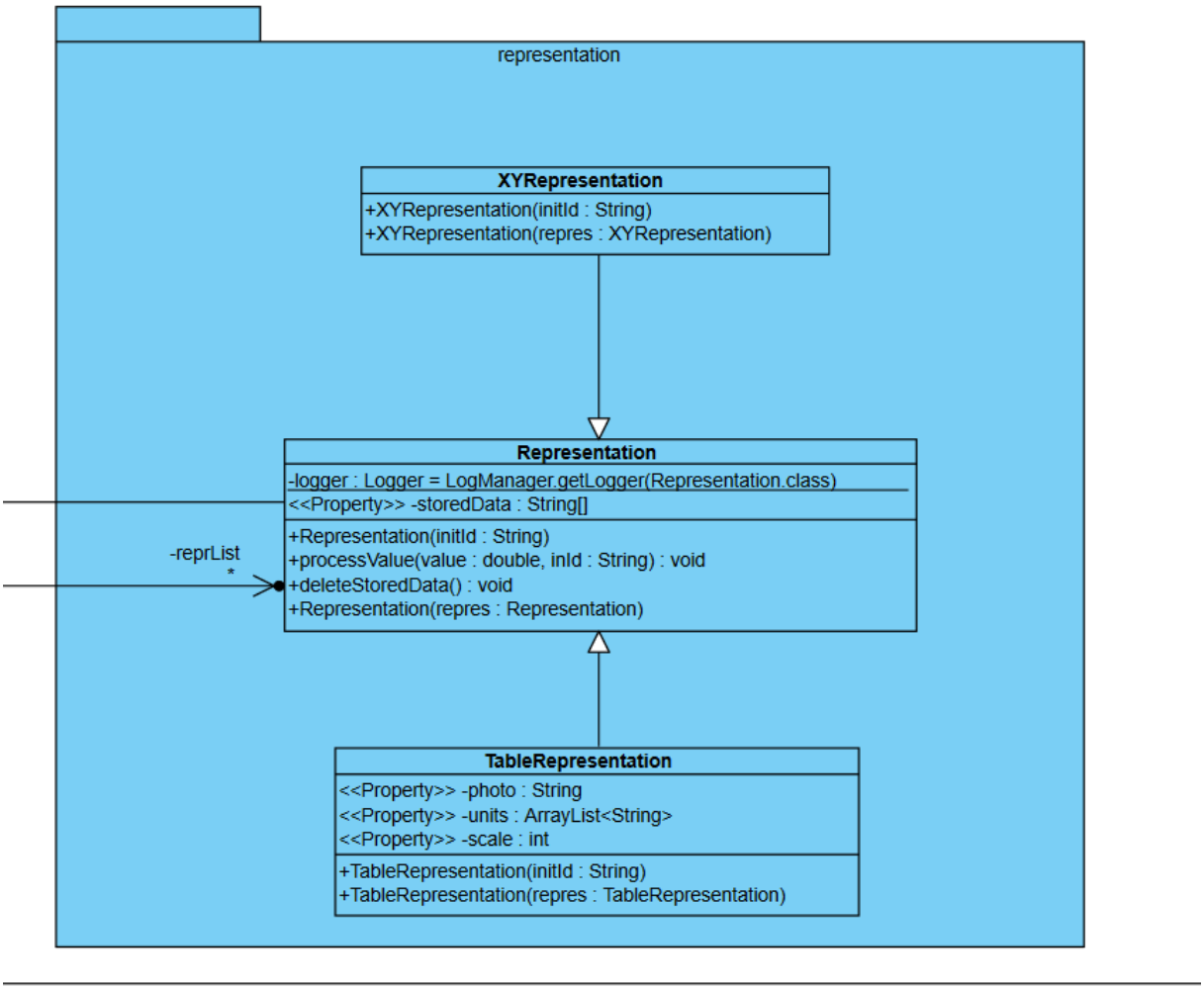

Abbildung 7: Aufbau des Repräsentation-Paket

Die Klasse Function aus dem Paket model.transformation hat ihre Methode applyFunction verloren und erstellt nun nur noch die benötigte Funktion aus den Daten der .yaml Datei. Die Funktionen der Methode applyFunction wurde in die Methode processValue der Klasse Transformation eingefügt, so dass der reinkommende Wert schneller verarbeitet und weiter gesendet werden kann.

Die Methode processValue aller Bausteine verarbeitet nun keine Pakete mehr sondern nur noch deren Werte.

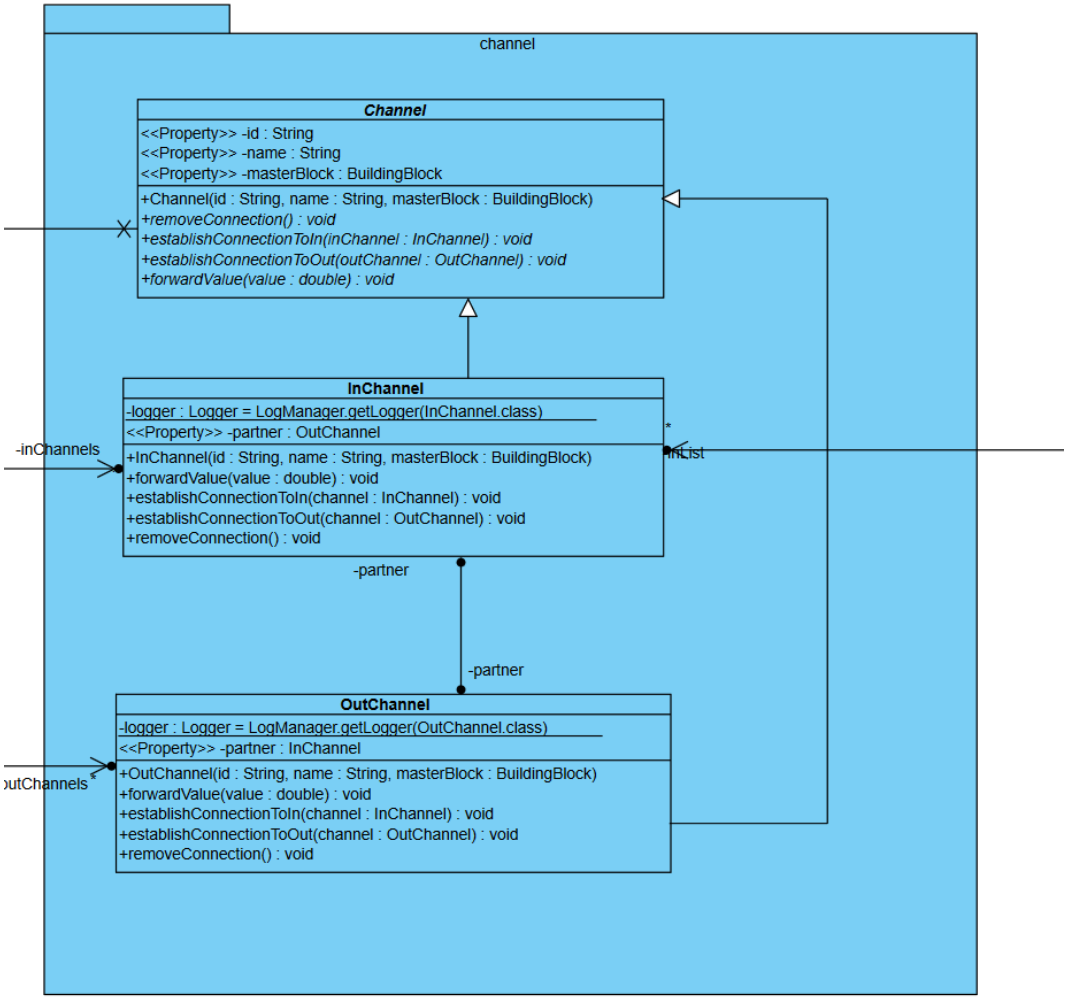

Abbildung 8: Aufbau des Channel-Paket

Das Entwurfspaket model.channellogic hat nun in der Implementierung den Namen model.channel (Abbildung 8). Die Klassen Channelstate, Connected, UnConnected und

Value Ready des Entwurfs haben keine Funktion mehr und werden daher verworfen. Dadurch sind auch die entprechenden Methoden connect, unconnect, receiveValue, eraseValue in der Klasse Channel verworfen worden. Außerdem wurde die Klasse Channel abstract gemacht. Die Methode establishConnection wurde je einmal für jede Kanalart in den Unterklassen implementiert. Dadurch kann der Versuch, eine Verbindung zwischen zwei gleichen Kanaltypen zu erstellen, verhindert werden.

3.1.4 Model Kommunikation mit anderen Modulen

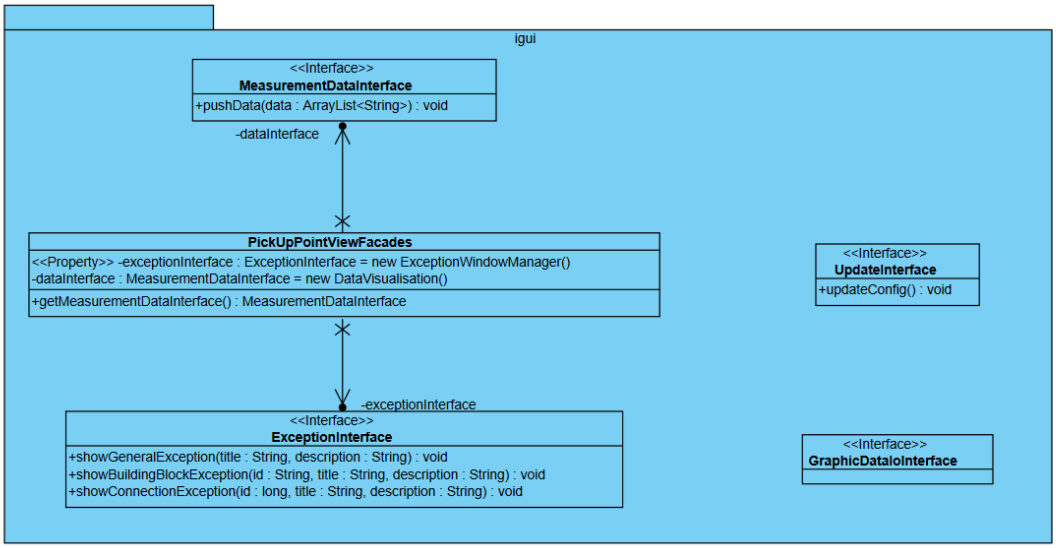

Abbildung 9: Aufbau des Pakets igui

Unsere Annahmen im Entwurf, welches Modul was von wem braucht, waren nicht alle korrekt. Darum wurden Schnittstellen und Implementierungen von Schnittstellen um nötige Methoden erweitert und um unnötige Methoden reduziert. Das Entwurfspaket model.facadeControllerView heißt nun model.igui, vergleiche dazu Abbildung 9. Die Klassen sind alle erhalten. Die Entwurfsmethode pushShellForVisualization wurde durch die Methode pushData ersetzt, da nun nur Daten und keine Graphen an die GUI gepusht werden.

Die Klasse MRunInfo (Abbildung 10) aus dem Modul model wurde um einige Methoden erweitert um die nötigen Informationen im richtigen Format zum Backend und zum Cache zu liefern. Dazu gehören die Methoden: consumeChannelIdsOfSensorAtCache, consumeChannelIdsOfSensorAtBackend, addSensorIdAndItsChannelIds und getNumberOfChannelIds.

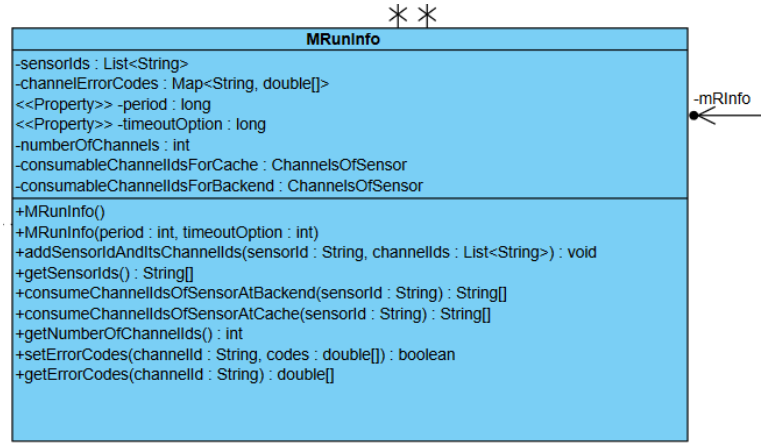

Abbildung 10: Die Klasse MRunInfo

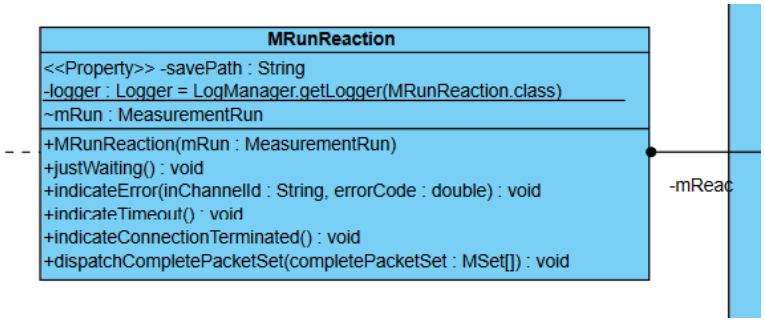

Abbildung 11: Die Klasse MRunReaction

Die Klasse MRunReaction (Abbildung 11) implementiert zwar alle Interface-Methoden des Entwurfs, allerdings wird bisher nur die Methode dispatchCompletePacketSet im Model benutzt um eine Welle von Daten im Model verarbeiten zu lassen. Die übrigen Methoden werden erst in der Qualitätssicherungsphase für die Bereitstellung eines robusteren Messlaufs benötigt.

3.1.5 Model Messlauf

Schon im Entwurf war es eine große Herausforderung einen Messlauf und eine dazu gehörige Messkonfiguration zu erstellen. Die Komplexität der Messkonfigurationsklasse hat sich in der Implementierung erhöht, da einige Funktionalitäten hinzukamen, die im Entwurf unscheinbar wirkten. Durch den Fallback zu einer textuellen Messkonfiguration wurden an diese Klasse zusammen mit der Messlauf Klasse noch mehr Veränderungen möglich um einen funktionierenden Prototypen zu erhalten.

In Der Klasse MeasurementRun (Abbildung 12) ist die private Entwurfsmethode checkSensorInChannelStates nicht implementiert, da ein Kanal überhaupt keinen Status mehr hat. Die anderen Entwurfsmethoden wurden alle implementiert. Zusätzlich wurde diese Klasse noch um die Methode pushData und saveData erweitert. Diese Methoden dienen dazu, Messdaten zu speichern oder zur Gui zu pushen. Schließlich gibt es noch die Methode getInChanMRun die für die Verwaltung von Ids und Kanälen benötigt wird und auch von anderen Model Klassen verwendet wird.

Die komplexe Klasse MeasurementConfiguration (Abbildung 13) implementiert alle Entwurfsklassen außer checkForCycle und createModelFromYamlDom. Die erste Methode wird erst für die Qualitätssicherung benötigt. Die Funktionalität der zweiten Methode wurde durch die Metode loadConfig ersetzt. Zusätzlich wurde die Methode saveConfig implementiert, um auch eine Messkonfiguration speichern zu können. Die neue Methode getInitId, getInChan, OutChannel, getBlockListPosition dienen zur Verwaltung der Messkonfiguration und den Verbindungen zwischen Blöcken.

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 Initialsierungs-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.

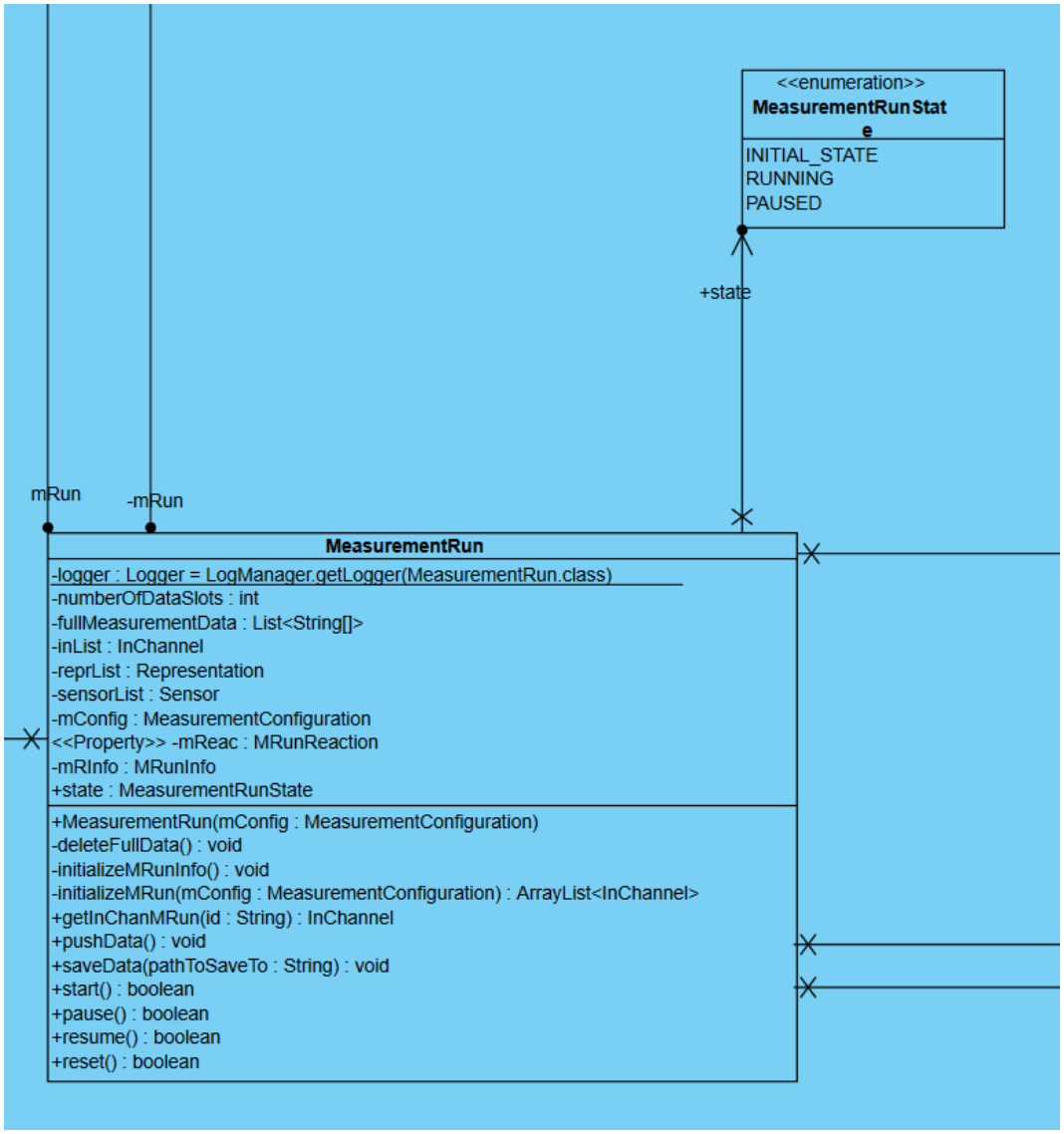

Abbildung 12: Die Klasse MeasurementRun

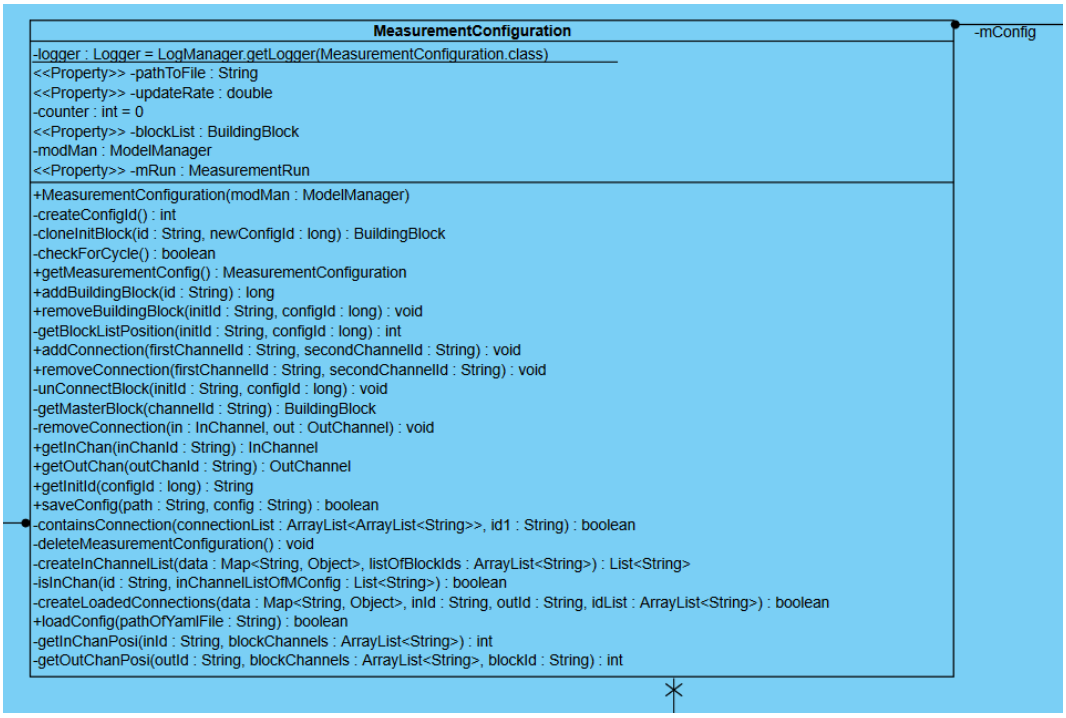

Abbildung 13: Die Klasse MeasurementConfiguration

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 Tab-Fenster in dem Übermenü PrototypeField und enthalten die am Anfang initialisierten 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 seperate 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 Measurement-ButtonField 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 Repräsentationen der Bausteine, der Verbindung, der Ein- und Ausgänge, sowie die zugehörigen Handler. Damit erfüllt das Configuration-Paket in der Implementierungsversion keine Funktionalität. Für eine Weiterentwickling der Anwendung mit einem Editor, welcher Drag-and-Drop unterstützt ist das Paket wieder eine zentrale Instanz der Anwendung.

3.2.4 BlockProperties-Paket

Durch die Verwendung des Swing-Framework entfallen ebenfalls in diesem Paket die Handler-Interfaces, da das Framework diese Funktionalität in Klassen und Methoden liefert. Daher entfallen in der Implementierung die Interfaces BuildingBlockProperties-Handler, TransBlockPropertiesHandler und ReprBlockPropertiesHandler. Für die Darstellung der Block-Eigenschaften sind die im Entwurf genannten Klassen implementiert. Durch die Darstellung der Eigenschaften als JFrame- Fenster werden die Klassen jedoch um die nötigen Attribute und Methoden ergänzt, um diese Funktionalität zu bieten.

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 diie 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 jetztigen Version der Anwendung keine Anwendung, sind aber für eine mögliche Weiterentwicklung notwendig, um bestimmte Funktionalitäten zu bieten.

3.2.8 Controller-Interface-Paket

Das Contoller-Interface-Paket wurde nach Entwurfsvorgaben implementiert. Einzelne Methoden aller Interfaces finden in der jetztigen 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

Die Rolle, die der Controller einnimmt, hat sich nicht geändert.

3.3.1 Anpassung an Model-View-Controller

In dem ursprünglichen Entwurf war der Controller nicht klar von einem *Presenter* abgegrenzt. Aus diesem Grund wurden alle Aufrufe, von dem Model, über den Controller, zur GUI, entfernt. Dadurch erfüllt die vorgesehene Rolle nun eindeutig.

3.3.2 Command zum Speichern von Daten

Ursprünglich waren für das Speichern der Bausteine unterschiedliche Commands vorgesehen. Namentlich waren diese:

EditBlockPropertiesCommand, ModifyChannelConnectionCommand, RemoveBlockFromConfigCommand, ExportBlockPrototypeCommand, CloneBlockCommand

Diese Commands wurden in `SavaDataCommand` zusammengefasst. Durch das Zusammenfassen werden bei dem Speichern von Änderungen auch die Informationen mit übergeben, die sich nicht geändert haben. Wir haben uns für dieses Vorgehen entschieden, weil die Datenhaltung über einen zentralen Punkt abgehandelt werden kann.

3.4 Backend

3.4.1 SSH-Verbindung

Für die Kommunikation mit dem Raspberry Pi wird eine SSH-Verbindung verwendet. So können über das lokale Netzwerk Phython-Skripte auf dem Raspberry Pi ausgeführt und Dateien kopiert werden.

Die SSH-Verbindung ist mithilfe der Bibliothek *SSHJ*[1] implementiert. Infolge dieser Technologieentscheidung sind folgende Klassen überflüssig geworden: `SystemProcessCommandLine`, `SshCommandGetSensorIds`, `SshCommandCopyFromPi`, `SshCommandCopyToPi`.

Die Aufgaben dieser Klassen werden stattdessen durch ihre vorgesehenen Oberklassen (`CommandGetSensorIds`, `CommandCopyFromi`, `CommandCopyTo`) und direkt durch `SshToPi` erfüllt. Letztere verwendet dabei *SSHJ*.

Nach außen ist der Zugriff auf das Backend über die Schnittstellen `IAccessToSensorInfo` und `IAccessToMeasurementRun` und die implementierenden Klassen `SensorInfoAgent`

[1] `https://github.com/hierynomus/sshj`

und `MRunAgent`, sowie die Klasse `PickupPointForBackendAgents` möglich. Diese wurden wie im Entwurf vorgesehen umgesetzt.

3.4.2 Simulierte Verbindung

Anstelle einer echten SSH-Verbindung kann das Backend auch simuliert werden. Dazu wurden die Klassen `ComToFile` und `FileCommandFactory` eingeführt. Es können damit unter anderem Messdaten aus einer zuvor erstellten Datei ausgelesen werden.

Die abstrakte Fabrik aus dem Entwurf, die nach wie vor auch zur Realisierung weiterer Anbindungsarten verwendet werden kann, wurde also zur simulierten Kommunikation unter Einsatz des lokalen Dateisystems erweitert. Hierbei wurde jedoch nicht jedes Kommando dupliziert, sondern lediglich die Austauschbarkeit des Kommunikationshilfsmittel genutzt.

Die Simulation des Backends ist nur eingeschränkt funktionstüchtig, da die Verbindung mit einem echten Gerät in der Entwicklung eine deutlich höhere Priorität hatte.

3.4.3 Nebenläufigkeit

Das Auslesen von Daten läuft parallelisiert ab. Dabei wird für jeden Sensor ein eigener `Thread` aufgebaut. (Nicht, wie im Entwurf angedeutet, für jeden Kanal.)

Jeder dieser Threads verwendet eine Instanz der Klasse `MeasurementRunnable`, welche die Klassen `CommandMRun` und `SshCommandMRun` aus dem Entwurf ersetzt. Über diese *Runnables* wird der Messlauf gesteuert (unter anderem pausiert und angehalten). Die Klasse `MRunThread` ist dabei überflüssig geworden.

3.5 Cache

Um Übersichtlichkeit und Handhabbarkeit zu verbessern, wurde im Cache-Modul, so wie auch im Backend-Modul, der Paketstruktur ein Subpackage hinzugefügt.

Die Attributierung der Cache-Klassen bzgl. des Timings wurde während der Implementierung stark verändert. Im Entwurf war sie über das gesamte Cache-Modul verteilt. Nun ist sie auf die Cache-Klasse konzentriert. In der Cache-Klasse wurde also etwas mehr an Funktionalität gekapselt als ursprünglich vorgesehen.

Private Methoden, die im Entwurf zur Deklaration und Strukturierung klasseninterne Funktionalität enthalten waren, sind nur noch zum Teil in der Implementierung enthalten. An Ihrer Stelle wurden neue Utility-Methoden hinzugefügt. Auch die Struktur und Art der Einbindung von Hilfsklassen hat sich geändert, genauso wie die Platzierung der Run-Methode, die für den Cache-Thread benötigt wird.

Die wesentlichen Strukturen bzgl. der Check-and-Notify-Aktionen, die Weiterleitungsstruktur per Listener, sowie der Thread-Aspekt des Moduls wurden wie vorgesehen implementiert.

Die Cache-Klasse selbst ist der Kern des Cache-Moduls. Hier erwies sich die Synchronisierung zu Beginn eines Messlaufs als die größte Herausforderung, und entsprechend gibt hierzu nun eine ganze Reihe von cache-internen Hilfsmethoden.

3.6 File-Service

CsvService Die Klasse CsvService wurde in der Implementierung durch die Methoden zum Umbennen und Exportieren von CsvDateien erweitern. 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 Umorgansiation 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, entsprechend der bereits genannten Gründe für diese Abweichungen. Die Darstellung soll Abhängigkeiten zwischen den Implementierungsschritten und kritische Pfade wiedergeben.

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

Dazu ist in Abbildung 14 das entsprechende Gantt-Diagramm zu sehen. In der oberen Zeile lassen sich die Meilenstein, mit den geplanten Fristen, erkennen. In der Tabelle

wird der Fertigstellungstermin der jeweiligen Komponente mit dem jeweiligen Kürzel bezeichnet. Wenn ein Meilenstein in Klammern steht, bedeutet dies, dass das Ziel nicht innerhalb der Implementierungszeit erreicht werden konnte.

Abbildung 14: Der reale Verlauf der Implementierung anhand eines Gantt-Diagramms

5 Glossar

Abstract Window Toolkit Ein GUI Toolkit..

allgemeinen Bausteinprototyp-Informationen Dazu gehören folgende Eigenschaften: Name, Typ, Configurations ID, Initialisierungs ID, Nutzerinformationen (Hilfetext).

Graphical Editing Framework Ein GUI Framework..

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..

Presenter Presenter bezieht sich auf die gleichnahmige Komponente in einem Model-View-Presenter..

Standard Widget Toolkit Ein GUI Framework..

Swing Ein GUI Toolkit..