



# Netzwerkbasierte Lichtsteuerung von DMX-fähigen Endgeräten im Heimbereich

von

**Sebastian Böttger**

Matrikelnummer: 018 104 030

Bachelor-Arbeit im Studiengang  
Technische Informatik KIA  
zur Erlangung des akademischen Grades  
Bachelor of Engineering

**Eingereicht am:** Heiligenhaus, den 02. April 2024

**1. Prüfer:** Prof. Dr.-Ing. Christian Weidauer

**2. Prüfer:** Prof. Dr. rer. nat. Peter Gerwinski

# Abstract

Diese Ausarbeitung befasst sich mit der Beleuchtung im Heimbereich und deren Steuerung, was als Heimilluminationssystem bezeichnet wird. Das Heimilluminationssystem besteht aus einer lokalen Lichtsteuereinheit, welche DMX-fähige Lampen über DMX ansteuert und einer Lichtsteuersoftware, mit der die Lampen gesteuert werden.

Es wird dargestellt, dass eine netzwerkbasierte Kommunikation zwischen der Lichtsteuersoftware und der Lichtsteuereinheit vorteilhaft ist. Diese ermöglicht eine Ortsunabhängigkeit zwischen den beiden Komponenten und die gleichzeitige Nutzung von mehreren Lichtsteuereinheiten an unterschiedlichen Orten. Die Ausarbeitung erläutert, warum Art-Net ein geeignetes Netzwerkprotokoll für diesen Anwendungsfall ist.

Sie beschreibt, wie das Art-Net-Protokoll in die lokale Lichtsteuereinheit und in die Lichtsteuersoftware integriert werden kann. Darüber hinaus erläutert sie die Konzepte hinter der Gestaltung der Benutzeroberfläche und zeigt, wie der Softwarekern der Lichtsteuersoftware mit Hilfe des Model-View-ViewModel-Entwurfsmuster mit der Benutzeroberfläche verbunden ist.

Abschließend werden die aktuellen Herausforderungen durch einen Leistungsengpass der Lichtsteuereinheit beim Verarbeiten von Art-Net-Paketen und fehlerhafte Animationen und Listenaktualisierungen in der Benutzerstelle beschrieben.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Aufgabenstellung und Zielsetzung . . . . .	2
1.3	Vorgehensweise und Aufbau der Arbeit . . . . .	2
<b>2</b>	<b>Lichtsteuerung im Heimbereich</b>	<b>4</b>
2.1	Architektur des Heimilluminationssystems . . . . .	4
2.2	DMX-512 als Steuerprotokoll . . . . .	6
2.3	Vorteile einer netzwerkbasieren Kommunikation . . . . .	8
<b>3</b>	<b>Netzwerkkommunikationsprotokoll des Heimilluminationssystems</b>	<b>10</b>
3.1	Anforderungen des Kommunikationsprotokolls . . . . .	10
3.2	Art-Net . . . . .	13
3.2.1	Verbindungsabbau zu einem Node . . . . .	14
3.2.2	Übertragen von DMX-Werten . . . . .	17
3.2.3	Übertragen von zusätzlichen Informationen . . . . .	17
3.3	sACN bzw. ANSI E1.31-2016. . . . .	20
3.4	MA-Net . . . . .	21
3.5	Evaluation und Auswahl des Protokolls . . . . .	22
<b>4</b>	<b>Aufbau der lokalen Lichtsteuereinheit</b>	<b>24</b>
4.1	Hardwareaufbau . . . . .	24
4.2	Datenstrukturen . . . . .	27
4.3	Integration des Netzwerkkommunikationsprotokolls . . . . .	30
<b>5</b>	<b>Aufbau der Lichtsteuersoftware</b>	<b>36</b>
5.1	Softwarearchitektur . . . . .	36
5.2	Benutzerschnittstelle . . . . .	40
5.3	Integration des Netzwerkkommunikationsprotokolls . . . . .	44
5.4	Realisierung des Model-View-ViewModel-Entwurfsmusters . . . . .	48
<b>6</b>	<b>Aktuelle Herausforderungen</b>	<b>52</b>
6.1	Leistungsengpass der Lichtsteuereinheit . . . . .	52
6.2	Grafikanimationen und Listenaktualisierungen . . . . .	52
<b>7</b>	<b>Fazit und Ausblick</b>	<b>54</b>
	<b>Anhang</b>	<b>55</b>
A	Literaturverzeichnis . . . . .	55

B	Abbildungsverzeichnis . . . . .	56
C	Tabellenverzeichnis . . . . .	56
D	Abkürzungsverzeichnis . . . . .	57
E	Quellcodeverzeichnis . . . . .	57

# 1 Einleitung

In der Vergangenheit entwickelte der Autor bereits im Rahmen des Software-Entwicklungsprojekts und des Moduls *Algorithmen und Datenstrukturen* eine Lichtsteuersoftware und eine Lichtsteuereinheit, die über Universal Serial Bus (USB) mit der Software verbunden ist. Die Lichtsteuereinheit steuert angeschlossene Lampen über das Protokoll Digital Multiplex (DMX)-512 (vgl. [Bö23b] und [Bö23a]).

Die folgende Ausarbeitung stellt dar, wie die Lichtsteuersoftware mit der Lichtsteuereinheit anstelle über USB über ein lokales Netzwerk kommuniziert.

## 1.1 Motivation

Die Internationale Funkausstellung (IFA) ist eine Messe, bei der Hersteller ihre neuesten Produkte und Konzepte, unter anderem in den Bereichen Home-Entertainment, Multimedia und Haushaltselektronik, vorstellen. Der Begriff „Smart Home“ hat hierbei in den letzten Jahren immer mehr an Präsenz dazugewonnen. Alle Geräte sind untereinander vernetzt, um Prozesse und Abläufe im Alltag zu optimieren. Ein Teilbereich dieser Thematik ist die Lichtsteuerung. Die Lampe im Wohnzimmer kann nun mit der Handy-Applikation (App) eingeschaltet werden und leuchtet in der zum Film passenden Farbe. Nachteilig an vielen Produkten im Bereich der Lichtsteuerung ist die Bindung an den Hersteller und die singuläre Art der Steuerung. Viele Produkte verschiedener Hersteller sind ohne weiteres nicht miteinander kompatibel, da die Kommunikationsprotokolle meist proprietär und die Spezifikationen dieser nicht immer öffentlich zugänglich sind. Zudem ermöglichen viele Apps nur das Steuern einzelner Lampen, so dass diese nicht in Kombination oder Gruppen gesteuert werden können.

Das Software-Entwicklungsprojekt des Autors beschreibt, zusammen mit der Ausarbeitung im Modul *Algorithmen und Datenstrukturen*, einen Ansatz, mit dem eine Heimbeleuchtung mit Hilfe des Steuerprotokolls DMX-512 ermöglicht wird und somit vom Hersteller unabhängig ist. Im Rahmen dieser Arbeiten wurde mit Hilfe des menschenzentrierten Gestaltungsprozesses eine Nutzungsweise eines sogenannten Heimilluminationssystems ermittelt. Auf Grundlage der ermittelten Nutzungsweise wurde eine Lichtsteuersoftware und eine Lichtsteuereinheit entwickelt. Zusammen ermöglichen diese einem Benutzer eine gruppenbasierte (Gruppe von Lampen) Lichtsteuerung von DMX-fähigen Endgeräten.

In der bisherigen Systemarchitektur kommuniziert die Lichtsteuersoftware mit der Lichtsteuereinheit über USB. Somit muss sich die Lichtsteuereinheit in der Nähe des Computers befinden, auf dem die Steuersoftware ausgeführt wird, um eine Verbindung herzustellen. Das verwendete Kommunikationsprotokoll über USB ist proprietär, wodurch die Nutzung von anderer Hard- oder Software erschwert wird. Auch ist es in

der bisherigen Systemarchitektur nicht möglich, mehrere Lichtsteuereinheiten mit der Software zu verbinden.

Eine netzwerkbasierte Kommunikation mit Hilfe eines bereits etablierten Protokolls zur Übertragung von DMX-Werten ermöglicht eine Lichtsteuerung mit mehreren, orts-unabhängigen Lichtsteuereinheiten, die sowohl mit Hard- als auch Softwareprodukten weiterer Hersteller kombiniert werden kann.

## **1.2 Aufgabenstellung und Zielsetzung**

In dieser Arbeit soll das zuvor verwendete USB-Kommunikationsprotokoll zwischen der Lichtsteuersoftware und der Lichtsteuereinheit durch ein netzwerkbasiertes Kommunikationsprotokoll ersetzt werden. Im Zuge dieser Änderungen werden Anpassungen an der Lichtsteuersoftware und Lichtsteuereinheit durchgeführt.

Im Software-Entwicklungsprojekt hat der Autor den Softwarekern (Programmlogik mit einer Kommandozeilen-Eingabe) separat von der Benutzerschnittstelle entwickelt. In dieser Arbeit soll mit dem Model-View-ViewModel-Entwurfsmuster eine Verbindung zwischen dem Softwarekern und der Benutzerschnittstelle erstellt werden.

Genau wie im Software-Entwicklungsprojekt ist die Zielgruppe dieses Systems die der Hobby- und Amateurbastler. Benutzer müssen das System um weitere Endgeräte und Systemeigenschaften erweitern können. Somit sollten möglichst offene Standards gewählt werden, die Hobby- und Amateurbastlern leicht nutzen können.

Ziel dieser Arbeit ist eine Lichtsteuersoftware für den Heimbereich, die über ein standardisiertes Netzwerkprotokoll mit der Lichtsteuereinheit kommunizieren kann und eine Benutzerschnittstelle besitzt, die nach den Design- und Interaktionsprinzipien des Curriculum CPUX-DS und der Norm DIN EN ISO 9241-110 gestaltet wurde.

Unter Beachtung des Umfangs der Arbeit wird angenommen, dass nur ein Typ von Endgerät verwendet wird. Dies setzt den Schwerpunkt der Arbeit auf die Kommunikation zwischen der Lichtsteuersoftware und den Lichtsteuereinheiten, schränkt die Funktionalität der Software aber nicht in großem Maße ein.

## **1.3 Vorgehensweise und Aufbau der Arbeit**

Einleitend stellt die Arbeit in Kapitel 2 die Architektur eines Heimilluminationssystems dar, damit die folgenden Themen korrekt verstanden und eingeordnet werden können. Hierbei werden das Prinzip des Steuerprotokolls DMX-512 grundlegend erläutert und die Vorteile der Verwendung einer netzwerkbasierte Steuerung in Bezug auf das aktuelle Heimilluminationssystem begründet.

Weiterführend werden in Kapitel 3 die Anforderungen an ein Kommunikationsprotokoll zwischen Lichtsteuersoftware und den Lichtsteuereinheiten dargestellt. Diese sind auf Grundlage der im menschenzentrierten Gestaltungsprozess erarbeiteten Nutzungsanfor-

derungen entwickelt. Anschließend werden die Protokolle *Art-Net*, *sACN* bzw. *ANSI E1.31-2018* und *MA-Net/ETCNet* dargestellt und mit den Anforderungen verglichen. Abgeschlossen wird das Kapitel mit der Evaluation und der damit verbundenen Auswahl eines Protokolls.

Kapitel 4 dokumentiert die Struktur der lokalen Steuereinheit mit netzwerkbasiertem Kommunikationsprotokoll. Im Kapitel ist sowohl der Aufbau der Hardware dokumentiert, als auch die Darstellung und Erläuterung der verwendeten Datenstrukturen. Ebenfalls wird erläutert, wie das Netzwerkprotokoll in die Lichtsteuereinheit integriert ist.

Ergänzend dazu beschreibt Kapitel 5 die Strukturen der Lichtsteuersoftware. Das Kapitel erläutert die Zusammenhänge der Softwarearchitektur und der Datenstrukturen und stellt die verschiedenen Aspekte der Benutzerschnittstelle dar. Auch zeigt das Kapitel, wie die netzwerkbasierte Kommunikation in der Software integriert ist und wie der Softwarekern mit der Benutzerschnittstelle mit Hilfe des Model-View-ViewModel-Entwurfsmusters verbunden ist.

Die Herausforderungen, die bei hoher Netzwerkbelastung der Lichtsteuereinheit entstehen und bei Animationen von Grafikelementen sowie bei Aktualisierungen von Listen der Benutzeroberfläche, werden in Kapitel 6 beschrieben.

Abschließend stellt Kapitel 7 das Fazit zum Projekt dar und gibt einen Ausblick für weitere Optimierungen in der Zukunft.

## 2 Lichtsteuerung im Heimbereich

Das Thema dieser Arbeit liegt im Kontext der Beleuchtung im Heimbereich. Damit die Inhalte zur Lichtsteuereinheit und der Lichtsteuersoftware besser eingeordnet und verstanden werden können, wird im folgenden Kapitel 2.1 die grundlegende Architektur eines Heimilluminationssystems erläutert. Bestandteil der Architektur ist das Steuerprotokoll DMX-512, welches in Kapitel 2.2 detaillierter erklärt wird. Aufbauend auf den Titel der Arbeit, stellt Kapitel 2.3 die Vorteile einer netzwerkbasierter Kommunikation innerhalb eines Heimilluminationssystems dar.

### 2.1 Architektur des Heimilluminationssystems

Beleuchtungssysteme existieren in den verschiedensten Bereichen. Eine sehr große Sparte ist zum Beispiel die Beleuchtung bei Veranstaltungen, wie Konzerten, Musicals und weiteren Events. Diese Systeme arbeiten häufig mit sehr vielen verschiedenen Endgeräten, die eine Menge an steuerbaren Eigenschaften (Farbe, Helligkeit, Ausrichtung der Lampe, Fokus, Zoom etc.) besitzen. Diese Endgeräte werden vielseitig eingesetzt und bei der Steuerung werden verschiedene Typen von Endgeräten miteinander kombiniert, um die passende Beleuchtung für die Veranstaltung zu erhalten.

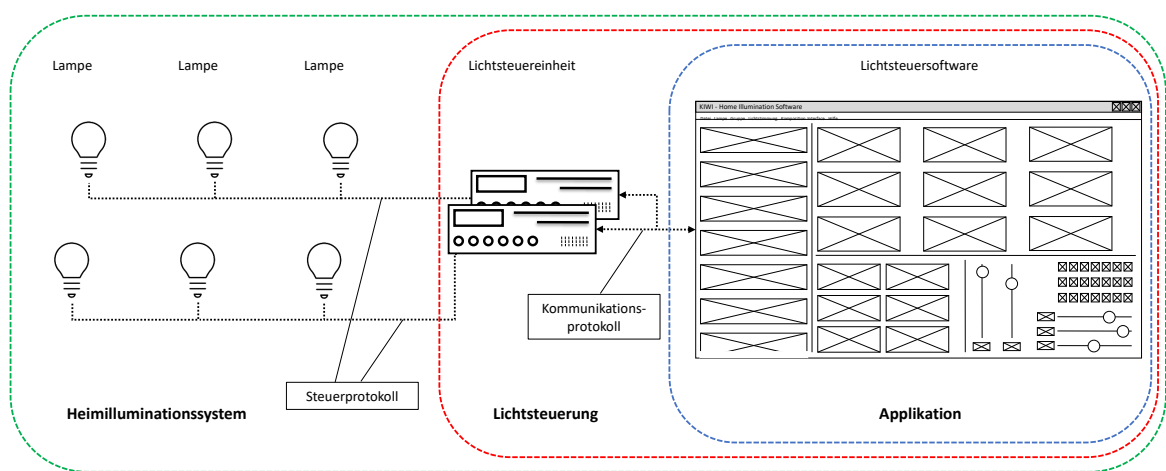
Dem gegenüber stehen Beleuchtungssysteme in Industrien und Einrichtungen, wie zum Beispiel Fabrikhallen oder Schulen. Diese Systeme steuern zentral die Lichtquellen im Gebäude und besitzen häufig Funktionen, wie zeitgesteuertes Ein- und Ausschalten von Lampen in einem bestimmten Bereich. Die Endgeräte dieser Systeme sind meist recht simpel und besitzen nur eine Helligkeitsregelung. Einzelne Lampen werden als Gruppe zusammengeschlossen, um diese gemeinsam zu steuern. So werden in einer Schule zum Beispiel alle Lampen in der Mensa ab 15 Uhr ausgeschaltet. Die Lampen in den Klassenräumen ab 18 Uhr und ab 20 Uhr leuchten nur noch einzelne Lampen in den Fluren, um die Notbeleuchtung zu gewährleisten.

Die Beleuchtung im Heimbereich (vor allem im Kontext von „Smart Home“) nutzt Elemente aus beiden genannten Systemen. Das Software-Entwicklungsprojekt des Autors beschreibt einen Nutzungskontext und eine Nutzungsweise für ein Beleuchtungssystem im Heimbereich, die durch den menschenzentrierten Gestaltungsprozess erarbeitet wurden. In diesem Nutzungskontext wird beschrieben, dass Lampen im Heimbereich häufig mehrere Funktionen besitzen. Es müssen die Farbe, Helligkeit und eventuell auch Effekte, wie Blinken eingestellt werden können. Die Steuerung dieser Lampen erfolgt gruppenweise. Die meisten Gruppen von Lampen entsprechen realen Abgrenzungen, wie Räumen, Etagen oder sonstigen Bereichen. Die Gruppen werden unabhängig voneinander gesteuert, so dass die Steuerung des Lichts im Wohnzimmer nicht das Licht in der Küche beeinflusst. Innerhalb einer Gruppe werden Lampen zum einen identisch angesteuert



(alle Lampen erhalten die selben Eigenschaftswerte für Farbe, Helligkeit und Effekt) und zum anderen aber auch individuell (alle Lampen erhalten eigene Eigenschaftswerte für Farbe, Helligkeit und Effekt).

Der letzte Aspekt des Nutzungskontexts ist die dezentrale Steuerung. Das Licht muss dort gesteuert werden können, wo der Nutzer sich aufhält. Somit reicht eine zentrale Steuerkonsole oder Steuersoftware nicht aus. Der Benutzer benötigt die Möglichkeit, am aktuellen Standort eine zuvor eingestellte Konfiguration der Lampen abzurufen. Mit Grundlage dieses Nutzungskontexts und den daraus entsprechenden Nutzungserfordernissen entsteht das sogenannte *Heimilluminationssystem*. Abbildung 1 zeigt die Systemarchitektur des Heimilluminationssystems.



**Abbildung 1** Systemarchitektur eines Heimilluminationssystems

Das Heimilluminationssystem ist in drei ineinander verschachtelte Bereiche aufgeteilt. Der innerste Bereich ist die *Applikation*. Bei der Applikation handelt es sich um eine Lichtsteuersoftware, mit der die Werte für die verschiedenen Lampen eingestellt und gespeichert werden können. Beim Speichern der Werte gibt es zwei Möglichkeiten:

**Lichtstimmung** Bei der Lichtstimmung werden allen Lampen einer Gruppe die selben Eigenschaftswerte zugewiesen. Sie kann genutzt werden, um einen Raum in einer einheitlichen Farbe und Helligkeit zu beleuchten.

**Komposition** Bei einer Komposition werden jeder Lampe einer Gruppe individuelle Eigenschaftswerte zugewiesen. Sie kann genutzt werden, um einen Raum präzise zu beleuchten.

Der nächste Bereich ist die *Lichtsteuerung*. Diese umfasst die Lichtsteuersoftware und die Lichtsteuereinheiten. Die Lichtsteuersoftware kommuniziert mit den Lichtsteuereinheiten über ein festgelegtes Kommunikationsprotokoll. Eine Lichtsteuereinheit hat zwei

Funktionen. Zum einem übermittelt sie mit Hilfe eines Steuerprotokolls die von der Lichtsteuersoftware festgelegten Eigenschaftswerte an die Lampen. Zum anderen stellt sie dem Benutzer Schnellauswahltasten zur Verfügung, mit denen dieser im Vorhinein ausgewählte Lichtstimmungen oder Kompositionen einer Gruppe aktivieren oder deaktivieren kann. Die Kombination aus Gruppe und Lichtstimmung oder Komposition heißt *Preset*.

Der letzte Bereich ist das Heimilluminationssystem selbst. Es umfasst die gesamte Lichtsteuerung und alle damit verbundenen Endgeräte. Im Rahmen dieser Ausarbeitung wird nur ein Typ von Endgerät verwendet. Hierbei beschränkt sich die Funktion des Endgerätes auf die drei Farbwerte rot, grün und blau, den Helligkeitswert und die Blinkrate.

## **2.2 DMX-512 als Steuerprotokoll**

Das Steuerprotokoll DMX-512 (abgekürzt auch nur DMX genannt) wird durch die Norm DIN 56930-2 definiert (vgl. [DIN00]). Das Protokoll dient „zur Verbindung von Lichtsteuerungen und Dimmern oder anderer angeschlossener Peripherie bei professioneller Anwendung in Veranstaltungs- und Produktionsstätten, z. B. in Theatern, Mehrzweckhallen, Studios und Diskotheken“([DIN00, S.1 Abschnitt 1]). Das Protokoll überträgt bis zu 512 sogenannter Kanäle, die zusammen als DMX-Universum oder DMX-Frame bezeichnet werden. Jeder dieser Kanäle enthält einen 8-Bit-Wert. Den angeschlossenen Endgeräten werden Kanalbereiche zugewiesen, um über diese Kanäle die Funktionen der Endgeräte zu steuern. Tabelle 1 zeigt einen Ausschnitt einer beispielhaften Belegung eines DMX-Universums.

Vorteile des DMX-Protokolls sind der simple Aufbau, die daraus resultierende Geräte- und Funktionsunabhängigkeit und die große Vielfalt an Produkten, die DMX unterstützen. Ein Kanal im Protokoll hat keine festgelegte Funktion, da die Funktionszuordnung des Kanals erst durch das Endgerät definiert wird. Somit ist das Protokoll nicht nur auf die Steuerung von Lampen beschränkt, sondern kann auch für weitere Endgeräte genutzt werden. Die Steuerung eines Rollladens oder einer Markise sind mögliche Anwendungsfälle. Auch können zwei oder mehr Kanäle zusammen genutzt werden um eine höhere Auflösung als 8-Bit zu erreichen. Nicht DMX-fähige Endgeräte können leicht mit Hilfe eines kleinen Mikrocontrollers auf DMX umgerüstet werden. Dies lädt viele Hobby- und Amateurbastler dazu ein, das Protokoll für kleine Heimprojekte zu verwenden.

Alternative Steuerprotokolle im Bereich von „Smart Home“ sind zum Beispiel *ZigBee* von der ZigBee Alliance und *Matter* von der Connectivity Standards Alliance (CSA). Diese Protokolle unterstützen die Ansteuerung von zahlreichen Smart-Home-Geräten. Allerdings haben diese Protokolle bezogen auf das Ziel dieser Ausarbeitung einige Nachteile gegenüber DMX-512.

DMX-Kanal	LED-Scheinwerfer	LED Moving Head	4-Kanal Dimmer
1	Intensität rot		
2	Intensität grün		
3	Intensität blau		
4	Intensität gesamt		
5	Blinkrate		
6		Intensität rot	
7		Intensität grün	
8		Intensität blau	
9		Intensität gesamt	
10		Pan	
11		Tilt	
12			Intensität Kanal 1
13			Intensität Kanal 2
14			Intensität Kanal 3
15			Intensität Kanal 4

**Tabelle 1** Beispielhafte Kanalbelegung eines DMX-Universums

Der Aufbau von Matter und ZigBee ist recht komplex und beinhaltet viele Mechanismen und Methoden, die es für Hobby- und Amateurbastler erschweren das Protokoll selbstständig im Heimnetz umzusetzen. Beide Protokolle verfügen über Verschlüsselungsprinzipien und Authentifizierungsprozesse, um eine sichere Kommunikation mit vertrauenswürdigen Endgeräten zu erreichen (vgl. [Allc, Kapitel 4] und [Alla, Kapitel 2.7]). Allerdings werden für diese Prozesse teilweise Zertifikate benötigt, die nur von den Urhebern des Protokolls (z. B. der CSA) ausgestellt werden können (vgl. [Alla, Kapitel 2.6]). Da das DMX-Protokoll als kabelgebundene Kette zwischen den Endgeräten aufgebaut wird, werden hierbei keine weiteren Sicherheitsprozesse benötigt.

Häufig geben die Protokolle auch nur den Rahmen der Kommunikation vor, aber definieren nicht genauer in welcher Weise die eigentlichen Nutzdaten übertragen werden. Diese Struktur ist somit weiterhin herstellerabhängig und muss auch nicht öffentlich zugänglich sein (vgl. [Allc, Kapitel 1.1.3]). Eine Ausnahme bildet hier Matter. Matter definiert mit sogenannten Device-Typen und Clustern, welche Daten für welche Art von Endgerät, wie übertragen werden. Dies ist allerdings gleichermaßen auch wieder

ein Nachteil, da somit nur vom Protokoll fest definierte Gerätetypen gesteuert werden können (vgl. [Allb, Kapitel 3, S. 29]). Durch die kanalbasierte Steuerung ermöglicht es DMX, vollkommen geräte- und funktionsunabhängig zu sein.

Für diese Arbeit wird, wie in Kapitel 1.2 beschrieben, nur ein Typ von Endgerät verwendet. Es handelt sich dabei um das Gerät *Stairville xBrick Full-Colour Fluter 16x3W LED*. Diese Lampe kann mit 5 DMX-Kanälen gesteuert werden. Die Belegung der Kanäle ist Tabelle 2 zu entnehmen.

DMX-Belegung xBrick LED Fluter	
DMX-Kanal	Funktion
1	Intensität rot
2	Intensität grün
3	Intensität blau
4	Intensität gesamt
5	Blinkrate

**Tabelle 2** DMX-Kanalbelegung der Stairville xBrick Full-Colour Fluter

## 2.3 Vorteile einer netzwerkbasierten Kommunikation

Beim ursprünglichen Heimilluminationssystem wurde die Lichtsteuersoftware mit Hilfe eines proprietären Kommunikationsprotokolls über USB mit der Lichtsteuersoftware verbunden. Durch diese Verbindung entstehen folgende Nachteile:

- Die Lichtsteuereinheit muss sich in der Nähe des Computers befinden
- Proprietäre Protokolle erschweren die Nutzung unterschiedlicher Hard- und Software
- Es ist nicht möglich, mehrere Lichtsteuereinheiten mit der Software zu verbinden
  - Dies liegt zum einen am Protokoll, aber auch an der begrenzten Anzahl der USB-Anschlüsse eines Computers

Mit der Umstellung auf ein netzwerkbasiertes Protokoll für die Kommunikation kann die Entfernung zwischen dem Computer mit der Lichtsteuersoftware und der Lichtsteuereinheit erhöht werden. Solange die Geräte Teilnehmer des selben Local Area Network (LAN) sind, ist deren geografische Entfernung zueinander irrelevant. Zudem ermöglicht die Topologie eines LANs das Verbinden von (fast) beliebig vielen Lichtsteuereinheiten.<sup>1</sup>

<sup>1</sup>Im privatem Umfeld ist ein /24-Netz üblich, was bis zu 256 Teilnehmer in einem Netz ermöglicht.

Die Verwendung eines bereits etablierten Standardprotokolls ermöglicht die Nutzung von weiterer Hard- und Software.

Um die Sicherheit in einem LAN zu gewährleisten, helfen in diesem Fall mehrere Aspekte. Ein Teilnehmer eines LANs muss entweder mit einem Kabel angeschlossen sein, oder erhält den Zugriff über Wireless Local Area Network (WLAN). Im Falle des Kabelanschlusses ist die Sicherheitsmaßnahme der physikalische Zugang zum Netzwerk. Ist dieser nicht gegeben, so kann keine Verbindung hergestellt werden. Im Falle einer WLAN-Verbindung kann durch die Nutzung einer geeigneten Authentifizierungsmethode sichergestellt werden, dass keine unbefugten Geräte ins Netzwerk gelangen.

Zusammenfassend lässt sich sagen, dass die Nutzung eines Netzwerks zur Kommunikation zwischen der Lichtsteuersoftware und Lichtsteuereinheit die genannten Nachteile der USB-Kommunikation ausgleicht und eine sichere und sinnvolle Lösung ist.

## 3 Netzwerkkommunikationsprotokoll des Heimilluminationssystems

Kapitel 2.3 erläutert, welche Vorteile eine netzwerkbasierte Kommunikation besitzt. In diesem Kapitel werden verschiedene Netzwerkprotokolle evaluiert und die Auswahl des Protokolls Art-Net wird erläutert. Die für die Evaluation der Protokolle benötigten Anforderungen werden in Kapitel 3.1 dargestellt. In den Kapiteln 3.2, 3.3 und 3.4 werden die zur Auswahl stehenden Protokolle vorgestellt, damit Kapitel 3.5 fundiert die Evaluation der Protokolle darstellen kann.

### 3.1 Anforderungen des Kommunikationsprotokolls

Die Anforderungen an das Heimilluminationssystem sind mit Hilfe einer Satzschablone formuliert, die auch als User-Story bekannt ist. Der Aufbau dieser Schablone ist im CPUX-UR Curriculum des UXQB e. V. definiert:

*„Der Benutzer muss am System eine **<Information oder Ressource>** erkennen/auswählen/eingeben können unter **<Bedingung (optional)>**.“*  
*[CU23, S. 68]*

Der „Benutzer“ steht in dieser Satzschablone stellvertretend für eine Benutzergruppe. Bei der Analyse des Nutzungskontextes ergeben sich folgende Benutzergruppen:

Benutzergruppe	Menschzentrierte Qualitätsziele
Heimanwender	<ul style="list-style-type: none"> <li>• Steuerung der Beleuchtung im Innen- und Außenbereich</li> <li>• Erstellen von Lichtstimmungen</li> <li>• Erhalten von Informationen zum aktuellen Beleuchtungszustand</li> </ul>
Installateur	<ul style="list-style-type: none"> <li>• Konfiguration der Steuerung für die vorhandenen Endgeräte</li> </ul>
Zuschauer	<ul style="list-style-type: none"> <li>• Anschauen einer schönen Beleuchtung</li> </ul>

**Tabelle 3** Übersicht der Benutzergruppen und deren menschzentrierte Qualitätsziele

Die menschzentrierten Qualitätsziele zeigen die Ziele der Benutzergruppen auf, die durch das System erfüllt werden sollen. Jede dieser Benutzergruppen hat Erfordernisse,

um ihre menschenzentrierten Qualitätsziele zu erfüllen.<sup>2</sup> Aus den ermittelten Erfordernissen werden anschließend die Anforderungen abgeleitet. Für das Dokumentieren der Anforderungen, kann die oben dargestellte Satzschablone genutzt werden. Folgende Anforderungen des Heimilluminationssystems betreffen die Kommunikation zwischen der Lichtsteuersoftware und der Lichtsteuereinheit bzw. haben Auswirkungen auf diese:

### **Gruppe**

- Der Heimanwender muss an der Lichtsteuersoftware eine Gruppe löschen können.
- Der Heimanwender muss an der Lichtsteuersoftware eine Gruppe aktivieren und deaktivieren können.
- Der Heimanwender muss an der Lichtsteuersoftware erkennen können, ob eine Gruppe aktiv, inaktiv oder teilaktiv ist.

### **Lichtsteuereinheit**

- Der Heimanwender muss an der Lichtsteuersoftware die Verbindung zur Lichtsteuereinheit herstellen können.
- Der Heimanwender muss an der Lichtsteuersoftware den Status der Verbindung (verbunden, nicht verbunden, Verbindung wird hergestellt) zur Lichtsteuereinheit erkennen können.
- Der Heimanwender muss an der Lichtsteuersoftware die Schnellauswahltasten der Lichtsteuereinheit konfigurieren können (einen Preset zuweisen).
- Der Heimanwender muss an der Lichtsteuereinheit Schnellauswahltasten aktivieren und deaktivieren können.
- Der Heimanwender muss an der Lichtsteuereinheit erkennen, welche Schnellauswahltasten aktiv, inaktiv oder teilaktiv sind.

### **Lichtstimmung**

- Der Heimanwender muss an der Lichtsteuersoftware eine Lichtstimmung löschen können.
- Der Heimanwender muss an der Lichtsteuersoftware eine Lichtstimmung aktivieren und deaktivieren können.
- Der Heimanwender muss an der Lichtsteuersoftware erkennen können, ob eine Lichtstimmung aktiv oder inaktiv ist.

### **Komposition**

- Der Heimanwender muss an der Lichtsteuersoftware eine Komposition löschen können.
- Der Heimanwender muss an der Lichtsteuersoftware eine Komposition aktivieren und deaktivieren können.

---

<sup>2</sup>Für die Beschreibung von Erfordernissen gibt es ebenfalls eine Satzschablone, die allerdings leicht von der Beschreibung der Anforderungen abweicht.

- Der Heimanwender muss an der Lichtsteuersoftware erkennen können, ob eine Komposition aktiv oder inaktiv ist.

### **Lampe**

- Der Heimanwender muss an der Lichtsteuersoftware eine Lampe einer Gruppe zuweisen können.
- Der Heimanwender muss an der Lichtsteuersoftware eine Lampe aus einer Gruppe entfernen können.

Für die Anforderungen an das Kommunikationsprotokoll gibt es keine konkreten Benutzer. Um dennoch die Satzschablone für die Formulierung von Anforderungen nutzen zu können, wird die *Lichtsteuersoftware* und *Lichtsteuereinheit* als „Benutzer“ verwendet. Es folgen nun die konkreten Anforderungen an das Kommunikationsprotokoll.

### **Preset**

- Die Lichtsteuersoftware muss ein Preset (Gruppen-Id und die DMX-Werte der in der Gruppe befindlichen Lampen) setzen können.
- Die Lichtsteuersoftware muss ein Preset der Lichtsteuereinheit löschen können (somit ist die Schnellauswahltaste funktionslos).
- Die Lichtsteuersoftware muss ein Preset der Lichtsteuereinheit aktivieren und deaktivieren können.
- Die Lichtsteuereinheit muss der Lichtsteuersoftware mitteilen können, dass ein Preset aktiviert oder deaktiviert wurde.
- Die Lichtsteuersoftware muss bei der Lichtsteuereinheit den aktuellen Status aller Schnellauswahltasten (aktiv, inaktiv oder teilaktiv) abfragen können.

### **Gruppe**

- Die Lichtsteuersoftware muss eine Zustandsänderung einer Gruppe der Lichtsteuereinheit mitteilen können.

### **DMX-Signale**

- Die Lichtsteuersoftware muss einen DMX-Frame an die Lichtsteuereinheit übertragen können (damit dieser an die Lampen übertragen wird).

Zusammengefasst ergeben sich für die Evaluation der Protokolle die folgenden drei Anforderungen:

1. Das Protokoll muss das Übertragen von DMX-Werten ermöglichen
2. Das Protokoll muss eine bidirektionale Kommunikation ermöglichen
3. Das Protokoll muss das Übertragen von zusätzlichen Informationen (Aktivieren/-Deaktivieren von Gruppen/Presets etc.) ermöglichen



## 3.2 Art-Net

Art-Net ist eines der verbreitetsten Protokolle im Bereich der Veranstaltungstechnik zur Übertragung von DMX-Werten über das Netzwerk.

„Art-Net is an Ethernet protocol based on the TCP/IP protocol suite. Its purpose is to allow transfer of large amounts of DMX512 data over a wide area using standard networking technology. Art-Net was invented by Wayne Howell, the founder of Artistic Licence.“ [Lic23, Seite. 6]

Die erste Version des Protokolls wurde 1998 vorgestellt und konnte bis zu 40 DMX-Universen per Broadcast-Verfahren übertragen. Die aktuellste Version ist Art-Net 4 und wurde mit dem PLASA Award for Innovation ausgezeichnet. Art-Net 4 bietet ein breites Spektrum an Funktionen. Es unterstützt bis zu 30.000 DMX-Universen, besitzt Verfahren zur automatischen Entdeckung von Art-Net-fähigen Geräten im Netzwerk und ermöglicht die Übertragung von Medien (Film und Foto) und des sogenannten Time-Codes.<sup>3</sup> (vgl. [Ltd24])

Art-Net 4 nutzt das Transportprotokoll User Datagram Protocol (UDP) um die Nutzdaten im Netzwerk zu versenden. Die versendeten Nutzdaten werden im folgenden als *Art-Net-Paket* oder auch kurz als Paket bezeichnet. UDP ist ein verbindungsloses Protokoll, welches keine Prüfung über das erfolgreiche Zustellen der versendeten Daten durchführt. Vorteilhaft an UDP ist die hohe Übertragungsrate, wodurch eine schnelle Steuerung der Endgeräte ermöglicht wird.

Diese Arbeit beschränkt sich auf die folgenden drei Aspekte des Protokolls:

- Verbindungsaufbau
- Übertragen von DMX-Werten
- Übertragen von zusätzlichen Informationen

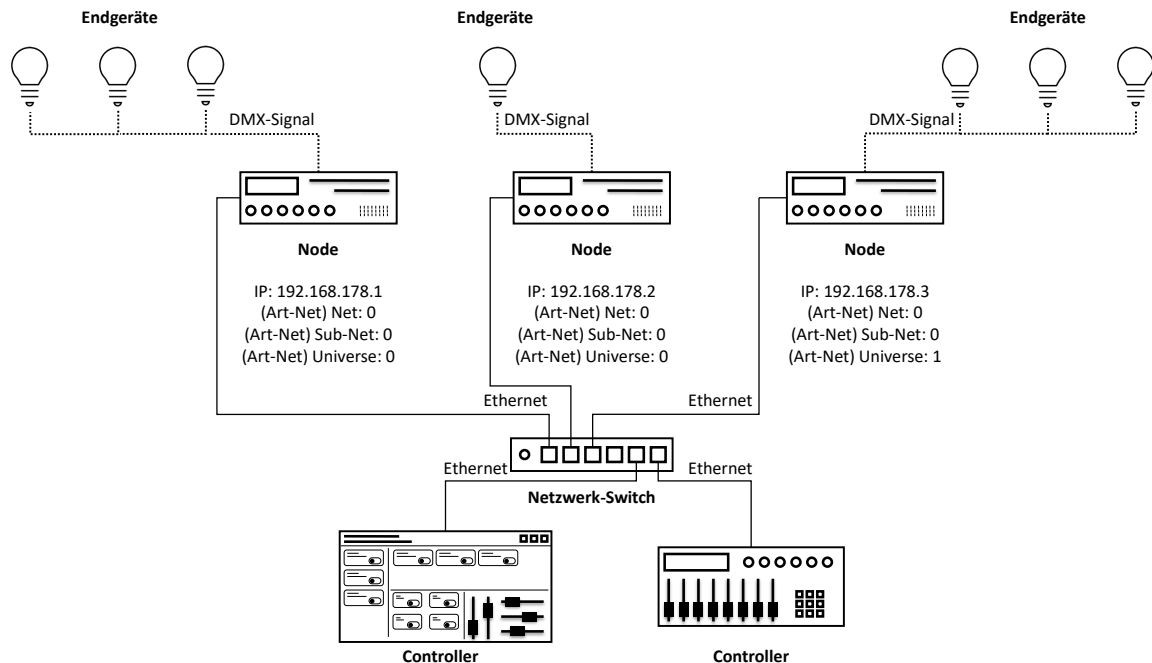
In der Spezifikation des Art-Net Protokolls werden unter anderem die Begriffe *Node* und *Controller* verwendet, die im folgenden erklärt werden:

<b>Node</b>	Ein Gerät, welches DMX-Daten von oder in das Art-Net-Protokoll übersetzt, wird als Node bezeichnet.
<b>Controller</b>	Ein Gerät zur zentralen Steuerung oder Überwachung (Lichtsteuerkonsole) ist ein Controller. Der Begriff wird auch generalisiert verwendet für alle Geräte, die Steuerungsdaten über Art-Net versenden.

**Tabelle 4** Definition der Begriffe Node und Controller. Sinnhaft übersetzt aus der Spezifikation (vgl. [Lic23, Seite. 9 f.] )

<sup>3</sup>eine Zeitinformation zur Synchronisierung von verschiedenen Geräten.

Abbildung 2 zeigt beispielhaft den Aufbau eines Art-Net-Netzwerks. Das Art-Net-Protokoll ermöglicht das parallele Nutzen von mehreren Controllern. In dieser Arbeit wird dies nicht benötigt, aber berücksichtigt.



**Abbildung 2** Beispielhafter Aufbau eines Art-Net Netzwerkes

Jeder Node besitzt eine einmalige IP-Adresse, über die der Node eindeutig identifiziert werden kann. Zusätzlich besitzt jeder Node noch die drei Eigenschaften *Net*, *Sub-Net* und *Universe*. Die Kombination aus diesen drei Werten heißt *Port-Address* und referenziert ein DMX-Universum eindeutig. Die Port-Address besteht aus 15 Bits und wird in zwei Bytes übertragen. Tabelle 5 stellt den Aufbau der Port-Address dar.

	Bit 15	Bit 14-8	Bit 7-4	Bit 3-0
Port-Address	0	Net	Sub-Net	Universe

**Tabelle 5** Aufbau der Port-Address

### 3.2.1 Verbindungsausbau zu einem Node

Für den Verbindungsaufbau stellt Art-Net zwei Typen von Paketen zur Verfügung. Zum einen das Paket *ArtPoll* und zum anderen das Paket *ArtPollReply*. Das Paket *ArtPoll* wird vom Controller per Broadcast im Netz versendet. Alle Art-Net-fähigen Endgeräte antworten anschließend mit einem *ArtPollReply* darauf. Mit diesem Vorgehen können neue Art-Net-fähige Geräte im Netz erkannt werden. Das *ArtPoll* Paket wird in

regelmäßigen Abständen versendet um auch den Verbindungsabbruch eines Gerätes zu erkennen.

Tabelle 6 zeigt eine gekürzte Version des Aufbaus des Art-Net-Pakets ArtPoll. Die Spalte *Größe* gibt hierbei den Datentypen an. Das Protokoll nutzt für alle Informationen einen unsigned Integer mit 8, 16 oder 32 Bit. Besteht ein Feld aus einem Array, so wird die Größe des Array in eckigen Klammern angegeben. Im wesentlichen besteht das Paket ArtPoll aus der *ID* (ein String zur Identifikation des Protokolls), dem *OPCode* zur Identifizierung des Pakettypen und der Versionsnummer (*ProtVerHi* und *ProtVerLo*).

Feld	Name	Größe	Beschreibung
1	ID	Int8[8]	Ein Array von acht Character, wobei der letzte Eintrag 0x00 entspricht. Feldwert = 'A' 'r' 't' '-' 'N' 'e' 't' 0x00
2	OpCode	Int16	Gibt den Nachrichtentyp an (siehe Tabelle 7). Aufbau nach Little-Endian-Prinzip. In diesem Fall <i>OpPoll</i> .
3	ProtVerHi	Int8	Höherwertiges Byte der Art-Net-Protokoll-Revisionsnummer.
4	ProtVerLo	Int8	Niederwertiges Byte der Art-Net-Protokoll-Revisionsnummer. Aktueller Wert ist 14.

**Tabelle 6** Gekürzter Aufbau des ArtPoll-Pakets (vgl. [Lic23, Seite. 17 f.]).

Name	Wert	Definition
OpPoll	0x2000	Dies ist ein ArtPoll Paket. Es sind keine weiteren Daten in diesem UDP-Paket enthalten.
OpPollReply	0x2100	Dies ist ein ArtPollReply Paket. Es enthält Statusinformationen des Gerätes.
OpCommand	0x2400	Dies ist ein ArtCommand Paket. Es enthält Textbasierte Befehle. Statusinformationen des Gerätes.
OpDataRequest	0x2700	Dies ist ein ArtDataRequest Paket. Es wird benutzt um Daten von einem Node anzufragen.
OpDataReply	0x2800	Dies ist ein ArtDataReply Paket. Es wird benutzt um Angefragte Daten zum Controller zu senden.
OpDmx	0x5000	Dies ist ein ArtDmx Paket. Es enthält die DMX-Werte eines DMX-Frames.

**Tabelle 7** Ein Auszug der OpCodes des Art-Net-Protokolls (vgl. [Lic23, Seite. 19-21]).

Tabelle 7 zeigt die für diese Arbeit relevanten OpCodes des Art-Net Protokolls. Mit diesen Codes wird der Typ des Art-Net-Pakets angegeben.

Erhält ein Node ein ArtPoll-Paket, antwortet es mit einem ArtPollReply-Paket. Ein gekürzter Aufbau des ArtPollReply-Pakets ist Tabelle 8 zu entnehmen.

Feld	Name	Größe	Beschreibung
2	OpCode	Int16	Gibt den Nachrichtentyp an (siehe Tabelle 7). Aufbau nach Little-Endian-Prinzip. In diesem Fall <i>OpPollReply</i> .
3	IP Address	Int8[4]	Ein Array mit den vier Zahlen der IPv4-Adresse des Nodes. Aufbau nach Big-Endian-Prinzip.
7	NetSwitch	Int8	Das erste Byte der Port-Address (siehe Tabelle 5).
8	SubSwitch	Int8	Bit 7-4 der Port-Address (siehe Tabelle 5).
20	PortTypes	Int8[4]	Dieses Array gibt an, wie die vier möglichen Ports genutzt werden. Für jeden Port wird ein Byte mit dem folgendem Aufbau zurückgegeben.
		Bit	Beschreibung
		7	Bit ist gesetzt, wenn der Port Daten aus dem Art-Net-Netzwerk ausgibt.
		6	Bit ist gesetzt, wenn der Port Daten aus dem Art-Net-Netzwerk empfängt.
		5-0	000000 = DMX512 000101 = Art-Net
		Ungenutzte Ports werden durch ein 0er-Byte gekennzeichnet.	
23	SwIn	Int8[4]	Gibt das Universe (Bit 7-4 der Port-Address) für die vier möglichen Eingangsports des Nodes an (siehe Tabelle 5).
24	SwOut	Int8[4]	Bit 7-4 der Port-Address für die vier möglichen Ausgangsports des Nodes (siehe Tabelle 5).

**Tabelle 8** Gekürzter Aufbau des ArtPollReply-Pakets (vgl. [Lic23, Seite. 24-32]).

Das ArtPollReply-Paket enthält die Information, welche DMX-Universen über dieses Node versendet werden. Jedes Node kann maximal vier *Ports* mit einem ArtPollReply rückmelden. <sup>4</sup> Ein Port stellt einen Signalein- und/oder Ausgang des Nodes dar. Feld 20

<sup>4</sup>Besitzt ein Node mehr als vier Ports, so müssen diese mit Hilfe des BindIndex in mehreren ArtPollReply-Paketen gemeldet werden.

des Pakets definiert die Signalrichtung und das Protokoll. Jedem Port wird ein DMX-Universum zugeordnet (siehe Feld 23 und 24). Erhält der Controller das ArtPollReply-Paket eines Nodes, so gilt dieser als verbunden.

### 3.2.2 Übertragen von DMX-Werten

Zum Übertragen der DMX-Werte wird das Paket *ArtDmx* verwendet. Ein gekürzter Aufbau ist Tabelle 9 zu entnehmen.

Feld	Name	Größe	Beschreibung
2	OpCode	Int16	Gibt den Nachrichtentyp an (siehe Tabelle 7). Aufbau nach Little-Endian-Prinzip. In diesem Fall <i>OpDmx</i> .
7	SubUni	Int8	Zweites Byte der PortAdress (siehe Tabelle 5).
8	Net	Int8	Erste Byte der PortAdress (siehe Tabelle 5).
9	LengthHi	Int8	Anzahl der übertragenen DMX-Kanäle (2-512). Höherwertiges Byte
10	Length	Int8	Anzahl der übertragenen DMX-Kanäle (2-512). Niederwertiges Byte
11	Data	Int8	Ein Array mit variabler Länge, welches die Werte der [Length] DMX-Kanäle enthält.

**Tabelle 9** Gekürzter Aufbau des ArtDmx-Pakets (vgl. [Lic23, Seite. 60 f.]).

Im ArtDmx-Paket wird die PortAdress für das entsprechende DMX-Universum angegeben (Feld 7 und 8), die Anzahl an übertragenen DMX-Kanälen (Feld 9 und 10) und die DMX-Werte selber.

### 3.2.3 Übertragen von zusätzlichen Informationen

Für diese Arbeit müssen der Lichtsteuereinheit (Node) zusätzliche Informationen mitgeteilt und Statusdaten abgefragt werden. Für diesen Zweck gibt es die Pakete *ArtDataRequest*, *ArtDataReply* und *ArtCommand*. Tabelle 10 zeigt den gekürzten Aufbau des ArtDataRequest-Pakets.

Im Paket ArtDataRequest wird mit Hilfe der Felder 9 und 10 ein DataRequest-Code übertragen. Dieser gibt an, was vom Controller angefragt wird. Für diese Arbeit ist der Code-Bereich von 0x8000 bis 0xffff relevant, da dieser Bereich für die herstellerspezifische Nutzung freigegeben ist. Somit kann in diesem Bereich ein Code zum Anfragen der Statusinformationen definiert werden. Tabelle 11 gibt einen Überblick über die möglichen DataRequest-Codes.

Feld	Name	Größe	Beschreibung
2	OpCode	Int16	Gibt den Nachrichtentyp an (siehe Tabelle 7). Aufbau nach Little-Endian-Prinzip. In diesem Fall <i>OpDataRequest</i> .
9	RequestHi	Int8	Kennzahl welche Daten angefragt werden (siehe Tabelle 11). Höherwertiges Byte.
10	RequestLo	Int8	Kennzahl welche Daten angefragt werden (siehe Tabelle 11). Niederwertiges Byte.

**Tabelle 10** Gekürzter Aufbau des ArtDataRequest-Pakets (vgl. [Lic23, Seite. 44 f.]).

Name	Wert	Definition
DrPoll	0x0000	Mit diesem Code wird angefragt, ob eine Datenabfrage möglich ist.
DrUrlProduct	0x0001	URL zur Produktseite des Nodes.
DrUrlUserGuide	0x0002	URL zur Benutzeranleitung des Nodes.
DrUrlSupport	0x0003	URL zur Hilfeseite des Nodes.
DrManSpec	0x8000- 0xffff	Hersteller spezifische Nutzung.

**Tabelle 11** Codes zum Anfragen von Daten (vgl. [Lic23, Seite. 45]).

Erhält ein Node nun eine Datenanfrage, so antwortet es mit einem ArtDataReply-Paket darauf. Tabelle 12 zeigt den gekürzten Aufbau eines ArtDataReply-Pakets. Feld 9 und 10 des Pakets geben erneut den DataRequest-Code an, damit die Daten korrekt zugeordnet werden können. Feld 11 und 12 geben die Länge des Datenarrays in Feld 13 an.

Feld	Name	Größe	Beschreibung
2	OpCode	Int16	Gibt den Nachrichtentyp an (siehe Tabelle 7). Aufbau nach Little-Endian-Prinzip. In diesem Fall <i>OpDataReply</i> .
9	RequestHi	Int8	Kennzahl welche Daten angefragt wurden (siehe Tabelle 11). Höherwertiges Byte.
10	RequestLo	Int8	Kennzahl welche Daten angefragt wurden (siehe Tabelle 11). Niederwertiges Byte.
11	PayLenHi	Int8	Gibt die Länge des Array der Nutzdaten (Feld 13) an. Höherwertiges Byte.
12	PayLenLo	Int8	Gibt die Länge des Array der Nutzdaten (Feld 13) an. Niederwertiges Byte.
13	PayLoad	Int8 [Length]	Datenarray der angefragten Daten.

**Tabelle 12** Gekürzter Aufbau des ArtDataReply-Pakets (vgl. [Lic23, Seite. 46 f.]).

Um Befehle, wie das Setzen von Presets oder das Aktivieren von Gruppen, übertragen zu können, wird das Paket ArtCommand verwendet. Mit diesem Paket wird ein textbasierter Befehl übertragen, der bis zu 512 Zeichen lang sein kann. Die Spezifikation empfiehlt den String nach dem folgendem Schema aufzubauen:

$$\text{Befehl} = \text{Data} \&$$

Das &-Zeichen dient als Trennzeichen zwischen mehreren Befehlen in einem String. Tabelle 13 zeigt einen gekürzten Aufbau des Pakets.

Feld	Name	Größe	Beschreibung
2	OpCode	Int16	Gibt den Nachrichtentyp an (siehe Tabelle 7). Aufbau nach Little-Endian-Prinzip. In diesem Fall <i>OpCommand</i> .
7	LengthHi	Int8	Anzahl der übertragenen Bytes. Höherwertiges Byte
8	LengthLo	Int8	Anzahl der übertragenen Bytes. Niederwertiges Byte
11	Data	Int8 [Length]	ASCII basierter Befehls-String.

**Tabelle 13** Gekürzter Aufbau des ArtCommand-Pakets (vgl. [Lic23, Seite. 52 f.]).

### 3.3 sACN bzw. ANSI E1.31-2016.

Das Protokoll streaming Architecture for Control Networks (sACN) wurde 2009 von der Entertainment Services and Technology Association (ESTA) in der Norm ANSI E1.31 definiert und stellt einen Standard dar, um DMX-Daten über ein Netzwerk zu übertragen. Das Protokoll basiert auf dem Standard Architecture for Control Networks (ACN), der in der Norm ANSI E.1.17 definiert ist. ACN nutzt UDP zur Netzwerkcommunication (vgl. [SA18, Seite 1]), wodurch nicht sichergestellt ist, dass ein Datenpaket ankommt. sACN selbst stellt auch keine weiteren Methoden für eine Verbindungsprüfung zur Verfügung. Insgesamt werden mit sACN drei Arten von Paketen übertragen:

- Data Packet
- Universe Discovery Packet
- Synchronization Packet

Für diese Arbeit ist nur das *Data Packet* relevant. Mit dem Data Packet können DMX-Daten versendet werden. Hierbei wird die Nummer des jeweiligen Universums und der damit verbundene DMX-Frame versendet. Tabelle 14 zeigt den Aufbau eines Data Packets. Die Spalte Größe gibt an, wie viele Bytes das Feld in Anspruch nimmt. Alle Daten in diesem Paket werden nach dem Big-Endian-Prinzip versendet.

Pos.	Name	Größe	Beschreibung
113- 114	Universe	2	Nummer des DMX-Universums
123- 124	Property value count	2	Anzahl der versendeten DMX-Kanäle
125- 637	Property values	1-513	DMX-Werte

**Tabelle 14** Gekürzter Aufbau des Data Packet (vgl. [SA18, Seite. 6 f.]).

Eine Methode, um weitere Informationen wie Befehle und Preset-Daten zu versenden, bietet sACN nicht. Es ist jedoch möglich, weitere Daten und Informationen über DMX-Kanäle in einem zweiten Universum zu versenden. Dies ist allerdings vom Protokoll so nicht vorgesehen und kann zu Problemen führen, wenn diese Kanäle von anderen Geräten für DMX genutzt werden.



## 3.4 MA-Net

Ein weiteres verbreitetes Netzwerkprotokoll zur Übertragung von DMX-Daten ist von der Firma MA Lighting Technology GmbH. Das sogenannte *MA-Net3* ist die neueste Version des Kommunikationsprotokolls, welches alle Geräte (Lichtsteuerkonsolen, Nodes, Medienserver, Netzwerk-Switches) der Firma für die Kommunikation nutzen. Mit diesem Protokoll, können sowohl DMX-Daten, als auch weitere Nutzdaten übertragen werden. Abbildung 3 zeigt eine beispielhaften Systemarchitektur von Steuergeräten und Lichtsteuereinheiten, welche mit MA-Net3 verbunden sind. MA-Net3 ermöglicht die gleichzeitige Nutzung von mehreren Controllern und Nodes, die über Switches miteinander kommunizieren. Aktuell sind die Spezifikationen des Protokolls nicht öffentlich zugänglich.

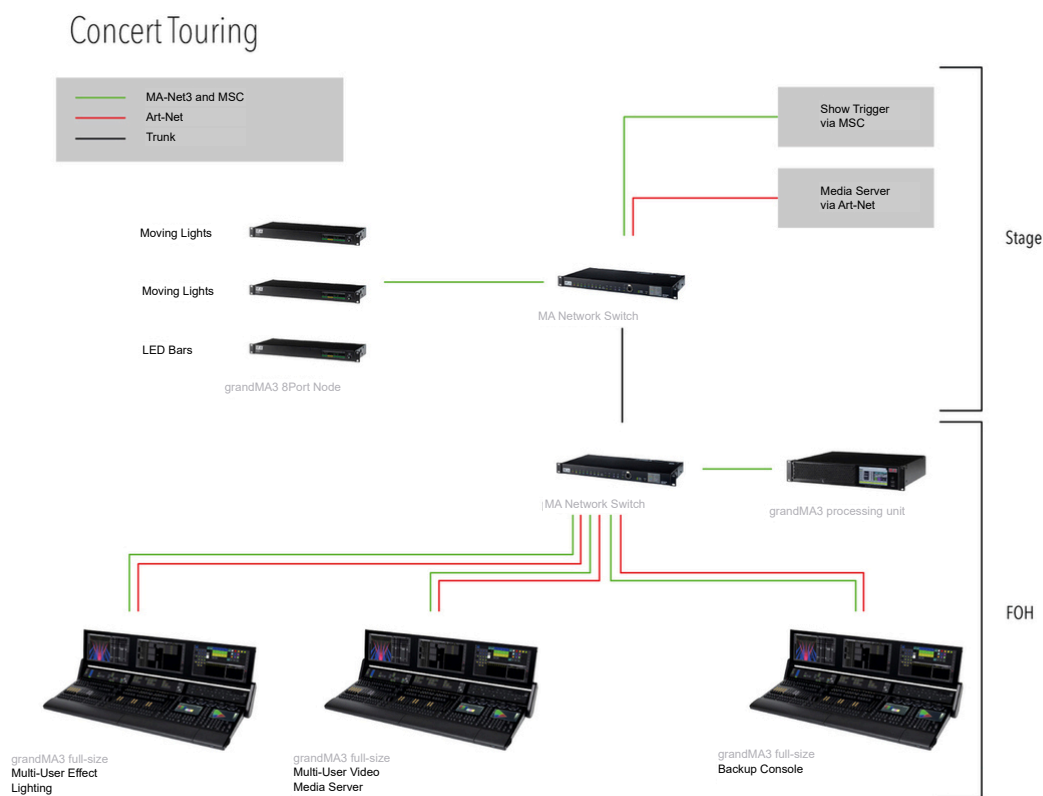


Abbildung 3 Beispielaufbau einer MA-Net-Systemarchitektur.<sup>5</sup>

Ein ähnliches Protokoll ist das ETCNet der Firma Electronic Theatre Controls, Inc. Auch zu diesem Protokoll sind die Spezifikationen nicht öffentlich zugänglich.

<sup>5</sup>Bildquelle: <https://www.malighting.com/de/ma-network-solutions>

## 3.5 Evaluation und Auswahl des Protokolls

Für die Evaluation der Protokolle beschreibt Kapitel 3.1 die fachlichen Anforderungen an das Protokoll. Das Protokoll muss DMX-Daten übertragen können, die Kommunikation muss bidirektional sein und es müssen zusätzliche Informationen wie das Setzen von Presets übertragen werden können. Hinzu kommt die Anforderung, dass die Protokollspezifikation öffentlich zugänglich sein muss, damit das Protokoll auf eigener Hard- und Software genutzt werden kann. Ebenfalls fließt in die Bewertung mit ein, ob es bereits Bibliotheken der entsprechenden Protokolle für den Arduino (Mikrocontroller für die Lichtsteuereinheit) und Java (Programmiersprache der Lichtsteuersoftware) gibt.

Tabelle 15 stellt das Ergebnis der durchgeführte Nutzwertanalyse dar.

Anforderungen	Art-Net	sACN	MA-Net3
Übermittlung von DMX-Daten	✓	✓	✓
Bidirektionale Kommunikation	✓	✗	✓
Übertragung von weiteren Informationen	✓	(✓)	✓
Öffentlich zugängliche Protokollspezifikation	✓	✓	✗
Bibliotheken für den Arduino	✓	✓	(✓)
Bibliotheken für Java	✓	✓	✗
Ergebnis	6 / 6	4,5 / 6	3,5 / 6
✓ : erfüllt      (✓) : teilweise erfüllt      ✗ : nicht erfüllt			

**Tabelle 15** Nutzwertanalyse der drei Protokolle Art-Net, sACN und MA-Net3.

Alle drei Protokolle besitzen Methoden, um DMX-Daten in ausreichender Menge und Geschwindigkeit zu übertragen. Eine bidirektionale Verbindung ist bei Art-Net und MA-Net3 möglich. Bei sACN ist keine bidirektionale Verbindung vorgesehen. Auch beim Übertragen von zusätzlichen Informationen stellt, sACN im Gegensatz zu Art-Net und MA-Net3 keine konkrete Methode zur Verfügung. Es ist zwar möglich, weitere Daten in Form von DMX-Kanalwerten in weiteren Universen zu übertragen, dies wäre aber recht aufwändig und nicht im Sinne des Protokolls. Der größte Nachteil von MA-Net3 besteht darin, dass die Spezifikationen des Protokolls nicht öffentlich zugänglich sind.

Dies macht eine private Nutzung des Protokolls mit eigener Hard- und Software so gut wie unmöglich. Für die beiden Protokolle Art-Net und sACN existieren bereits Bibliotheken, um diese Protokolle auf dem Arduino und in Java nutzen zu können. Für MA-Net3 gibt es vereinzelt auch Bibliotheken, mit denen der Arduino mit den MA-Net3 interagieren kann. Da die Spezifikation des Protokolls nicht öffentlich ist, basieren diese Bibliotheken häufig nur auf dem analysierten Netzwerkverkehr eines aktiven Systems. Somit bieten diese Bibliotheken keine Garantie, dass das Protokoll tatsächlich korrekt genutzt bzw. imitiert wird.

Zusammenfassend erfüllt Art-Net sechs von sechs Kriterien, sACN vier Kriterien, eins nur teilweise und MA-Net3 erfüllt drei Kriterien und eins nur teilweise. Die Nutzwertanalyse zeigt somit, dass Art-Net das optimale Protokoll für diesen Anwendungsfall ist. Es erfüllt alle fachlichen Anforderungen aus Kapitel 3.1, ist ein weitverbreiteter Standard, dessen Spezifikationen öffentlich zugänglich sind und es existieren bereits Bibliotheken, um das Protokoll auf dem Arduino und in Java zu nutzen.

## 4 Aufbau der lokalen Lichtsteuereinheit

Die lokale Lichtsteuereinheit ermöglicht den Benutzern mit Tastern, Presets zu aktivieren oder deaktivieren, wodurch der Benutzer das Heimilluminationssystem auch ohne aktiv laufender Lichtsteuerungssoftware steuern kann. Zusätzlich erzeugt die lokale Lichtsteuereinheit auch das DMX-Steuersignal, das an die Lampen gesendet wird. Es bildet somit das Bindeglied zwischen der Lichtsteuersoftware und den Endgeräten.

Das Kapitel 4.1 beschreibt den Aufbau der Hardwarekomponenten und gibt Hinweise, was bei diesem zu beachten ist. Kapitel 4.2 stellt die Datenstrukturen des Mikrocontrollers dar und erläutert, wie die Daten der Presets im Electrically Erasable Programmable Read-Only Memory (EEPROM) gespeichert werden. Abschließend beschreibt das Kapitel 4.3, wie die Datenstruktur genutzt wird, um Art-Net-Pakete zu empfangen, auszuwerten und zu versenden.

### 4.1 Hardwareaufbau

Die Lichtsteuereinheit besteht aus dem Mikrocontroller Arduino Mega 2560, dem Arduino Ethernet Shield 2, sechs Schnellauswahltasten (bestehend aus mehreren Komponenten) und dem Chip MAX 485.

Der Arduino verarbeitet die Signale der Schnellauswahltaster, und sendet das entsprechende DMX-Signal an die Endgeräte. Für das Erzeugen des DMX-Signals verwendet die Lichtsteuereinheit den Chip MAX 485. Der Arduino sendet das DMX-Signal mit Hilfe der Spannungen 0 V für eine LOW-Byte und 5 V für ein HIGH-Byte. Um eine durch das DMX-Protokoll vorgegebene Spannung von mindestens  $\pm 1,5$  V bis maximal  $\pm 5$  V zu erhalten, wandelt der Chip MAX 485 die beiden Spannungen vom Arduino in  $\pm 2,5$  V um. Dieser symmetrische Spannungspegel dient zur Fehlererkennung im Protokoll. Empfängt ein Endgerät 0 V anstelle der  $\pm 2,5$  V, so ist sofort erkennbar, dass eine Störung in der Leitung vorliegt.

Ein Schnellauswahltaster besteht aus einem Drucktaster und einer RGB-LED. Für die Betreibung der RGB-LED sind zusätzlich noch drei Widerstände mit einem Widerstandswert von  $220\ \Omega$  in Verwendung. Alle Drucktaster sind auf einem Pin mit dem Arduino verbunden und auf dem anderen mit Ground. Der Arduino steuert die Drucktaster mit dem Pin Mode `INPUT_PULLUP` an. Bei diesem Pin Mode liegt eine geringe Spannung am entsprechenden Pin an und ein interner Widerstand ist in Reihe geschaltet. Ist der Pin mit Ground verbunden, fließt ein geringer Strom, wodurch das Eingangssignal auf HIGH steht.

Das Ethernet Shield ist auf dem Arduino aufgesteckt und über die ICSP-Pins des Arduinos verbunden. Beim Arduino Mega 2560 sind diese gleichgeschaltet mit den Digital Pins 50 bis 53, weswegen diese im weiteren Verlauf nicht für weitere Zwecke zu nutzen sind.

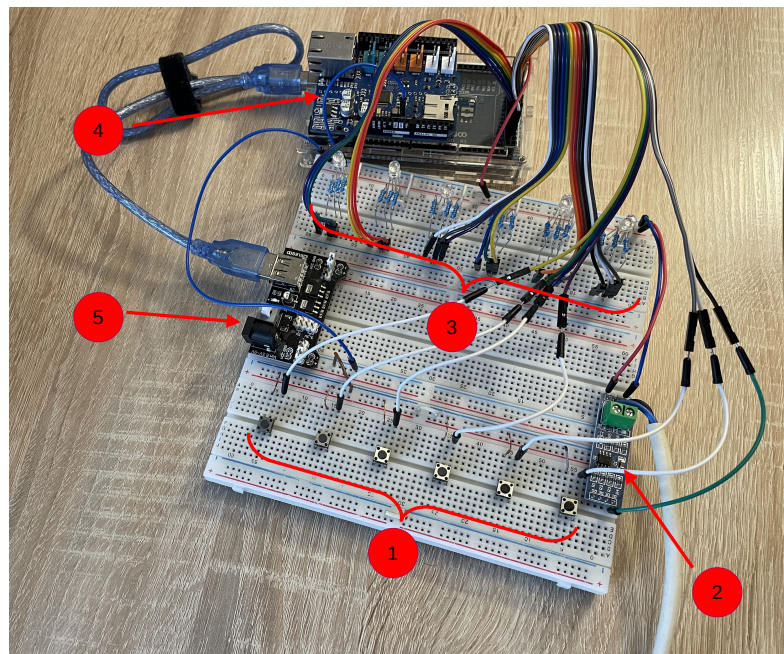
Alle Pinbelegungen der Hardwarekomponenten sind zentral in der Datei `config.h` verwaltet. Sie enthält alle allgemeinen, statischen Werte der Lichtsteuereinheit. <sup>6</sup> In Quellcode 1 ist ein Ausschnitt der `config.h` abgebildet. In der Datei ist unter anderem die Anzahl der angeschlossenen Presets hinterlegt, damit global die Größe des damit verbundenen Arrays definiert ist.

<sup>6</sup>Es existieren noch weitere Dateien mit statischen Werten, die aber spezifisch für einen Teil der Lichtsteuereinheit benötigt werden, z. B. die OpCodes des Art-Net-Protokolls.

```
15 #define BUTTON_2_PININ 34
16 #define BUTTON_3_PININ 32
17 #define BUTTON_4_PININ 30
18 #define BUTTON_5_PININ 28
19 #define BUTTON_6_PININ 26
20
21 // dmx pins
22 #define DMX_PINOUT 24
23
24 // preset state values
25 #define P_DEACTIV 0
26 #define P_ACTIVE 1
27 #define P_PART_ACTIVE 2
28
29 // common group id values
30 #define DEFAULT_GROUP_ID 0
31
32 [...]
```

**Quellcode 1** Auszug aus der Konfigurationsdatei config.h.

Abbildung 5 zeigt den realen Aufbau der Lichtsteuereinheit. Am unteren Ende des Breadboards befinden sich die sechs Drucktaster, welche mit Ground verbunden sind (1). Rechts daneben befindet sich der Chip Max 485, mit dem das DMX-Signal erzeugt wird (2). Am oberen Ende des Breadboards befinden sich sechs RGB-LEDs, zum Anzeige des Presetstatus (3). Darüber befindet sich der Arduino



**Abbildung 5** Foto des realen Aufbaus der Lichtsteuereinheit.

Mega 2560 mit dem Ethernet Shield(4). Links in der Mitte des Breadboards befindet sich ein Modul zur Stromversorgung des Arduinos(5).



## 4.2 Datenstrukturen

Die Datenstruktur der Lichtsteuereinheit lässt sich in vier grobe Bereiche unterteilen. Abbildung 6 zeigt ein einfaches Klassendiagramm, ohne Attribute oder Methoden der Klassen. Die gestrichelten Kästchen rahmen die vier Bereiche Hauptapplikation, Presets, DMX und Art-Net ein. Klassen mit gepunkteten Rahmen stellen dar, dass es sich um eine Datei mit statischen Werten oder um eine oder mehrere Strukturen handelt.

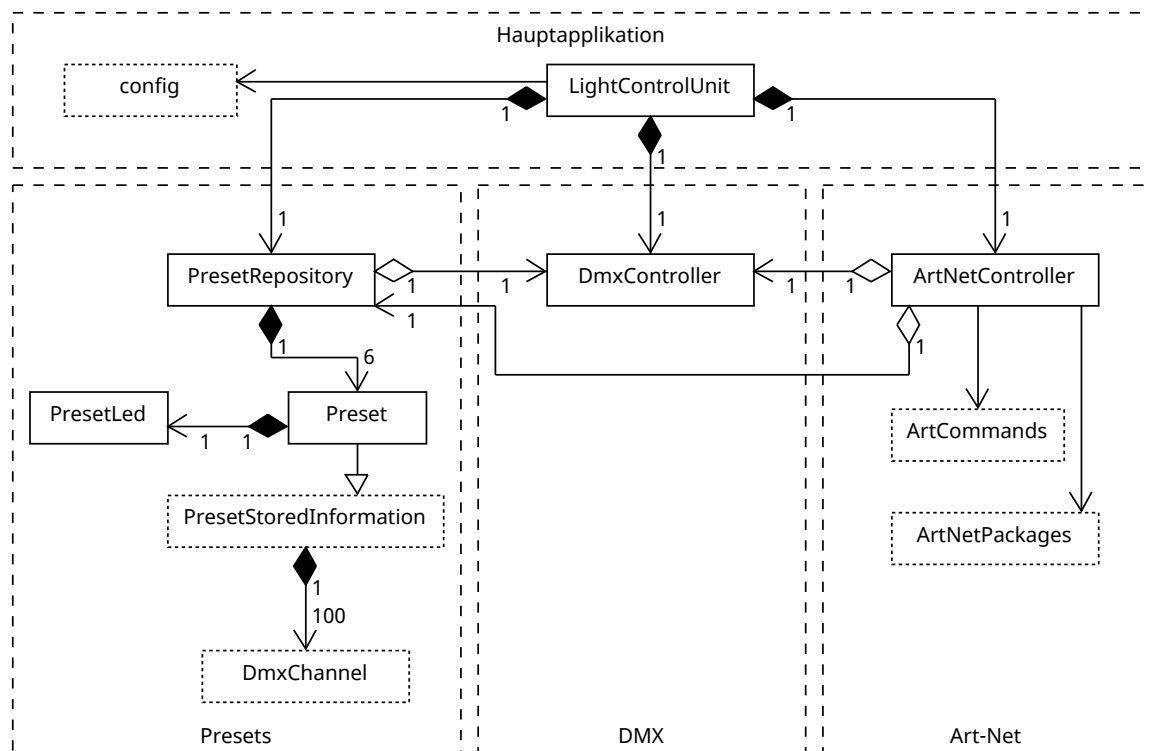


Abbildung 6 Einfaches Klassendiagramm der Lichtsteuereinheit.

Der Bereich Hauptapplikation enthält die Ino-Datei `lightControlUnit.ino`, die die Setup- und Loop-Methode des Arduinos beinhaltet. In diesem Bereich ist auch die Datei `config.h` enthalten, die, wie bereits erwähnt, alle allgemeinen statischen Werte enthält. Die `LightControlUnit` erstellt und besitzt jeweils ein Objekt der Klassen `PresetRepository`, `DmxController` und `ArtNetController`.

Im Bereich Presets sind alle Klassen und Strukturen enthalten, die für die Steuerung und Speicherung der sechs Presets benötigt werden. Die kleinste Einheit eines Presets ist die Struktur `DmxChannel`, die in Abbildung 7 dargestellt ist. Sie besteht aus dem `uint8_t`-Wert des DMX-Kanals und einer `bool`, die angibt, ob dieser Kanal vom Preset genutzt wird. Die nächstgrößere Einheit ist die Struktur `PresetStoredInformation`.

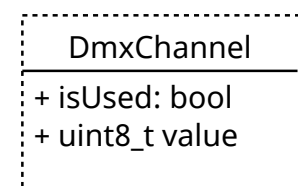


Abbildung 7 Aufbau der Struktur `DmxChannel`.

Diese beinhaltet alle Informationen, die von einem Preset auf dem EEPROM abgespeichert werden müssen. Abbildung 8 zeigt den Aufbau der Struktur. Sie enthält ein Array von 100 DMX-Kanälen, die Gruppen-Id und einen `uint8_t`-Wert für den Status des Presets. Die drei Zustände inaktiv, aktiv und teilaktiv werden durch die Zahlen 0, 1 und 2 dargestellt. Diese Werte sind global in der `config.h`-Datei definiert (vgl. Zeile 25-27 Quellcode 1).

Die EEPROM-Bibliothek vom Arduino ermöglicht es, mit der Methode `put()` eine Struktur beliebiger Größe (aber kleiner als die Speicherkapazität des EEPROMs) im EEPROM zu speichern.

Die Struktur `PresetStoredInformation` belegt 202 Byte (100 Kanäle mit je 2 Byte plus 2 Byte für State und Gruppen-Id) an Speicherplatz. Die Lichtsteuereinheit besitzt sechs Presets und benötigt somit eine Speicherkapazität von 1.212 Byte. Der EEPROM des Arduino Mega besitzt eine Speicherkapazität von 4.096 Byte. Die benötigte Speicherkapazität ist somit ausreichend vorhanden.

Preset
<ul style="list-style-type: none"> <li>- <code>epromAddress</code>: int</li> <li>- <code>presetLed</code>: <code>PresetLed</code></li> <li>- <code>usedChannel</code>: Set</li> </ul>
<ul style="list-style-type: none"> <li>+ <code>Preset()</code></li> <li>+ <code>Preset(epromAddress: int, presetLed: PresetLed)</code></li> <li>+ <code>getState(): uint8_t</code></li> <li>+ <code>getGroupId(): uint8_t</code></li> <li>+ <code>getDmxChannel(): DmxChannel*</code></li> <li>+ <code>setState(state: uint8_t): void</code></li> <li>+ <code>setGroupId(groupId: uint8_t): void</code></li> <li>+ <code>setChannelValue(channel: int, value: uint8_t): void</code></li> <li>+ <code>clearPreset(): void</code></li> <li>+ <code>loadPresetData(): void</code></li> <li>+ <code>savePresetData(): void</code></li> <li>+ <code>updateState(activeChannel: Set, activeGroupMember: bool): void</code></li> </ul>

**Abbildung 9** Aufbau der Klasse `Preset`.

enthält die Nummern der Kanäle, die im Preset verwendet werden. Die Methode `updateState` vergleicht dieses Set mit dem Set der aktiven Kanäle der DMX-Ausgabe. Enthält das Set der aktiven Kanäle eine oder mehrere Kanalnummern des Presets, so wird der State des Preset auf teilaktiv gesetzt. Falls nicht, ist das Preset inaktiv.

Die Klasse `PresetRepository` ist für die Verwaltung der Presets zuständig. Diese aktiviert oder deaktiviert die Presets, übermittelt deren DMX-Werte an den DMX-Controller und wird vom Art-Net-Controller angesteuert, um Preset-Werte zu setzen oder den State des Presets zu aktualisieren. Wird ein Preset aktiviert, prüft die Klasse, ob weitere Presets derselben Gruppe deaktiviert werden müssen oder ob andere Presets nun teilaktiv sind. Abbildung 10 zeigt die Attribute und Methoden der Preset-Repository-Klasse.

PresetStoredInformation
<ul style="list-style-type: none"> <li>+ <code>state</code>: <code>uint8_t</code></li> <li>+ <code>groupId</code>: <code>uint8_t</code></li> <li>+ <code>dmxChannels</code>: <code>DmxChannel[100]</code></li> </ul>

**Abbildung 8** Aufbau der Struktur `PresetStoredInformation`.

Da die Klasse `Preset` als Oberklasse von der Struktur `PresetStoredInformation` erbt, ist es möglich, das Preset-Objekt als Struktur zu casten und im EEPROM zu speichern. Der Aufbau der Klasse `Preset` ist in Abbildung 9 dargestellt. Die Hauptfunktion der Klasse ist das Speichern und Laden der Information eines Presets. Das Set `usedChannel`



PresetRepository
<ul style="list-style-type: none"> <li>- presets: Preset[6]</li> <li>- eepromStartAddress: int</li> <li>- dmxController: DmxController*</li> <li>- sendDmxValuesOfPreset(presetId: int): void</li> <li>- disableOtherGroupPresets(presetId: int): void</li> <li>- isGroupMemberActive(groupId: uint8_t): bool</li> </ul>
<ul style="list-style-type: none"> <li>+ PresetRepository()</li> <li>+ PresetRepository(eepromStartAddress: int, dmxController: DmxController*, leds: int**);</li> <li>+ getPreset(): Preset*</li> <li>+ getPresetState(presetId: int): uint8_t</li> <li>+ setPresetActive(presetId: int): void</li> <li>+ setPresetDeactiv(presetId: int): void</li> <li>+ setPresetGroupId(presetId: int, groupId: uint8_t): void</li> <li>+ setPresetChannelValue(presetId: int, channel: int, value: uint8_t): void</li> <li>+ activatePreset(presetId: int): void</li> <li>+ deactivatePreset(presetId: int): void</li> <li>+ activateGroup(groupId: uint8_t): void</li> <li>+ deactivateGroup(groupId: uint8_t): void</li> <li>+ clearPreset(presetId: int): void</li> <li>+ updatePresetState(): void</li> <li>+ updatePresetState(groupId: uint8_t): void</li> </ul>

**Abbildung 10** Aufbau der Klasse **PresetRepository**.

Der Bereich DMX besteht nur aus der Klasse **DmxController**. Dieser besitzt ein Array an DMX-Werten und ein Set mit den Nummern der aktiven Kanäle.

Er wird vom Preset-Repository oder vom Art-Net-Controller aufgerufen um DMX-Werte zu setzen bzw. zu senden. Um den State der Presets korrekt bestimmen zu können, muss beim Setzen von DMX-Werten auch immer mit angegeben werden, ob der Kanal aktiv ist. Dies wird durch einen zusätzlichen Pa-

DmxController
<ul style="list-style-type: none"> <li>- dmxOut: uint8_t[100]</li> <li>- activeChannel: Set</li> </ul>
<ul style="list-style-type: none"> <li>+ DmxController()</li> <li>+ getActiveChannel(): Set</li> <li>+ setChannelStatus(isActive: bool, channel: int): void</li> <li>+ setChannelValue(channel: int, value: uint8_t): void</li> <li>+ setChannelValue(isActive: bool, channel: int, value: uint8_t): void</li> </ul>

**Abbildung 11** Aufbau der Klasse **DmxController**.

rameter beim Methodenaufruf oder durch den Aufruf einer weiteren Funktion mitgeteilt. Abbildung 11 zeigt den Aufbau der Klasse **DmxController**.

Der letzte Bereich (Art-Net) besteht aus der Klasse **ArtNetController** und den zwei .h-Dateien, die statische Werte und Strukturen für den Art-Net-Controller enthalten. Der Art-Net-Controller nutzt die Bibliothek **EthernetUDP** vom Arduino, um ein UDP-Socket zu erstellen und auf dem Port 6454 nach Art-Net-Paketen zu lauschen. Um die Pakete korrekt interpretieren oder versenden zu können, enthält die Datei **ArtNetPackageTypes.h** eine Struktur für jeden benötigten Pakettypen. Die mit dem Paket *ArtCommand* versendeten Befehle sind als JSON formatiert. Damit dieser JSON-

String erstellt und auch wieder ausgewertet werden kann enthält die Datei **ArtCommands** für jeden möglichen Befehl den JSON-Aufbau als Struktur.

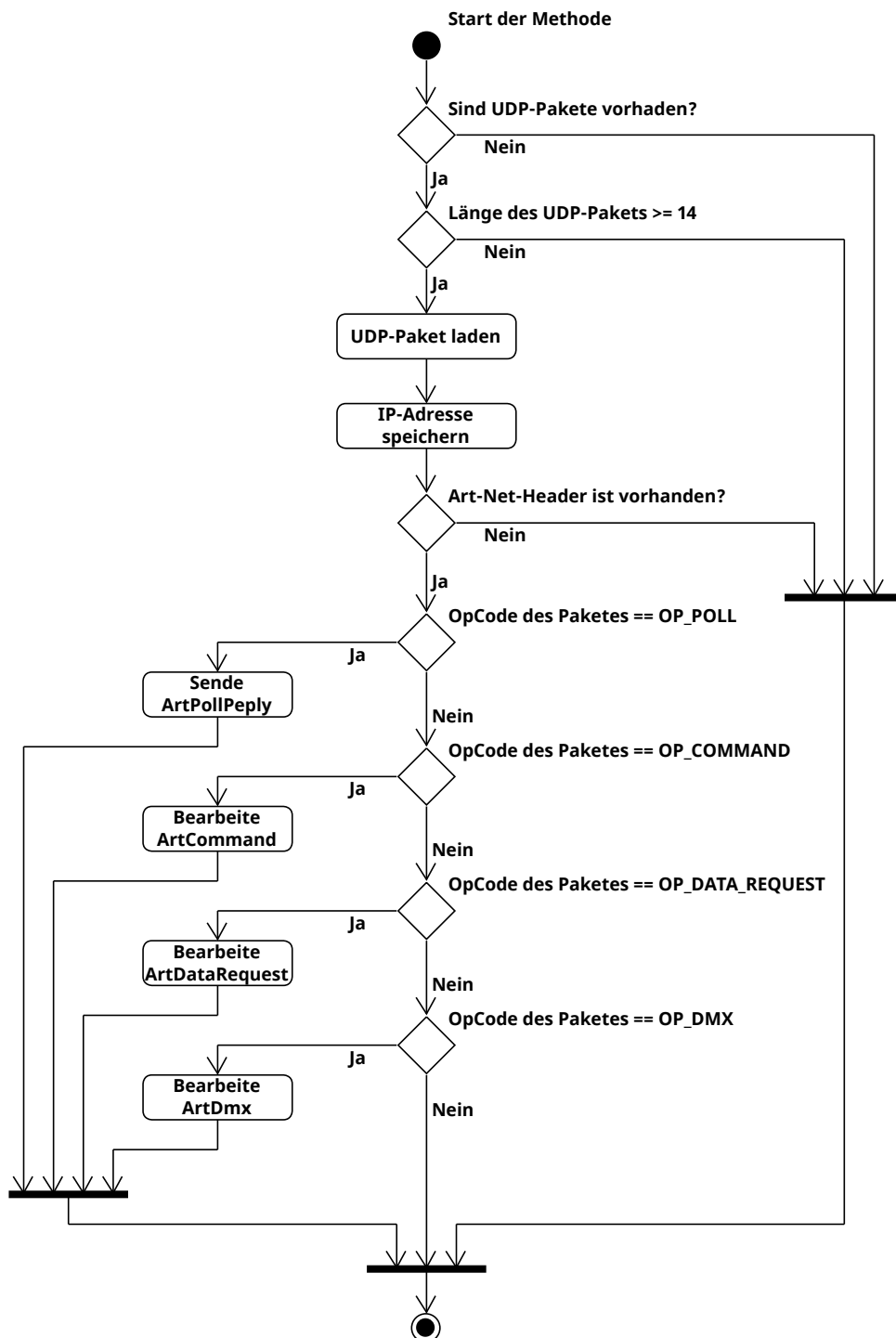
ArtNetController
<ul style="list-style-type: none"> <li>- dmxController: DmxController*</li> <li>- presetRepository: PresetRepository*</li> <li>- mac: uint8_t[6]</li> <li>- udp: EthernetUDP</li> <li>- nodeIpAddress: uint8_t[4]</li> <li>- controllerIpAddress: uint8_t [4]</li> </ul>
<ul style="list-style-type: none"> <li>- startUdp(): int</li> <li>- getPresetStatus(): uint8_t*</li> <li>- doc: StaticJsonDocument&lt;50&gt;</li> <li>- checkArtNetHeader(udpData: uint8_t[]): bool</li> <li>+ ArtNetController()</li> <li>+ ArtNetController(dmxController: DmxController*, presetRepository: PresetRepository*)</li> <li>+ listenToArtNet(): void</li> <li>+ sendArtPollReply(): void</li> <li>+ sendPresetActivated(presetId: int): void</li> <li>+ sendPresetDeactivated(presetId: int): void</li> <li>+ sendArtDataReply(dataRequstCode: uint16_t): void</li> <li>+ sendArtCommand(message: char*, length: uint16_t): void</li> <li>+ processArtDmx(updData: uint8_t[], length: int): void</li> <li>+ processArtCommand(updData: uint8_t[], length: int): void</li> <li>+ processArtDataRequest(updData: uint8_t[], length: int): void</li> <li>+ parseToArtDmx(updData: uint8_t[], length: int): ArtDmx</li> <li>+ parseToArtPoll(updData: uint8_t[], length: int): ArtPoll</li> <li>+ parseToArtCommand(updData: uint8_t[], length: int): ArtCommand</li> <li>+ parseToArtDataRequest(updData: uint8_t[], length: int): ArtDataRequest</li> <li>+ parseFromArtCommand(artCommand: ArtCommand, udpSendData: uint8_t*, length: int): void</li> <li>+ parseFromArtPollReply(artPollReply: ArtPollReply, udpSendData: uint8_t*, length: int): void</li> <li>+ parseFromArtDataReply(artDataReply: ArtDataReply, udpSendData: uint8_t*, length: int): void</li> </ul>

**Abbildung 12** Aufbau der Klasse ArtNetController.

## 4.3 Integration des Netzwerkkommunikationsprotokolls

Der Art-Net-Controller besitzt zwei Hauptfunktionen. Zum einem lauscht dieser auf dem Port 6454 nach Art-Net-Paketen und zum anderen versendet dieser ArtCommand-Pakete, um mitzuteilen, dass ein Preset aktiviert oder deaktiviert wurde. Bei der Initialisierung des Objektes wird ein UDP-Socket gestartet, mit dem der Port 6454 abgehört werden kann. In der Methode `loop()` der Hauptapplikation wird die Methode `listenToArtNet` aufgerufen, welche prüft, ob Datenpakete am Port anliegen. Der Ablauf dieser Methode wird in Abbildung 13 dargestellt. Sobald ein Paket am Port anliegt, überprüft die Methode, ob das Paket mindestens eine Länge von 14 Bytes besitzt. Dies ist die minimale Größe eines Art-Net-Pakets (vgl. [Lic23, Seite. 17 f.]). Ist dieses Kriterium erfüllt, wird das Paket in ein Byte-Puffer-Array geladen und die IP-Adresse des Senders wird gespeichert um eine anschließende Antwort versenden zu können. Anschließend überprüft die Methode `checkArtNetHeader`, ob das Paket den Art-Net-Header enthält. Entsprechen

die ersten acht Bytes des Pakets dem Byte-Array {'A', 'r', 't', '-', 'N', 'e', 't', 0x00}, so ist dieses ein gültiges Art-Net-Paket.



**Abbildung 13** Aufbaudiagramm der Methode `listenToArtNet`.

Ist die Prüfung erfolgreich, wird anhand des Op-Codes des Art-Net-Pakets entschieden, um welchen Pakettypen es sich handelt. Der Op-Code befindet sich in den Bytes 8 und 9 des Pakets und wird mit Hilfe einer Switch-Anweisung ausgewertet. Handelt

es sich beim OpCode um den Code OpPoll(0x2000), so wird als Antwort darauf ein ArtPollReply-Paket gesendet. Dies übernimmt die Methode `sendArtPollReply`, welche in Quellcode 2 abgebildet ist.

```
1  int packageSize = 236;
2  ArtPollReply artPollReply;
3  // set controller ip
4  for(int i = 0; i < 4; i++)
5  {
6      artPollReply.IpAddress[i] = this->nodeIpAddress[i];
7  }
8  char* longName = "KIWI(is)a_Illumination_Weeny_
   ↳ Interface";
9  [...]
10 // set mac address
11 for(int i = 0; i < 6; i++)
12 {
13     artPollReply.Mac[i] = this->mac[i];
14 }
15 uint8_t updSendData[packageSize];
16 this->parseFromArtPollReply(artPollReply, updSendData,
   ↳ packageSize);
17 IPAddress remoteIp = IPAddress(this->controllerIpAddress);
18 this->udp.beginPacket(remoteIp, UDP_PORT_ADRESS);
19 this->udp.write(updSendData, packageSize);
20 this->udp.endPacket();
```

**Quellcode 2** Auszug aus der Methode `sendArtPollReply`.

In Zeile 2 des Quellcodes 2 wird eine neue Struktur vom Typen `ArtPollReply` erstellt, welche in der Datei `ArtNetPackageTypes.h` definiert ist. Alle Felder in der Struktur besitzen bereits einen Standardwert, der nun mit den korrekten Werten überschrieben werden muss. Die For-Schleife in Zeile 4 bis 7 setzt z. B. die IP-Adresse des Controllers, von dem das ArtPoll-Paket gekommen ist. Sind alle Werte im Paket gesetzt, wird dieses mit Hilfe der Methode `parseFromArtPollReply` zu einem `uint8_t`-Array umgewandelt (siehe Zeile 16). Zu jedem Pakettypen, der versendet wird, existiert eine `parseFrom`-Methode, die eine Struktur des Pakettypen erhält und diese entsprechend der Paketdefinition in ein `uint8_t`-Array umwandelt. Manche Pakettypen besitzen Felder, die nicht zwingend mitübertragen werden müssen. Damit die Methode unterscheiden kann, welche Felder mitgesendet werden sollen, erhält sie die gewünschte Anzahl der Bytes, die das Paket lang ist. Mit dieser Anzahl und der Definition des Pakettypen kann ermittelt werden, welche Felder übertragen werden und welche nicht. Quellcode 3 stellt eine gekürzte Form der Methode `parseFromArtPollReply` dar.

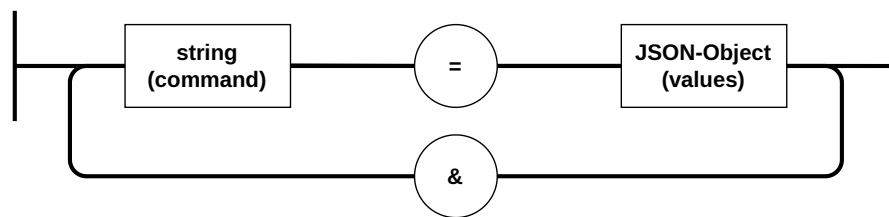
```
1  if(length >= 207){
2      // id
3      for(int i = 0; i<8; i++) {
4          udpSendData[i] = artPollReply.Id[i];
5      }
6      // op code
7      udpSendData[8] = artPollReply.OpCode;
8      udpSendData[9] = artPollReply.OpCode >> 8;
9      // node ip
10     for(int i= 0; i < 4; i++) {
11         udpSendData[10+i] = artPollReply.IpAddress[i];
12     }
13     [...]
14     // net address
15     udpSendData[18] = artPollReply.NetSwitch;
16     // sub net address
17     udpSendData[19] = artPollReply.SubSwitch;
18     [...]
19 }
20 if(length >= 208){
21     // bind ip
22     for (int i = 0; i < 4; i++) {
23         udpSendData[207+i] = artPollReply.BindIp[i];
24     }
25 }
26 if(length >= 212){[...] }
27 [...]
```

**Quellcode 3** Auszug aus der Methode parseFromArtPollReply.

Abhängig von der an die Methode übergebenen Länge des Pakets, werden weitere Felder des Pakets zum Byte-Array hinzugefügt (vgl. Zeile 1, 20 und 26 des Quellcodes 3). Für die Umwandlung von eingehenden Art-Net-Paketen in die entsprechende Struktur werden die `parseTo`-Methoden verwendet. Diese sind im Kern identisch zu den `parseFrom`-Methoden aufgebaut, mit dem Unterschied, dass nun vom Byte-Array in die Struktur geschrieben wird.

Das komplexeste Paket für die Auswertung und zum Senden, ist das Paket `ArtCommand`. Dieses enthält einen String, welcher aus einem Befehlswort (`command`) und den Werten des Befehls (`values`) besteht. Die Werte werden durch ein JSON-Objekt dargestellt. Ein `ArtCommand` kann auch mehrere Befehle übertragen, welche durch das `&`-Zeichen separiert werden. Abbildung 14 zeigt ein Syntax-Diagramm eines `ArtCommands`.

ArtCommand



**Abbildung 14** Syntax-Diagramm des Befehlsstrings eines ArtCommands.

Quellcode 4 zeigt die Verarbeitung eines eingehenden ArtCommands. Der Befehlsstring wird zuerst an allen Stellen geteilt, an denen &-Zeichen steht. Dies wird mit Hilfe des Befehls `strtok` ermöglicht. In Zeile 3 wird der Befehlsstring geladen und der Teilstring zurückgegeben. Sofern dieser nicht gleich `NULL` ist, wird er innerhalb der While-Schleife weiter verarbeitet. In Zeile 37 wird der nächste Teilstring geladen und die Schleife beginnt von vorne. Wurde ein Teilstring erfolgreich geladen, wird er im Anschluss in die beiden Bestandteile Befehl und Werte aufgeteilt (vgl. Zeile 5 bis 16). Die Bibliothek `ArduinoJson` von Benoît Blanchon wandelt mit dem Befehl `deserializeJson` den JSON-String in ein `StaticJsonDocument` um (siehe Zeile 17). Anschließend kann mit Hilfe der JSON-Objektnamen auf die Werte zugegriffen werden. Die Strukturen und deren Objektnamen der verschiedenen ArtCommands sind in der Datei `ArtCommands.h` enthalten.

```
1 ArtCommand artCommand;  
2 artCommand = this->parseToArtCommand(updData, length);  
3 char *commandString = strtok(artCommand.Data, "&");  
4 while (commandString != NULL) {  
5     String command = String(commandString)  
6         .substring(0, String(commandString)  
7             .indexOf('='));  
8     int commandValuelength = (String(commandString)  
9         .substring(String(commandString)  
10             .indexOf('=')+1)  
11         .length() + 1;  
12     char commandValue[commandValuelength];  
13     String(commandString)  
14         .substring(String(commandString)  
15             .indexOf('=')+1)  
16         .toCharArray(commandValue, commandValuelength);  
17     deserializeJson(doc, commandValue, commandValuelength);  
18  
19     if (command.equals(ART_COM_AP)) {  
20         this->presetRepository
```

```
21         ->activatePreset(  
22             doc[activatePresetStructure().presetId]);  
23     }  
24     [...]  
25     else if(command.equals(ART_COM_SP)){  
26         this->presetRepository  
27             ->setPresetGroupId(  
28                 doc[setPresetStructure().presetId],  
29                 doc[setPresetStructure().groupId]);  
30         this->presetRepository  
31             ->setPresetChannelValue(  
32                 doc[setPresetStructure().presetId],  
33                 doc[setPresetStructure().channel],  
34                 doc[setPresetStructure().value]);  
35     }  
36     [...]  
37     commandString = strtok(NULL, "&");  
38 }
```

**Quellcode 4** Auszug aus der Methode processArtCommand.

Damit die Lichtsteuersoftware nach dem Verbindungsaufbau weiß, welche Presets bereits aktiv sind, kann sie ein ArtDataRequest-Paket senden. In diesem Paket ist ein Request-Code enthalten, der in Tabelle 11 beschrieben ist. Der Bereich von 0x8000 bis 0xFFFF ist für die herstellerspezifische Nutzung vorgesehen. Mit dem Request-Code 0x8000 fragt man den aktuellen Preset-Status an. Die Lichtsteuereinheit antwortet anschließend mit einem ArtDataReply-Paket darauf. In diesem Paket ist eine Payload enthalten, die sechs Bytes groß ist. Jedes Byte gibt hierbei den Status eines Preset an. Das Byte kann die drei Werte inaktiv(0), aktiv(1) oder teilaktiv(2) enthalten.

## 5 Aufbau der Lichtsteuersoftware

Die Lichtsteuersoftware KIWI, ist das zentrale Bedienelement des Heimilluminationssystems. Der Name KIWI steht dabei für **KIWI** (is a) **I**llumination **W**eeny **I**nterface (KIWI). Die Software steuert alle Lampen des Systems und programmiert die Presets der lokalen Lichtsteuereinheit. Das folgende Kapitel stellt die Software- und Datenstruktur des Programms dar und erläutert den Aufbau der Benutzerschnittstelle. Aufbauend auf Kapitel 3.2 stellt Kapitel 5.3 die Integration des Art-Net-Protokolls in der Software dar. Abschließend beschreibt Kapitel 5.4, wie mit Hilfe von Listnern und Bindings, das Model-View-ViewModel Prinzip umgesetzt ist.

### 5.1 Softwarearchitektur

Die Lichtsteuersoftware besteht im Groben aus den vier Haupt-Paketen `application`, `model`, `viewModel` und `view`. Diese Pakete enthalten jeweils eine Hauptklasse und weitere Pakete. Abbildung 15 zeigt in vereinfachter Form die Zusammenhänge zwischen den Paketen. Das Paket `application` besteht nur aus der Klasse `MainKiwi`. Diese Klasse enthält die Main-Methode, in welcher die jeweiligen Handlerklassen der andere Pakete erzeugt werden und die Benutzeroberfläche gestartet wird.

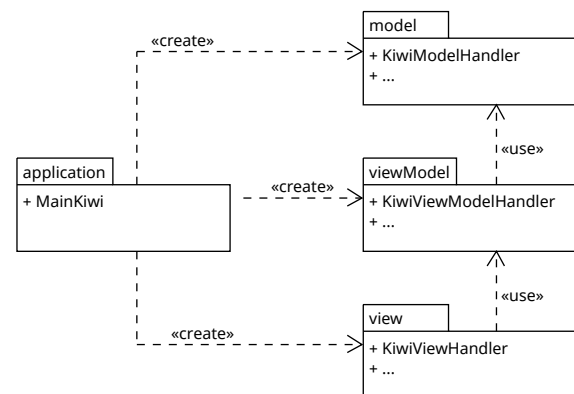


Abbildung 15 Paketdiagramm der Lichtsteuersoftware.

Die Logik der Software ist im Paket `model` enthalten, welches die Komponenten des Heimilluminationssystems abbildet. Abbildung 16 gibt eine Übersicht aller Klassen des Pakets und stellt deren Abhängigkeiten dar. Die Hauptklasse des Pakets ist die Klasse `KiwiModelHandler`. Diese erstellt die drei Repositories (Fixtures, Groups und Presets) und den Art-Net-Controller. Sie stellt Methoden zur Verfügung, durch die das ViewModel auf die Model-Objekte zugreifen kann.

Die jeweiligen Repositories verwalten Objekte wie `Fixture`, `Group` und `Preset`. Zur Speicherung und Verwaltung wird eine `Map<>` verwendet. Als Key wird das Attribut `name` verwendet, welches alle drei Objekte besitzen. Sie stellen Methoden zum Erstellen, Löschen und Bearbeiten der Objekte zur Verfügung.

Die Klasse `ArtNetController` besitzt ebenfalls eine Repository-Klasse, in der die Art-Net-Nodes verwaltet werden. Sie implementiert das Interface `LightControllable`, damit andere Objekte des Paketes auf einen Licht-Controller zugreifen können, ohne die genaue Protokollimplementierung dahinter kennen zu müssen. Des Weiteren besitzt sie ein Objekt vom Typ `UdpListener`, welches in einem eigenem Thread gestartet wird.



Die Klasse `PropertyItem` wird an alle Klassen vererbt. Sie besitzt die beiden Methoden `addPropertyChangeListener` und `removePropertyChangeListener`, mit denen die Klassen des Viewmodels Listener für Attributwerte an- und abmelden können.

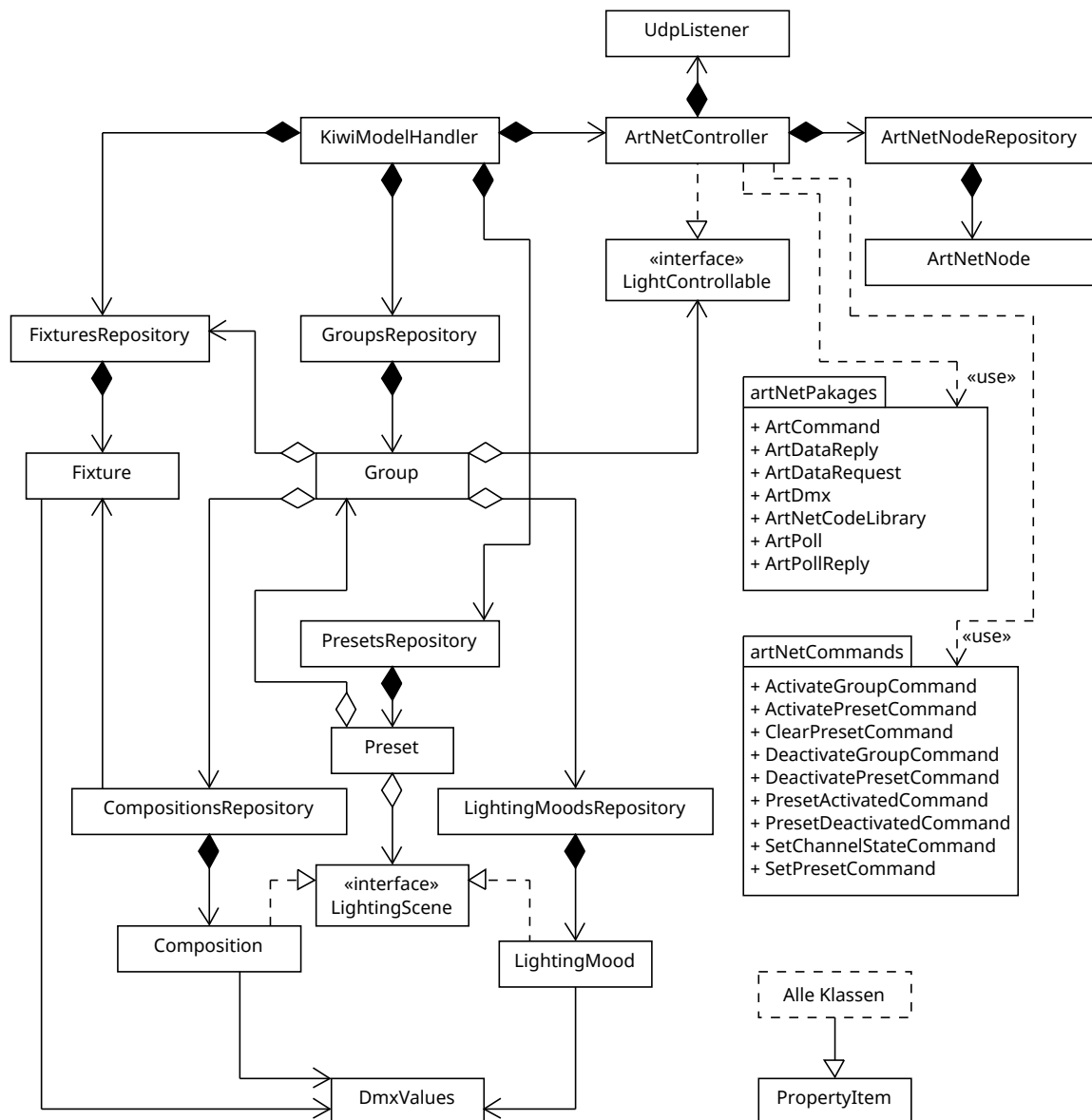


Abbildung 16 Klassendiagramm des Pakets `model`.

Die Klassen `Composition` und `LightingMood` implementieren das Interface `LightingScene`. Der Begriff Lichtszene (eng. `LightingScene`) ist ein Oberbegriff für Lichtstimmung und Komposition. Das Interface wird von der Klasse `Preset` und `Group` verwendet. Da ein `Preset` sowohl eine Lichtstimmung als auch eine Komposition enthalten kann, wird hier eine gemeinsame übergeordnete Struktur benötigt. Das Objekt der

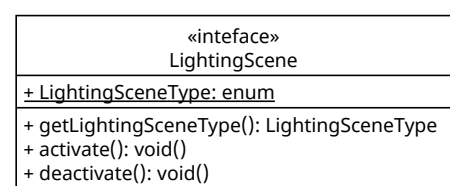


Abbildung 17 Aufbau des Interfaces `LightingScene`.

Gruppe besitzt ebenfalls ein Attribut, welches sowohl eine Lichtstimmung als auch eine Komposition sein kann. Die Methode `getLightingSceneType` ermöglicht es herauszufinden, ob es sich um eine Lichtstimmung oder eine Komposition handelt. Die Gruppe benötigt diese Information, um das Objekt im entsprechenden Repository zu suchen.

Group
<ul style="list-style-type: none"> <li>- id: byte</li> <li>- name: String</li> <li>- currentState: State = State.DEAKTIV;</li> <li>- currentLightingScene: LightingScene</li> <li>- lightController: LightControllable</li> <li>- fixtureRepository: FixtureRepository</li> <li>- lightingMoodRepository: LightingMoodRepository</li> <li>- compositionRepository: CompositionRepository</li> <li>- fixtureRepositoryListener: PropertyChangeListener</li> <li>+ State: enum</li> </ul>
<ul style="list-style-type: none"> <li>+ Group(name: String, lightController: LightControllable)</li> <li>+ getId(): byte</li> <li>+ getName(): String</li> <li>+ getCurrentState(): State</li> <li>+ getFixtureRepository(): FixtureRepository</li> <li>+ getCompositionRepository(): CompositionRepository</li> <li>+ getLightingMoodRepository(): LightingMoodRepository</li> <li>+ setId(id: byte): void</li> <li>+ setName(name: String): void</li> <li>+ setCurrentState(currentState: State): void</li> <li>+ addFixture(fixture: Fixture): void</li> <li>+ addComposition(composition: Composition): void</li> <li>+ addLightingMood(lightingMood: LightingMood): void</li> <li>+ removeFixture(fixture: Fixture): void</li> <li>+ removeComposition(composition: Composition): void</li> <li>+ removeLightingMood(lightingMood: LightingMood): void</li> <li>+ activateComposition(composition: Composition): void</li> <li>+ activateLightingMood(lightingMood: LightingMood): void</li> <li>+ deactivateComposition(composition: Composition): void</li> <li>+ deactivateLightingMood(lightingMood: LightingMood): void</li> <li>+ void activate(): void</li> <li>+ void deactivate(): void</li> </ul>

Das zentrale Element des Pakets ist die Klasse **Group**. Eine Übersicht der Methoden und Attribute der Klasse ist in Abbildung 18 zu sehen. Sie besitzt ein Fixture-, LightingMood- und Composition-Repository. Das Fixture-Repository beinhaltet alle Lampen der Gruppe und ist unabhängig vom globalen Fixture-Repository. Das Attribut `currentLightingScene` vom Typ `LightingScene` enthält die aktuell aktive Lichtstimmung oder Komposition. Da in einer Gruppe immer nur eine Lichtstimmung oder eine Komposition aktiv sein kann, muss der Gruppe beim Aktivieren einer weiteren Lichtstimmung oder Komposition bekannt sein, welche Lichtszene gerade aktiv ist, um diese zu deaktivieren.

**Abbildung 18** Aufbau der Klasse **Group**.

Die Klasse **Group** stellt Methoden zur Verfügung, mit denen Lampen hinzugefügt oder auch entfernt werden können. Das Hinzufügen und Entfernen einer Lampe findet nicht direkt über das Fixture-Repository der Gruppe statt, sondern über eigene Funktionen der Gruppe. Wird eine Lampe einer Gruppe hinzugefügt oder gelöscht, so hat dies auch Auswirkungen auf die Repositories der Lichtstimmung und Komposition.

Der Quellcode 5 zeigt die Methode `addFixture`, der Klasse **Group**. In Zeile 1 wird die Lampe dem Fixture-Repository hinzugefügt. In Zeile 2 bis 9 wird die neue Lampe allen vorhandenen Kompositionen der Gruppe hinzugefügt. Hierbei werden alle DMX-Werte der Lampe auf 0 gesetzt. Falls aktuell eine Lichtstimmung aktiv ist, werden der Lampe die DMX-Werte der Lichtstimmung zugewiesen (vgl. Zeile 11 bis 14). In Zeile 22 wird anschließend noch überprüft, ob die Gruppe aktuell aktiv ist und falls ja wird die Lampe ebenfalls auf aktiv gesetzt.

```

1  this.fixtureRepository.addFixture(fixture);
2  this.compositionRepository
3      .getCompositionMap()

```

```

4      .forEach((String name, Composition composition) -> {
5          composition
6          .setDmxValuesForFixture(
7              fixture.getName(),
8              new DmxValues());
9      });
10     if(this.currentLightingScene != null) {
11         if(this.currentLightingScene.getLightingSceneType() ==
12             ⇨ LightingSceneType.LIGHTING_MOOD) {
13             fixture.setDmxValues(
14                 ((LightingMood) this.currentLightingScene)
15                 .getDmxValues());
16         } else if(
17             this.currentLightingScene.getLightingSceneType() ==
18             ⇨ LightingSceneType.COMPOSITION) {
19             fixture.setDmxValues(
20                 ((Composition) this.currentLightingScene)
21                 .getDmxValuesForFixture(fixture.getName()));
22         }
23     }
24     if(this.currentState == State.AKTIV) {
25         fixture.activate();
26     }

```

**Quellcode 5** Methode addFixture der Klasse Group.

Die Klasse PresetRepository mit den darin enthaltenen Presets, ist eine Abbildung der Presets der Lichtsteuereinheit. Ein Preset enthält für eine Gruppe eine Lichtszene. Wird dieser aktiviert, wird die entsprechende Gruppe mit der dazugehörigen Lichtszene aktiviert. Abbildung 19 zeigt den Aufbau der Klasse Preset. Die Methode updatePreset überträgt die vom Benutzer eingestellte Lichtszene an die Lichtsteuereinheit. Quellcode 6 zeigt den Inhalt der Methode.

Preset
<pre> - numberOfPresets: int = 0 - id: int - group: Group - lightingScene: LightingScene - state: State = State.DEAKTIV - lightController: LightControllable = KiwiModelHandler.getLightController() + State: enum + Preset() + Preset(group: Group, lightingScene: LightingScene) + getId(): int + getState(): State + getGroup(): Group + getLightingScene(): LightingScene + setGroup(Group group): void - setState(State state): void + setLightingScene(lightingScene: LightingScene): void + setDmxValueForPreset(int address, DmxValues dmxValues): void + activate(): void + deactivate(): void + updatePreset(): void </pre>

**Abbildung 19** Aufbau der Klasse Preset aus der Lichtsteuersoftware.

Zu Beginn der Methode wird der Preset der lokalen Lichtsteuereinheit geleert, damit keine alten Werte mehr enthalten sind (vgl. Zeile 2). Um den Preset nun mit neuen Werten zu befüllen, muss unterscheiden werden, ob es sich bei der Lichtszene um eine

Lichtstimmung oder eine Komposition handelt (vgl. Zeile 3). Bei einer Lichtstimmung werden für jede Lampe der Gruppe die Werte der ausgewählte Lichtstimmung in den Preset der Lichtsteuereinheit übertragen (vgl. Zeile 5 bis 11). Handelt es sich um eine Komposition, so werden die in der Komposition für die Lampen hinterlegten Werte an den Preset gesendet (vgl. Zeile 15 bis 21).

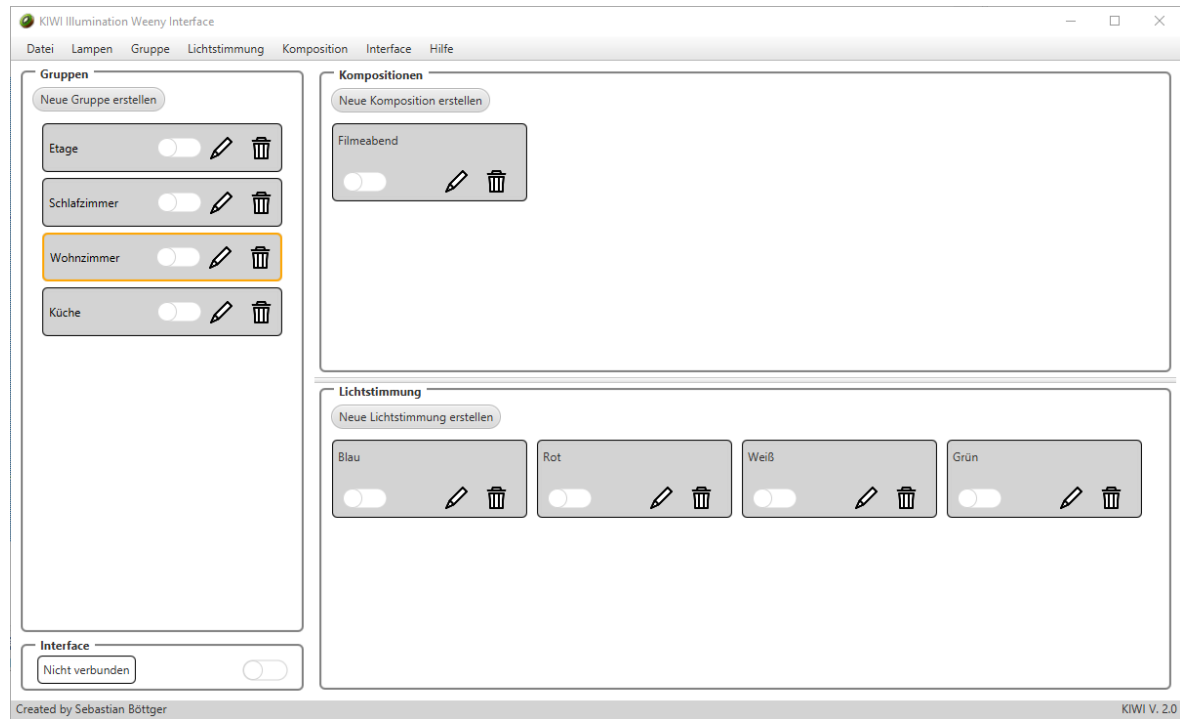
```
1  if(lightningScene != null) {
2      this.lightController.clearPreset(this.id);
3      if(lightningScene.getLightingSceneType() ==
4          LightingSceneType.LIGHTING_MOOD) {
5          group.getFixtureRepository()
6              .getFixtureMap()
7              .forEach((String name, Fixture fixture) -> {
8                  this.setDmxValueForPreset(
9                      fixture.getAddress(),
10                     ((LightingMood)lightningScene)
11                     .getDmxValues());
12              });
13      } else if (lightningScene.getLightingSceneType() ==
14          LightingSceneType.COMPOSITION) {
15          group.getFixtureRepository()
16              .getFixtureMap()
17              .forEach((String name, Fixture fixture) -> {
18                  this.setDmxValueForPreset(
19                      fixture.getAddress(),
20                     ((Composition)lightningScene)
21                     .getDmxValuesForFixture(name));
22              });
23      }
24  }
```

**Quellcode 6** Methode updatePreset der Klasse Preset.

## 5.2 Benutzerschnittstelle

Im Software-Entwicklungsprojekt wurde eine Nutzungsweise für ein Heimilluminationssystem ermittelt. Die Benutzerschnittstelle ist auf Grundlage dieser Nutzungsweise entwickelt. Abbildung 20 zeigt das Hauptfenster der Lichtsteuersoftware. Entsprechend der ermittelten Nutzungsweise bildet die Gruppe das wichtigste Objekt im Kontext eines Heimilluminationssystems. Die Liste aller Gruppen ist im linken Teil des Hauptfensters der Software lokalisiert. Durch die Auswahl einer Gruppe werden weitere Elemente im Fenster geladen. Somit besitzt die Liste der Gruppen eine Art Navigationscharakter.

Dies wird durch die Positionierung am linken Rand des Fensters unterstützt. Der Benutzer wird in der Bedienung der Software unterstützt, da die Bedienabfolge (Gruppe auswählen → Lichtszene auswählen) der gewohnte Leserichtung (von Links nach Rechts) entspricht.<sup>7</sup>



**Abbildung 20** Hauptfenster der Lichtsteuersoftware KIWI.

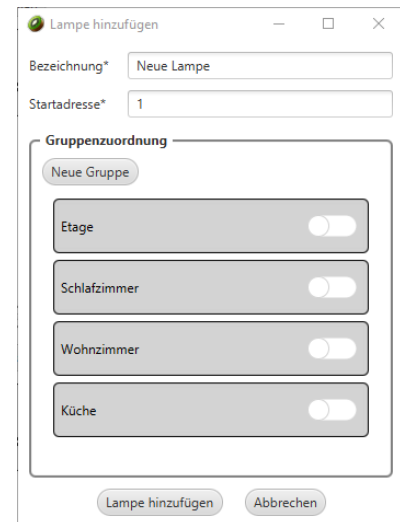
Unterhalb der Gruppenliste befindet sich der Bereich für die Lichtsteuereinheit. Diesem Bereich kann der Benutzer die Information über den Verbindungsstatus zur Lichtsteuereinheit entnehmen und diesen mit Hilfe des Schalters auch ändern.

Auf der rechten Seite des Fensters befinden sich beiden Bereiche für die Auswahl der Lichtstimmungen und der Kompositionen. Die Listenelemente sind ähnlich zu denen aus der Gruppenliste aufgebaut. Dies sorgt dafür, dass der Benutzer ein einheitliches Nutzungsgefühl erhält und gelerntes Nutzungsverhalten auch auf andere Bereiche der Software übertragen kann. So befinden sich z. B. auf den Listenelementen der Gruppenliste Icons mit einem Bleistift und einem Mülleimer. Klickt der Benutzer diese Icons an, kann er die ausgewählte Gruppe bearbeiten oder löschen. Diese Icons befinden sich auch auf den Listenelementen der anderen Listen. Der Benutzer kann nun die korrekte Funktion dieser Icons vermuten, ohne diese vorher betätigt zu haben, was die Bedienung der Software vereinfacht.

Am oberen Fensterrand befindet sich die Menübar, über die der Benutzer alle Funktionen der Software erreichen kann.

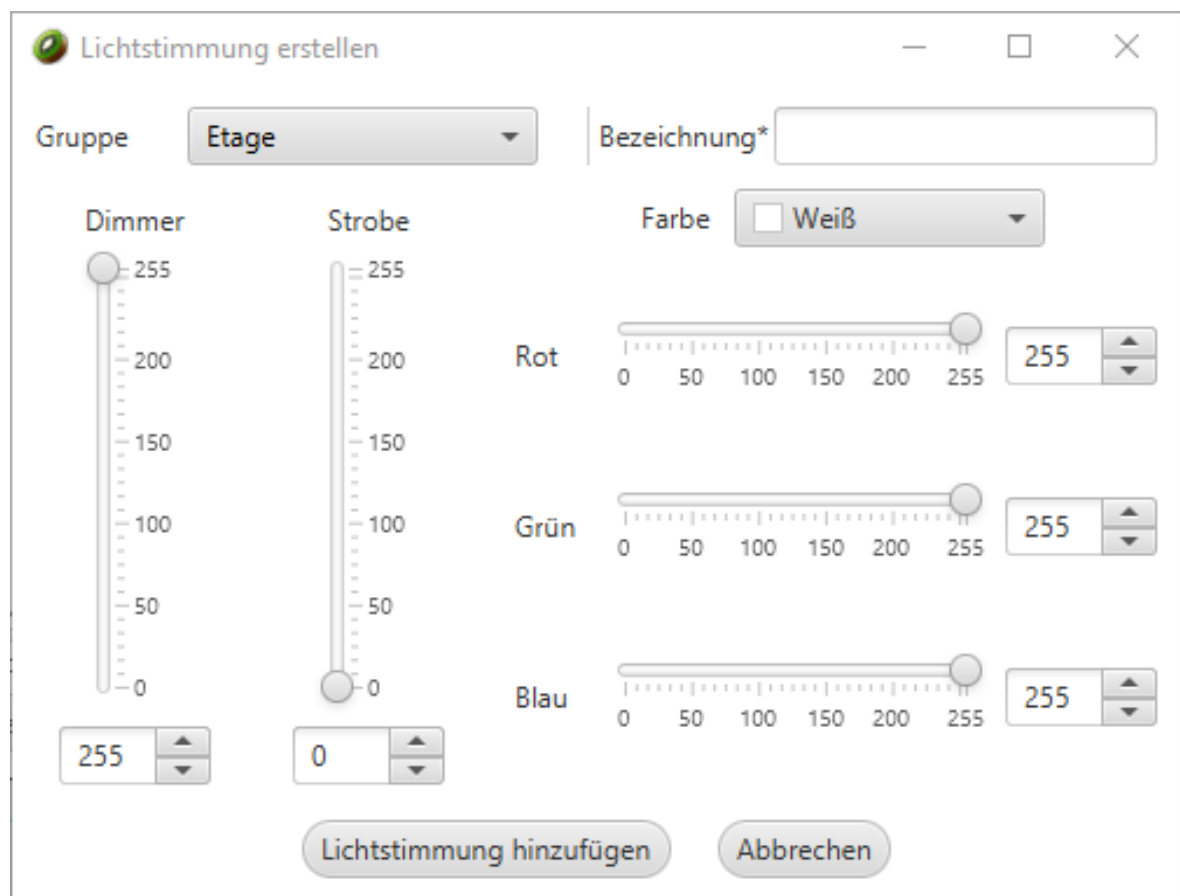
<sup>7</sup>Entspricht die gewohnte Leserichtung nicht von Links nach Rechts, so müsste die Anordnung entsprechend angepasst werden.

Entsprechend des Nutzungskontextes fügt der Benutzer nur zu Beginn und anschließend sehr selten und unregelmäßig Lampen dem System hinzu. Somit muss diese Funktionalität nicht direkt aus dem Hauptbildschirm erreichbar sein und kann in die Menübar verlagert werden. Abbildung 21 zeigt das Fenster zur Erstellung einer neuen Lampe im System. Dieses kann der Benutzer über das Menü Lampen und den Menüeintrag Lampe hinzufügen erreichen. Damit der Benutzer beim Hinzufügen einer Lampe nicht das Fenster wechseln muss, um eine neue Gruppe für die entsprechende Lampe zu erstellen, ist der Button zur Erstellung neuer Gruppen mit im Fenster platziert.



**Abbildung 21** Fenster zur Erstellung einer neuen Lampe

Zum Erstellen einer neuen Lichtstimmung kann der Benutzer den Button im oberen Bereich der Lichtstimmungsliste nutzen, oder den Menüeintrag im Menü Lichtstimmungen aufrufen. Abbildung 22 zeigt das Fenster zur Erstellung einer Lichtstimmung.

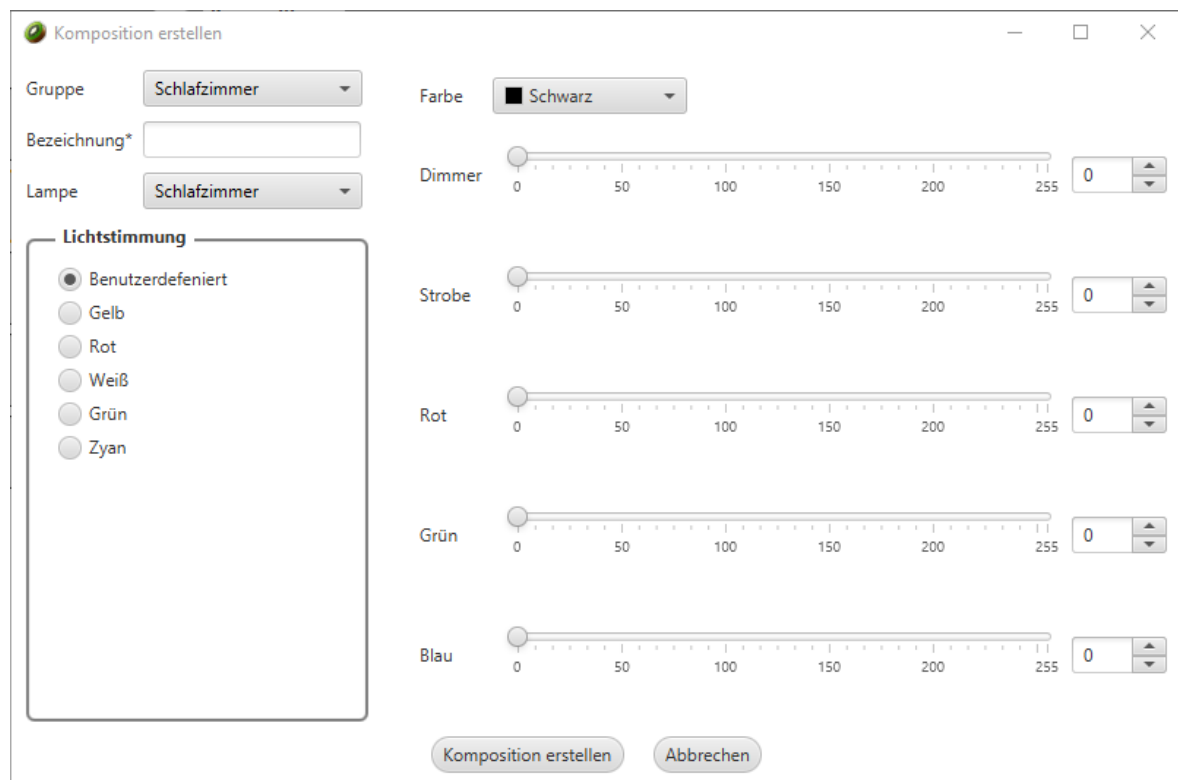


**Abbildung 22** Fenster zur Erstellung einer Lichtstimmung.

Für die Erstellung einer Lichtstimmung muss der Benutzer die Gruppe auswählen, in

der die Lichtstimmung erstellt werden soll. Nutzt der Benutzer den Button im Hauptfenster, so wird die aktuell ausgewählte Gruppe vorausgewählt. Jede Lichtstimmung benötigt einen Namen bzw. eine Bezeichnung. Diese kann der Benutzer im Textfeld oben rechts eingeben. Das Sternchen vor dem Textfeld gibt an, dass es sich hierbei um ein Pflichtfeld handelt. Lässt der Benutzer das Feld leer, wird er mit einer Fehlermeldung darauf hingewiesen. Die Fehlermeldung erscheint, sobald der Benutzer versucht, die Lichtstimmung hinzuzufügen. Mit den Schiebereglern oder den Zahlenfeldern kann der Benutzer die gewünschten Werte der Lampe einstellen. Für die Auswahl der Farbe steht diesem auch ein Farbauswahlfenster zur Verfügung, in dem er benutzerdefinierte Farben speichern kann.

Analog zur Erstellung einer Lichtstimmung kann der Benutzer eine neue Komposition über den Button der Kompositionsliste oder über den entsprechenden Menüeintrag erstellen. Abbildung 23 zeigt das Fenster zur Erstellung einer Komposition.



**Abbildung 23** Fenster zur Erstellung einer Komposition.

Für die Komposition muss ebenfalls eine Gruppe ausgewählt und ein Name vergeben werden. Bei der Komposition wählt der Benutzer die einzelnen Lampen aus und weist diesen eine Lichtstimmung zu. Dazu kann er entweder aus den vorhandenen Lichtstimmungen der Gruppe auswählen oder eine benutzerdefinierte Lichtstimmung für diese Komposition einstellen. Wechselt der Benutzer zwischen den verschiedenen Lampen, werden die eingestellten Lichtstimmungen für die entsprechende Lampe zwischengespeichert.

## 5.3 Integration des Netzwerkkommunikationsprotokolls

Die Klasse `ArtNetController` steuert die Art-Net-Kommunikation. Sie stellt die Verbindung zu den Nodes her, versendet Art-Net-Pakete wie z. B. `ArtDmx` zum Übertragen der DMX-Werte und startet den UDP-Client um Art-Net-Pakete zu empfangen. Abbildung 24 zeigt den Aufbau der Klasse `ArtNetController`.

Um sich mit den Nodes zu verbinden, sendet die Klasse ein `ArtPoll`-Paket per Broadcast. Dies wird in der Methode `connectToNodes` durchgeführt.

Als Erstes wird in der Methode der UDP-Client der Klasse `UdpListener` gestartet, damit die Antwort der Nodes empfangen werden kann. Anschließend wird ein `ArtPoll`-Paket versendet. Um ein `ArtPoll`-Paket zu versenden, wird ein Objekt vom Typ `ArtPoll` erstellt.

Für jeden Pakettypen gibt es eine entsprechende Klasse, welche die Attribute bzw. Felder des Paktes besitzt. Jede dieser Klassen besitzt auch einen Konstruktor, dem ein Byte-Array übergeben werden kann, damit dieses direkt zum entsprechenden Pakettypen geparsed wird. Zusätzlich besitzen alle Paketklassen auch die Methode `getUdpArray`, die ein Byte-Array zurück gibt, das dem Aufbau des entsprechenden Paketes entspricht. Quellcode 7 zeigt beispielhaft die Methode `getUdpArray` der Klasse `ArtDmx`.

ArtNetController
<ul style="list-style-type: none"> <li>- <code>artNetNodeRepository: ArtNetNodeRepository</code></li> <li>- <code>udpListener: UdpListener</code></li> <li>- <code>presetRepository: PresetRepository</code></li> <li>- <code>udpListenerThread: Thread</code></li> <li>- <code>dmxValues: byte[]</code></li> <li>- <code>artNodePort: int = 6454</code></li> <li>- <code>final runningUdpListener: AtomicBoolean {read only}</code></li> <li>- <code>gson: Gson</code></li> </ul>
<ul style="list-style-type: none"> <li>+ <code>ArtNetController(presetRepository: PresetRepository)</code></li> <li>+ <code>getNodeRepository(): ArtNetNodeRepository</code></li> <li>+ <code>setDmxValue(channel: int, value: byte): void</code></li> <li>+ <code>setPreset(presetId: int, groupId: byte, channel: int, value: byte): void</code></li> <li>- <code>sendArtPoll(): void</code></li> <li>- <code>sendDataRequest(): void</code></li> <li>- <code>sendArtCommand(artCommand: ArtCommand): void</code></li> <li>- <code>sendUdpForEachNode(udpSendData: byte[]): void</code></li> <li>+ <code>sendDmxValue(): void</code></li> <li>+ <code>sendDmxValue(channel: int, value: byte): void</code></li> <li>+ <code>activateGroup(groupId: byte): void</code></li> <li>+ <code>activatePreset(presetId: int): void</code></li> <li>+ <code>activateDmxChannel(channel: int): void</code></li> <li>+ <code>deactivateGroup(groupId: byte): void</code></li> <li>+ <code>deactivatePreset(presetId: int): void</code></li> <li>+ <code>deactivateDmxChannel(channel: int): void</code></li> <li>+ <code>clearPreset(presetId: int): void</code></li> <li>+ <code>connectToNodes(): void</code></li> <li>+ <code>disconnectNodes(): void</code></li> </ul>

Abbildung 24 Aufbau der Klasse `ArtNetController`

```

1 byte[] udpArray = new byte[18 + this.length];
2 byte[] opCodeArray = ByteBuffer.allocate(2)
3                                     .putShort(this.opCode)
4                                     .array();
5 byte[] lengthArray = ByteBuffer.allocate(2)
6                                     .putShort(this.length)
7                                     .array();
8 // id
9 for(int i = 0; i < 8; i++) {
10     udpArray[i] = (byte) this.id.charAt(i);
11 }

```



```

12 // op code
13 udpArray[8] = opCodeArray[1];
14 udpArray[9] = opCodeArray[0];
15 // protocoll version
16 udpArray[10] = this.protVerHi;
17 udpArray[11] = this.protVerLo;
18 // sequence
19 udpArray[12] = this.sequence;
20 // physical
21 udpArray[13] = this.physical;
22 // sub-net and universe
23 udpArray[14] = this.subUni;
24 // net
25 udpArray[15] = this.net;
26 // length
27 udpArray[16] = lengthArray[0];
28 udpArray[17] = lengthArray[1];
29 for(int i = 0; i < this.length; i++) {
30     udpArray[18 + i] = this.data[i];
31 }
32 return udpArray;
    
```

**Quellcode 7** Methode `getUdpArray` der Klasse `ArtDmx`.

Die Methode `processArtDataReply` der Klasse `UdpListener` verarbeitet im Anschluss die Antworten der Nodes. Diese prüft die IP-Adressen der eingegangenen Antworten mit denen der bekannten Nodes im Node-Repository. Stimmt diese bei einer Antwort überein, wird der Verbindungsstatus des Nodes auf „Verbunden“ gesetzt. Sind alle Nodes verbunden, so gilt die Verbindung als hergestellt.

Bei der Aktivierung von Gruppen müssen DMX-Werte mit Hilfe von Art-Net übertragen werden. Damit die Anzahl der versendeten Pakete möglichst minimal gehalten werden kann, stellt der Art-Net-Controller zwei Methoden zur Verfügung. Mit der Methode `setDmxValue` können dem Controller neue DMX-Werte mitgeteilt werden. Dieser speichert diese dann in seinem eigenem DMX-Ausgabe-Array. Erst mit dem Aufruf der Funktion `sendDmxValue` werden alle aktuell im Controller gespeicherten DMX-Werte per Art-Net versendet.

Wird ein Preset an der Lichtsteuereinheit aktiviert, informiert diese die Software mit Hilfe eines `ArtCommands`. Die Klasse `UdpListener` muss diesen empfangen und auswerten. Abbildung 25 zeigt den Aufbau der Klasse `UdpListener`. Die Klasse implementiert das Interface `Runnable` und wird von der Klasse `ArtNetController` in einem eigenem Thread ausgeführt.

Die Methode `run` enthält eine While-Schleife, die als Abbruchkriterium das Attribut `running` besitzt. Dieses Attribut ist vom Typ `AtomicBoolean` und kann somit threadsicher von außerhalb des Threads gesetzt werden. Möchte der Benutzer die aktuelle Verbindung zu den Nodes aufheben, wird der Wert des Attributs auf `false` gesetzt und die Schleife bricht ab.

Um ein Paket zu empfangen startet die `run`-Methode einen UDP-Socket mit einer Ablaufzeit, falls kein Paket kommt. Die Ablaufzeit wird benötigt, damit in regelmäßigen Abständen das Abbruchkriterium der Schleife getestet werden kann. Quellcode 8 zeigt den Aufbau der Methode `run`.

UdpListener
<ul style="list-style-type: none"> <li>- artNetController: ArtNetController</li> <li>- presetRepository: PresetRepository</li> <li>- port: int = 6454;</li> <li>- opCodeArray: byte[]</li> <li>- opCode: short</li> <li>- incomingData: byte[]</li> <li>- running: AtomicBoolean</li> <li>- gson: Gson</li> </ul>
<ul style="list-style-type: none"> <li>+ UdpListener(ArtNetController artNetController, int port, PresetRepository presetRepository, AtomicBoolean running)</li> <li>- checkIfArtNetPakage(incomingData: byte[]): boolean</li> <li>- convertIp(ip: byte[]): String</li> <li>- sendArtPollReply(reciverIP: InetAddress): void</li> <li>- processArtCommand(): void</li> <li>- processArtPollReply(): void</li> <li>- processArtDataReply(): void</li> <li>+ run(): void</li> </ul>

Abbildung 25 Aufbau der Klasse `UdpListener`

```

1  try(DatagramSocket socket = new DatagramSocket(this.port))
   ⇨ {
2      socket.setSoTimeout(1000);
3      while(this.running.get()) {
4          try {
5              DatagramPacket incomingDatagram = new
               ⇨ DatagramPacket(this.incomingData,
               ⇨ this.incomingData.length);
6              socket.receive(incomingDatagram);
7              this.incomingData = incomingDatagram.getData();
8              if(checkIfArtNetPakage(this.incomingData)) {
9                  opCodeArray[0] = this.incomingData[9];
10                 opCodeArray[1] = this.incomingData[8];
11                 this.opCode =
               ⇨ ByteBuffer.wrap(opCodeArray).getShort();
12                 [...]
13                 // if else Anweisungen entsprechend des
               ⇨ OP-Codes
14             }
15         } catch (SocketTimeoutException e) {}
16     }
17     socket.close();
18 } catch (SocketException e) {
19     e.printStackTrace(System.err);

```

```

20 } catch (IOException e) {
21     e.printStackTrace(System.err);
22 } finally {
23     artNetController.getNodeRepository()
24         .getNodesMap()
25         .forEach((String name, ArtNetNode node) -> {
26             node.setConnectStatus(false);
27         });
28 }

```

**Quellcode 8** Methode `run` der Klasse `UdpListener`.

Erhält der `UdpListener` ein `ArtCommand`-Paket, so wird die Methode `processArtCommand` aufgerufen. Die Methode zerlegt den Befehlsstring entsprechend des Syntax-Diagramms in die verschiedenen Bestandteile (vgl. Abbildung 14). Für das Parsen des JSON-Objekts wird die Bibliothek `GSON` verwendet. Quellcode 9 zeigt einen Ausschnitt der Methode `processArtCommand`.

```

1 [...]
2 PresetActivatedCommand presetActivatedCommand =
  ↳ this.gson.fromJson(commandJSON,
  ↳ PresetActivatedCommand.class);
3 presetRepository.getPreset(
  ↳ presetActivatedCommand.getPresetId()).activate();
4 [...]

```

**Quellcode 9** Methode `processArtCommand` der Klasse `UdpListener`.

Zum deserialisieren des JSON-Objektes wird der Befehl `fromJson` des `gson`-Objektes verwendet. Diesem wird der JSON-String und die Klasse mit der dazugehörigen Struktur übergeben.

## 5.4 Realisierung des Model-View-ViewModel-Entwurfsmusters

Um die Objekte aus dem Paket `model` mit der Benutzeroberfläche zu verbinden, ist das Model-View-ViewModel-Entwurfsmuster im Einsatz. Das Entwurfsmuster ist eine Alternative zum Entwurfsmuster Model-View-Controller. Die Objekte der drei Schichten werden jeweils mit Hilfe von Bindings miteinander verbunden. Anstelle von Bindings können auch Listener verwendet werden. Abbildung 26 stellt das Entwurfsmuster dar. Das ViewModel verarbeitet die Eingaben der View und reicht sie ans Model weiter. Sobald die Daten im Model verändert wurden sind, werden diese per Binding auch auf der Benutzeroberfläche angepasst. Das ViewModel besitzt ebenfalls alle Eigenschaften der Benutzeroberfläche als Attribute. Kann ein Listenelement z. B. selektiert werden, so benötigt das ViewModel einen `boolean`-Parameter, der diesen Zustand abbildet.

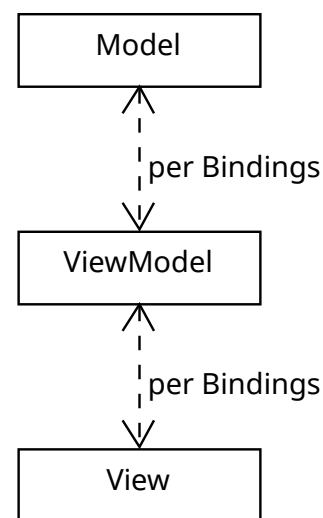


Abbildung 26 Entwurfsmuster Model-View-ViewModel

Abbildung 27 stellt grundlegend dar, wie das Entwurfsmuster in der Software realisiert wurde. Die Hauptapplikation `MainKiwi` besitzt die drei Handler-Objekte für Models, ViewModels und die View. Die Handler-Objekte besitzen jeweils eine Referenz auf den nächst niedrigeren Handler, damit diese das Binding herstellen oder einen Listener erstellen können. Die eigentlichen Objekte (`Fixture`, `Group`, `FixtureVM` etc.) der Schicht erhalten über diese Handler die entsprechenden Objekte der niedrigeren Schicht.

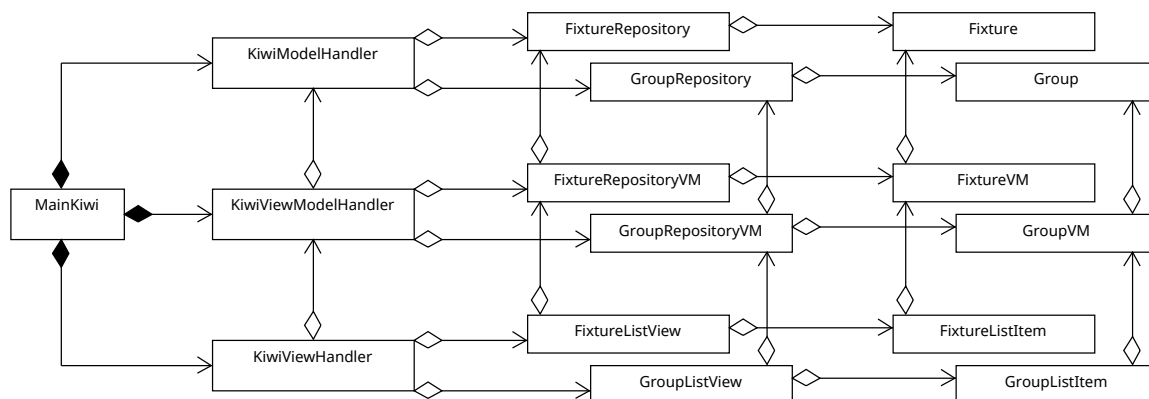


Abbildung 27 MVVM-Struktur der Applikation

Um das Model unabhängig von der Benutzeroberfläche zu halten, wird anstelle der `SimpleProperty`-Klassen der JavaFX Bibliothek die Klasse `PropertyChangeSupport` der

Java Beans Bibliothek verwendet. Die Klasse ermöglicht es anderen Klassen, sich für Änderungen anzumelden. Für das An- und Abmelden von Listnern benötigt die Model-Klasse die beiden Methoden `addPropertyChangeListener` und `removePropertyChangeListener`, die durch die Klasse `PropertyItem` an alle Model-Klassen vererbt werden. Ändern sich Werte im Model, können diese Änderungen wie in Quellcode 10 dargestellt, an die angemeldeten Listener mitgeteilt werden.

```

1 public void setName(String name) {
2     this.propertySupport.
      ↳ firePropertyChange("name", this.name, name);
3     this.name = name;
4 }

```

**Quellcode 10** Methode `setName` der Klasse `Group`.

Die Methode `firePropertyChange` teilt allen angemeldeten Listnern mit, welches Attribut sich geändert hat, was der alte und der neue Wert ist. Ein `ViewModel` kann sich dann, wie in Quellcode 11 dargestellt, an einer Klasse anmelden und auf Änderungen reagieren.

```

1 this.groupListener = new PropertyChangeListener() {
2     @Override
3     public void propertyChange(PropertyChangeEvent event) {
4         if(event.getPropertyName().equals("name")) {
5             groupName.setValue((String)event.getNewValue());
6         } [...]
7     }
8 };
9 group.addPropertyChangeListener(this.groupListener);

```

**Quellcode 11** Auszug aus dem Konstruktor der Klasse `GroupVM`.

Bei dem Attribut `groupName` handelt sich um ein `SimpleStringProperty`. Dieses kann an ein `TextProperty` eines `JavaFX-Elements` gebunden werden. Quellcode 12 zeigt das Binding des Gruppennamens an das Label des `GroupListItems`.

```

1 this.groupVM = groupVM;
2 this.groupRepositoryVM =
      ↳ viewModelHandler.getGroupRepositoryVM();
3 this.groupId.bind(this.groupVM.groupIdProperty());
4 this.groupName.bind(this.groupVM.groupNameProperty());
5 this.groupState.bindBidirectional(
      ↳ this.groupVM.groupStateProperty());

```

```

6  this.groupSelected.bind(
    ↪ this.groupVM.groupSelectedProperty());
    
```

**Quellcode 12** Auszug aus dem Konstruktor der Klasse `GroupListItem`.

Kann der Benutzer das JavaFX-Element beeinflussen, so kann diese Eigenschaft auch bidirektional verbunden werden (siehe Zeile 5). Bei einer bidirektionalen Verbindung wird der in der View eingestellte Wert auch zurück ins ViewModel übertragen. Dies macht besonders Sinn in den Eingabemasken zum Ändern von bestehenden Objekten, wie Gruppen oder Lichtszenen.

Zum Verbinden von ganzen Listen enthält das ViewModel eine `ObservableList`. Diese ermöglicht es, die Liste direkt zu verbinden oder einen Listener für Änderungen an der Liste anzumelden. Quellcode 13 zeigt, wie der `GroupMainPanelController` über Änderungen an der Liste der Gruppen benachrichtigt wird.

```

1  groupRepositoryVM.groupsListProperty()
    ↪ .addListener((ListChangeListener.Change<? extends
    ↪ GroupVM> change) -> {
2      while(change.next()) {
3          change.getAddedSubList().forEach(group -> {
4              this.groups.add(new GroupListItem(group,
                    ↪ this.viewModelHandler));
5          });
6
7          change.getRemoved().forEach(group -> {
8              int groupListIndex = -1;
9              for(int i = 0; i < this.groups.size(); i++) {
10                 if(this.groups.get(i).getGroupName()
                    ↪ .equals(group.getGroupName())) {
11                     groupListIndex = i;
12                     break;
13                 }
14             }
15             if(groupListIndex != -1) {
16                 this.groups.remove(groupListIndex);
17             }
18         });
19     }
20 });
    
```

**Quellcode 13** Auszug aus der Methode `init` der Klasse `GroupMainPanelController`.

Werden neue Objekte durch die Benutzeroberfläche erzeugt oder geändert, so reicht die View diesen Vorgang an das ViewModel weiter, das die Objekte anschließend im Model erstellt oder bearbeitet. Sind die Änderungen im Model durchgeführt, werden ViewModel und View über Listener und Bindings darüber informiert und aktualisiert. Quellcode 14 zeigt beispielhaft, die Methode `newGroup` der Klasse `GroupRepositoryVM`, welche von der View aufgerufen wird, wenn eine neue Gruppe erstellt werden soll.

```
1 public void newGroup(String name) {  
2     this.groupRepository.addGroup( new  
    ↳ Group(name, KiwiModelHandler.getLightController()));  
3 }
```

**Quellcode 14** Methode `newGroup` der Klasse `GroupRepositoryVM`.

## 6 Aktuelle Herausforderungen

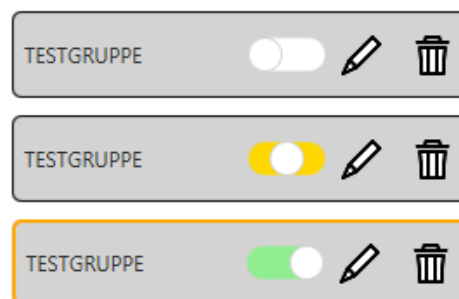
Zum Zeitpunkt der Verfassung dieser Arbeit bestehen noch zwei Probleme in der aktuellen Konfiguration des Systems. Zum einem stürzt die lokale Lichtsteuereinheit bei zu hoher Belastung durch Netzwerkpakete ab und zum anderen sind manche Animationen und Listen bei Aktualisierungen der Benutzerschnittstelle fehlerhaft.

### 6.1 Leistungsgengpass der Lichtsteuereinheit

Beim Dauerbetrieb der lokalen Lichtsteuereinheit ist aufgefallen, dass bei einer zu hohen Paketlast der Mikrocontroller abstürzt oder falsche Befehle ausführt. Nach aktuellen Erkenntnissen ist dies auf eine Art von fehlerhaften Speicherzugriff oder überlaufenden Speicher zurückzuführen. In durchgeführten Tests, bei denen alle Paket-Typen einzeln versendet wurden, konnte ein fehlerfreies Verhalten nachgewiesen werden. Alle Befehle werden korrekt erkannt und im erwarteten Zeitraum umgesetzt. Auch die Befehle und Informationen, die von der lokalen Lichtsteuereinheit versendet werden, konnten mit Hilfe von WireShark auf Korrektheit überprüft und bestätigt werden.

### 6.2 Grafikanimationen und Listenaktualisierungen

Ein weiteres Problem sind Grafikanimationen von JavaFX Elementen und die Aktualisierungen von Listen. Für die Statusanzeige einer Gruppe wird ein sogenannter ThreeStageSwitchButton verwendet. Dieser Switch-Button hat drei Zustände: AN, AUS und MITTE. Abbildung 28 zeigt die drei Zustände des Switch-Buttons. Die Position und Farbe des Switch-Buttons ist mit Hilfe eines Listeners an ein ObjectProperty des Listenelements gebunden. Ändert sich der Status der Gruppe, so wird eine Positions- und Farbanimation vom alten Zustand zum neuen durchgeführt. Quellcode 15 zeigt den Listener im Switch-Button-Objekt.



**Abbildung 28** Zustände des ThreeStageSwitchButtons

```

1 buttonState.addListener((obs, oldState, newState) -> {
2     switch (newState) {
3         case ON:
4             translateAnimation.setToX(background.getWidth() -
               ↳ background.getArcHeight());
5             fillAnimation.setFromValue(oldState == State.OFF
               ↳ ? Color.WHITE : Color.GOLD);
6             fillAnimation.setToValue(Color.LIGHTGREEN);

```



```
7         break;
8     case MID:
9         translateAnimation.setToX((background.getWidth()
10        ↪ - background.getArcHeight())/2);
11         fillAnimation.setFromValue(oldState == State.OFF
12        ↪ ? Color.WHITE : Color.LIGHTGREEN);
13         fillAnimation.setToValue(Color.GOLD);
14         break;
15     case OFF:
16         translateAnimation.setToX(0);
17         fillAnimation.setFromValue(oldState == State.ON ?
18        ↪ Color.LIGHTGREEN : Color.GOLD);
19         fillAnimation.setToValue(Color.WHITE);
20         break;
21     }
22     animation.play();
23 }));
```

**Quellcode 15** Statelister im ThreeStageSwitchButton

Wird der Status der Gruppe jedoch zu schnell geändert, wird die Animation nicht mehr korrekt durchgeführt. Wird eine Gruppe z. B. schnell hintereinander auf den Status „TEILAKTIV“ und dann auf den Status „AKTIV“ gesetzt, so bleibt der Switch-Button in der Mittleren Position stehen.

Auch bei den Aktualisierungen von Listen kommt es des Häufigeren zu Ungleichheiten zwischen der View und dem Model. Fügt man z. B. eine neue Lichtstimmung einer Gruppe hinzu, so kann es passieren, dass die neu erstellte Lichtstimmung mehrfach in der Gruppe angezeigt wird. Wechselt man zwischen den Gruppen hin und her, wird beim erneuten Laden der Lichtstimmungen die Anzahl wieder korrekt angezeigt.

## 7 Fazit und Ausblick

Zusammenfassend lässt sich das Ergebnis dieser Arbeit als erfolgreich bezeichnen. Im Rahmen der Arbeit wurde ein etabliertes und standardisiertes Netzwerkprotokoll ermittelt, mit dem Lampen in einem Heimilluminationssystem gesteuert werden können. Das ermittelte Protokoll Art-Net ist in die lokale Lichtsteuereinheit und die Lichtsteuersoftware integriert. Der Softwarekern der Lichtsteuereinheit ist mit der Benutzeroberfläche mit Hilfe des Model-View-ViewModel-Entwurfsmusters verbunden.

Beim Dauerbetrieb der Lichtsteuereinheit wurden Performance-Engpässe beobachtet, die in Kapitel 6.1 beschrieben sind. Auch sind manchen Animationen und Listenaktualisierungen in der Benutzeroberfläche noch fehlerhaft (siehe Kapitel 6.2).

Unabhängig von diesen Herausforderungen kann die Lichtsteuersoftware mit einem handelsüblichen Art-Net-Interface betrieben und zum Steuern des Heimilluminationssystems genutzt werden. Sämtlicher Quellcode zum entwickelten Heimilluminationssystem steht unter der MIT-Lizenz und ist unter dem folgenden Link öffentlich zugänglich:

<https://gitlab.cvh-server.de/sboettger/kiwi-home-light-control-system>

In der Zukunft könnten folgende Aspekte am System noch verbessert werden:

- Stabile Paketverarbeitung der Lichtsteuereinheit
- Verbesserte Grafikanimation von JavaFX-Elementen
- Korrekte Aktualisierung von Listen auf der Benutzeroberfläche
- Speichern und Laden der aktuellen Konfiguration der Lichtsteuersoftware
- Verbesserte Usability der Benutzeroberfläche
  - Duplizieren von Gruppen
  - Hinweiskfelder für zusätzliche Beschreibungen von Funktionen
  - Erstellen von Lichtszenen auf Grundlage der aktuell eingestellten Beleuchtung

# Anhang

## A Literaturverzeichnis

- [SA18] Entertainment Services und Technology Association. *Entertainment Technology Lightweight streaming protocol for transport of DMX512 using ACN*. Techn. Ber. 630 Ninth Avenue Suite 609, New York NY 10036, USA, Nov. 2018.
- [Lic23] Artistic Licence. *Art-Net 4 - Specification for the Art-Net 4 Ethernet Communication Protocol*. Techn. Ber. Bovey Tracey GB, 2023. URL: <https://www.artisticlicence.com/WebSiteMaster/UserGuides/art-net.pdf>.
- [CU23] UXQB-Arbeitsgruppe CPUX-UR. *CPUX-UR Curriculum – Certified Professional for Usability and User Experience – User Requirements Engineering*. Techn. Ber. März 2023.
- [DIN00] DIN e.V. *DIN 56930-2 - Bühnentechnik - Bühnenlichtstellsysteme - Teil 2: Steuersignale*. Techn. Ber. Frankfurt am Main, 2000-03.
- [Alla] Connectivity Standards Alliance. *Matter Specification - 23-27349-003-Matter-1.2-Core-Specification*. Techn. Ber. 508 Second Street, Suite 109B Davis, CA 95616 - USA. URL: <https://csa-iot.org/wp-content/uploads/2023/10/Matter-1.2-Core-Specification.pdf>.
- [Allb] Connectivity Standards Alliance. *Matter Device Library Specification - 23-27351-003-Matter-1.2-Device-Library-Specification*. Techn. Ber. 508 Second Street, Suite 109B Davis, CA 95616 - USA. URL: <https://csa-iot.org/wp-content/uploads/2023/10/Matter-1.2-Device-Library-Specification.pdf>.
- [Allc] ZigBee Alliance. *ZigBee Specification- ZigBee Document – 05-3474-21*. Techn. Ber. 508 Second Street, Suite 206 Davis, CA 95616 - USA. URL: <https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf>.
- [Bö23a] Sebastian Böttger. *Entwicklung einer Steuersoftware für Heimilluminationsysteme mit DMX-fähigen Endgeräten*. Techn. Ber. Hochschule Bochum, Sep. 2023.
- [Bö23b] Sebastian Böttger. *Programmierbares Hardwareinterface für ein Heimilluminationssystem*. Techn. Ber. Hochschule Bochum, Aug. 2023.
- [Ltd24] Artistic Licence Engineering Ltd. *Art-Net*. <https://art-net.org.uk/> [Aufgerufen am: 24. Februar 2024]. 2024.

## B Abbildungsverzeichnis

1	Systemarchitektur eines Heimilluminationssystems . . . . .	5
2	Beispielhafter Aufbau eines Art-Net Netzwerkes . . . . .	14
3	Beispielhafte Systemarchitektur des MA-Net3. . . . .	21
4	Aufbau der lokalen Lichtsteuereinheit schematisch dargestellt. . . . .	25
5	Foto des realen Aufbaus der Lichtsteuereinheit. . . . .	26
6	Einfaches Klassendiagramm der Lichtsteuereinheit. . . . .	27
7	Aufbau der Struktur <code>DmxChannel</code> . . . . .	27
8	Aufbau der Struktur <code>PresetStoredInformation</code> . . . . .	28
9	Aufbau der Klasse <code>Preset</code> . . . . .	28
10	Aufbau der Klasse <code>PresetRepository</code> . . . . .	29
11	Aufbau der Klasse <code>DmxController</code> . . . . .	29
12	Aufbau der Klasse <code>ArtNetController</code> . . . . .	30
13	Aufbaudiagramm der Methode <code>listenToArtNet</code> . . . . .	31
14	Syntax-Diagramm des Befehlsstrings eines <code>ArtCommands</code> . . . . .	34
15	Paketdiagramm der Lichtsteuersoftware. . . . .	36
16	Klassendiagramm des Pakets <code>model</code> . . . . .	37
17	Aufbau des Interfaces <code>LightingScene</code> . . . . .	37
18	Aufbau der Klasse <code>Group</code> . . . . .	38
19	Aufbau der Klasse <code>Preset</code> aus der Lichtsteuersoftware. . . . .	39
20	Hauptfenster der Lichtsteuersoftware KIWI. . . . .	41
21	Fenster zur Erstellung einer neuen Lampe . . . . .	42
22	Fenster zur Erstellung einer Lichtstimmung. . . . .	42
23	Fenster zur Erstellung einer Komposition. . . . .	43
24	Aufbau der Klasse <code>ArtNetController</code> . . . . .	44
25	Aufbau der Klasse <code>UdpListener</code> . . . . .	46
26	Entwurfsmuster Model-View-ViewModel . . . . .	48
27	MVVM-Struktur der Applikation . . . . .	48
28	Zustände des <code>ThreeStageSwitchButtons</code> . . . . .	52

## C Tabellenverzeichnis

1	Beispielhafte Kanalbelegung eines DMX-Universums . . . . .	7
2	DMX-Kanalbelegung der Stairville xBrick Full-Colour Fluter . . . . .	8
3	Übersicht der Benutzergruppen und deren menschenzentrierte Qualitätsziele	10
4	Definition der Begriffe Node und Controller. Sinnhaft übersetzt aus der Spezifikation (vgl. [Lic23, Seite. 9 f.]) . . . . .	13
5	Aufbau der Port-Address . . . . .	14
6	Gekürzter Aufbau des ArtPoll-Pakets (vgl. [Lic23, Seite. 17 f.]). . . . .	15

7	Ein Auszug der OpCodes des Art-Net-Protokolls (vgl. [Lic23, Seite. 19-21]).	15
8	Gekürzter Aufbau des ArtPollReply-Pakets (vgl. [Lic23, Seite. 24-32]).	16
9	Gekürzter Aufbau des ArtDmx-Pakets (vgl. [Lic23, Seite. 60 f.]).	17
10	Gekürzter Aufbau des ArtDataRequest-Pakets (vgl. [Lic23, Seite. 44 f.]).	18
11	Codes zum Anfragen von Daten (vgl. [Lic23, Seite. 45]).	18
12	Gekürzter Aufbau des ArtDataReply-Pakets (vgl. [Lic23, Seite. 46 f.]).	19
13	Gekürzter Aufbau des ArtCommand-Pakets (vgl. [Lic23, Seite. 52 f.]).	19
14	Gekürzter Aufbau des Data Packet (vgl. [SA18, Seite. 6 f.]).	20
15	Nutzwertanalyse der drei Protokolle Art-Net, sACN und MA-Net3.	22

## D Abkürzungsverzeichnis

<b>ACN</b>	Architecture for Control Networks	20
<b>App</b>	Applikation	1
<b>CSA</b>	Connectivity Standards Alliance	6
<b>DMX</b>	Digital Multiplex	1
<b>EEPROM</b>	Electrically Erasable Programmable Read-Only Memory	24
<b>ESTA</b>	Entertainment Services and Technology Association	20
<b>IFA</b>	Internationale Funkausstellung	1
<b>KIWI</b>	KIWI (is a) Illumination Weeny Interface	36
<b>LAN</b>	Local Area Network	8
<b>sACN</b>	streaming Architecture for Control Networks	20
<b>UDP</b>	User Datagram Protocol	13
<b>USB</b>	Universal Serial Bus	1
<b>WLAN</b>	Wireless Local Area Network	9

## E Quellcodeverzeichnis

1	Auszug aus der Konfigurationsdatei config.h.	25
2	Auszug aus der Methode sendArtPollReply.	32
3	Auszug aus der Methode parseFromArtPollReply.	33
4	Auszug aus der Methode processArtCommand.	34
5	Methode addFixture der Klasse Group.	38
6	Methode updatePreset der Klasse Preset.	40
7	Methode getUdpArray der Klasse ArtDmx.	44
8	Methode run der Klasse UdpListener.	46
9	Methode processArtCommand der Klasse UdpListener.	47
10	Methode setName der Klasse Group.	49

11	Auszug aus dem Konstruktor der Klasse <code>GroupVM</code> . . . . .	49
12	Auszug aus dem Konstruktor der Klasse <code>GroupListItem</code> . . . . .	49
13	Auszug aus der Methode <code>init</code> der Klasse <code>GroupMainPanelController</code> . .	50
14	Methode <code>newGroup</code> der Klasse <code>GroupRepositoryVM</code> . . . . .	51
15	Statelister im <code>ThreeStageSwitchButton</code> . . . . .	52

# Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit selbständig verfasst und keinen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Die Regelungen der geltenden Prüfungsordnung zu Versäumnis, Rücktritt, Täuschung und Ordnungsverstoß habe ich zur Kenntnis genommen.

Diese Arbeit hat in gleicher oder ähnlicher Form keiner Prüfungsbehörde vorgelegen.

Heiligenhaus, den \_\_\_\_\_

\_\_\_\_\_  
Unterschrift