Designdokument

Projekt

WakeUp-Light

Gruppe

- Andreas Züger
- Markus Schenk
- Endre Marczi

Vorwort

Das nachfolgende Designdokument soll die Anforderungen an das WakeUp-Light, die in der Analyse definiert wurden, in umsetzbare Spezifikationen manifestieren. Dazu werden die bereits gefundenen Use Cases ausgebaut und mit Details angereichert, es werden erste Klassendiagramme eingeführt, das Datenmodell und der WebService spezifiziert und das Schaltbild wird zum ersten Mal präsentiert.

Projektmanagement

Um das Projekt WakeUp-Light besser zu koordinieren wurde das Vorhaben in fünf Phasen gegliedert.

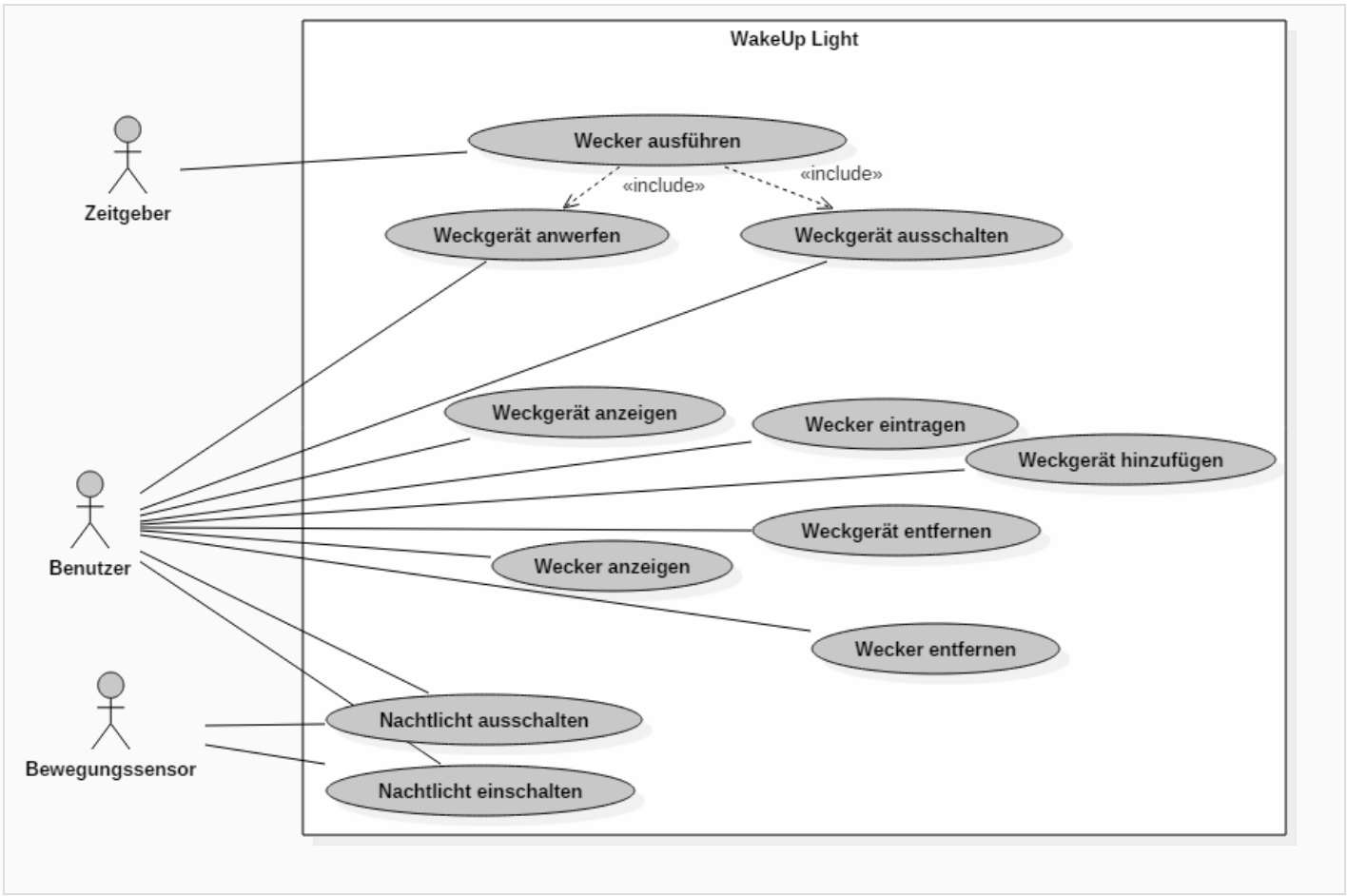
1. Analyse
2. Design
3. Implementierung
4. Test
5. Abgabe und Präsentation

Zu jeder Phase wurden terminierte und beschriebene Work Items erstellt. Jedes Work Item stellt eine unabhängig, abschliessbare Arbeitseinheit ein. Die Projektteilnehmer können sich für Work Items selbstständig eintragen und sind dann dafür verantwortlich, sie bis zum Endtermin abzuliefern. Zurzeit besteht das Projekt aus 49 Work Items die bis zum Abschluss der Phase 3 reichen.

9	**P00 - Projektidee**	**Abgabe Projektidee**		**Markus Schenk**	**15.10.2016**	**1**	**1**	**WI09_Abgabe.Projektidee**
0	P01 - Analyse	Anforderungen beschreiben		Markus Schenk		1	1	WI10_Anforderungen.beschreiben
1	P01 - Analyse	UseCases beschreiben		Markus Schenk		1	1	WI11_UseCases.beschreiben
2	P01 - Analyse	Kontextdiagramm erstellen		Andreas Züger		1	1	WI12_Kontextdiagramm.erstellen
9	**P01 - Analyse**	**Abgabe Analyse**		**Markus Schenk**	**05.11.2016**	**1**	**1**	**WI19_Abgabe.Analyse**
0	P02 - Design	Klassendiagramm Treiberlayer	Für den Treiberlayer soll ein Klassendiagramm ers	Markus Schenk		2	2	WI20_Klassendiagramm.Treiberlayer
1	P02 - Design	Klassendiagramm Middleware	Für den Middlewarelayer (WebService) soll ein Kla	Markus Schenk		2		WI21_Klassendiagramm.Middleware
2	P02 - Design	Datenmodell	Für die persistente Datenhaltung soll ein Datenmo	Markus Schenk		1	1	WI22_Datenmodell
3	P02 - Design	Sequenzdiagramm Treiberlayer	Erstellen eines Sequenzdiagramms anhand des Kl			2		WI23_Sequenzdiagramm.Treiberlayer
4	P02 - Design	Sequenzdiagramm Middleware	Erstellen eines Sequenzdiagramms anhand des Kl			2		WI24_Sequenzdiagramm.Middleware
5	P02 - Design	Erstellen Testplan	Ausdenken von Testfällen anhand des bestehender	Andreas Züger		4	2	WI25_Erstellen.Testplan
6	P02 - Design	Erstellen Schaltungsentwurf	Erstellen eines schriftlichen Schaltungsentwurf	Andreas Züger		2	2	WI26_Erstellen.Schaltungsentwurf
7	P02 - Design	Webservice Definition	Beschreiben des WebServices mit allen Schnittste	Markus Schenk		4		WI27_Webservice.Definition
9	**P02 - Design**	**Abgabe Design**	**Zusammenfügen der Designdokumente und pi**	**Markus Schenk**	**19.11.2016**	**1**		**WI29_Abgabe.Design**
0	P03 - Implementierung	Coding TL Daemon	Erstellen des Linux daemons anhand des Klassen			8		WI30_Coding.TL.Daemon
1	P03 - Implementierung	Coding TLHardwareconnector	Erstellen der Treiberfactory, den spezifischen Treib			8		WI31_Coding.TLHardwareconnector
2	P03 - Implementierung	Coding TL Datenbankconnector	Erstellen des Datenbankconnectors in Python			8		WI32_Coding.TL.Datenbankconnector
3	P03 - Implementierung	Coding MW Datenbankconnector	Erstellen der Datenbankanbindung im Middlewarela			8		WI33_Coding.MW.Datenbankconnector
4	P03 - Implementierung	Coding MW WebService	Erstellen des WebServices anhand der WebServic			8		WI34_Coding.MW.WebService
5	P03 - Implementierung	Aufbau und Dokumentation Schaltung				8		WI35_Aufbau.und.Dokumentation.Schaltung
6	P03 - Implementierung	Installation Webserver	Inklusive Dokumentation			3		WI36_Installation.Webserver
7	P03 - Implementierung	Installation Datenbank	Inklusive Dokumentation			3		WI37_Installation.Datenbank
9	P03 - Implementierung	Coding Datenbankscripts	Erstellen von Datenbankscripts, die automatisch d			4		WI39_Coding.Datenbankscripts
3	P03 - Implementierung	Coding GUI Design	Erstellen eines kleinen GUI Designs und der nötige			4		WI38_Coding.GUI.Design
9	P03 - Implementierung	Coding GUI WebService Anbindung	Erstellen der WebService Anbindung			8		WI39_Coding.GUI.WebService.Anbindung
0	P03 - Implementierung	Coding GUI Model	Zusatzzeit für unvorhergesehenes im GUI			8		WI40_Coding.GUI.Model
1	P03 - Implementierung	Zusammenfügen Code	Zusammenfügen all der Elemente aus der Impleme			8		WI41_Zusammenfügen.Code
3	P03 - Implementierung	Dokumentation zusammenführen	Zusammenführen der Dokumentation			8		WI48_Dokumentation.zusammenführen
9	**P03 - Implementierung**	**Funktionierendes System**			**19.12.2016**			**WI49_Funktionierendes.System**
0	P04 - Test							WI50_

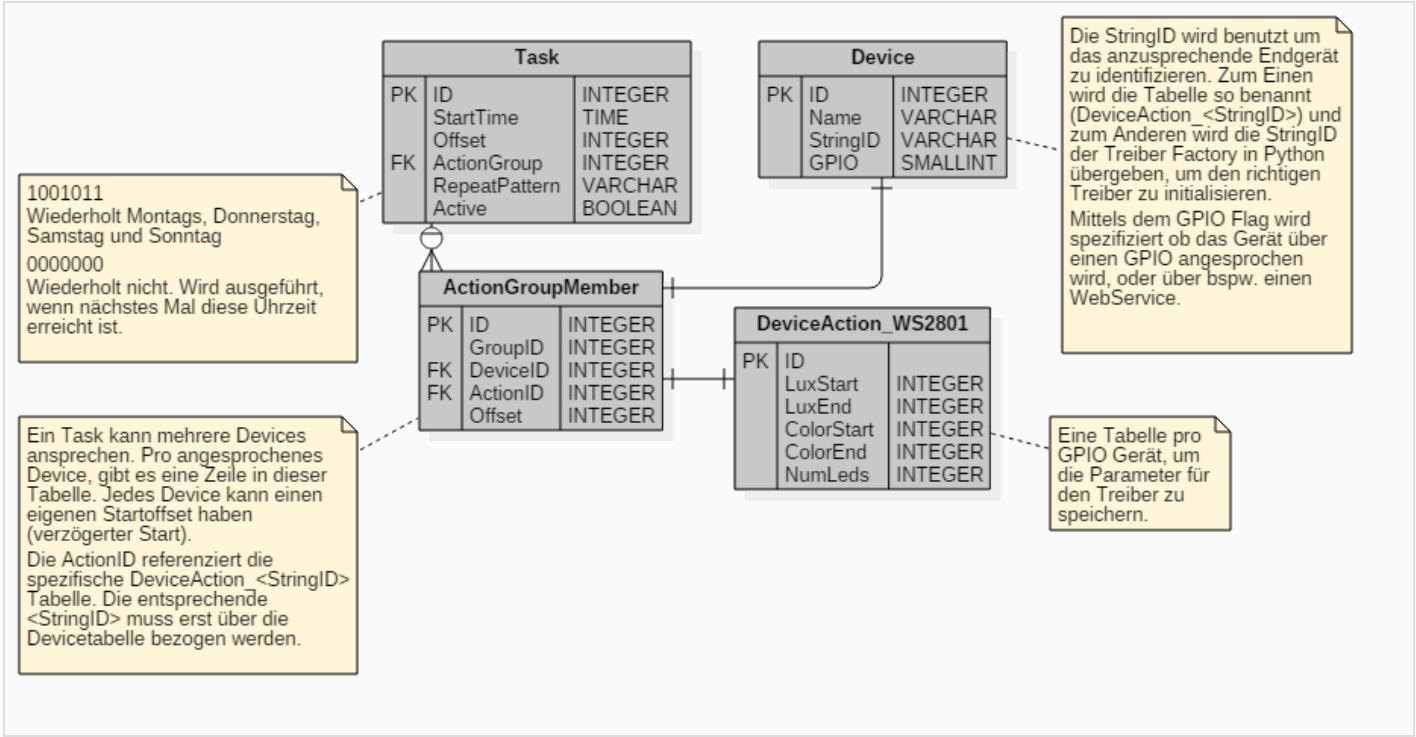
Use Case Diagramme

Beim entwerfen der Klassendiagramme wurde auf den bestehenden Use Cases aus der Analysephase aufgebaut. Die Use Cases wurden wo sinnvoll erweitert, umbenannt oder ergänzt um möglichst stimmig für den Endbenutzer und die Entwickler zu sein.



Datenmodell

Das Datenmodell stellt die persistente Datenhaltung in der Datenbank dar. Der Treiberlayer zieht Aufträge aus der Datenbank und der Middlewarelayer schreibt Aufträge in die Datenbank und liest Informationen zur Anzeige aus der Datenbank.



Für jedes anzusprechende Device gibt es einen Eintrag in der Tabelle «Device». Dort wird ein ID-String abgelegt, über den man das Gerät auf allen Schichten eindeutig identifizieren kann. Zu jeder Zeile in «Device» gibt es eine eigene Tabelle «DeviceAction_<ID_STRING>». Dort werden die Parameter des Device abgelegt. Im Falle unseres LED-Strips sind das die Start- und Endhelligkeit, die Start- und Endfarbe sowie die Anzahl der LEDs (bzw. Pixel).

In der Tabelle «ActionGroupMember» werden die Geräte zu einer ActionGroup zusammengefasst. Die Tabelle Task ruft also eine ActionGroup auf und in der «ActionGroupMember» Tabelle gibt es für jedes Gerät, dass zu diesem Task etwas ausführen soll, eine Zeile. In jeder Zeile kann ein zusätzlicher Offset angegeben werden, wenn beispielsweise ein Gerät in der ActionGroup erst später anlaufen soll.

Definition Web Service

Um zur Steuerung des WakeUp Lights nicht von einem spezifischen Gerätetyp abhängig zu sein, werden die Steuerungsaufträge sowie die Informationsabfragen über Web Service Abfragen getätigt. Dieser Web Service wird hier zum ersten Mal spezifiziert. Die nachfolgenden Klassendiagramme basieren auf dieser Spezifikation.

Die volle Spezifikation befindet sich in der Projektablage als Excel-Datei.

Web Service Operations

- GetDevice
- AddDevice
- RemoveDevice
- GetAlarm
- AddAlarm
- RemoveAlarm
- GetDeviceAction
- AddDeviceAction
- RemoveDeviceAction
- GetActionGroupMember
- AddActionGroupMember
- RemoveActionGroupMember
- ActivateActionGroup
- DisableActionGroup
- ActivateNightLight
- DisableNightLight

Web Service Requests

Nachfolgend ist eine Übersicht der zu den Operations gehörigen Requests abgebildet. Das Bild ist ein statisches Beispiel. Die Dokumentation wird in der Projektablage in der Excel-Datei nachgeführt.

Operation	Feld	Opt.	Beispiel	DB-Feld	Typ	Unique
GetDeviceRequest	StringID	x	WS2801	Device.StringID	String	Nein
AddDeviceRequest	Name		WS2801	Device.Name	String	Nein
AddDeviceRequest	StringID		WS2801	Device.StringID	String	Ja
AddDeviceRequest	GPIO		1	Device.GPIO	Boolean	Nein
RemoveDeviceRequest	StringID		WS2801	Device.StringID	String	Ja
GetAlarmRequest		x				
AddAlarmRequest	StartTime		9:00:00 vorm.	Task.StartTime	Time	Nein
AddAlarmRequest	Offset	x	600	Task.Offset	Integer	Nein
AddAlarmRequest	RepeatPattern	x	0100010	Task.RepeatPattern	String	Nein
AddAlarmRequest	Enabled	x	1	Task.Active	Boolean	Nein
AddAlarmRequest	ActionGroup		1	Task.ActionGroup	Integer	Nein
RemoveAlarmRequest	ID		1	Task.ID	Integer	Ja
GetDeviceActionRequest	StringID		WS2801	Device.StringID	String	Ja
GetDeviceActionRequest	ID		1	DeviceAction_WS2801.ID	Integer	Ja
AddDeviceActionRequest	StringID		WS2801	Device.StringID	String	Ja
AddDeviceActionRequest	ID		1	DeviceAction_WS2801.ID	Integer	Ja
AddDeviceActionRequest	FieldNName		LuxStart			
AddDeviceActionRequest	FieldNType		Integer			
AddDeviceActionRequest	FieldNValue		100	DeviceAction_WS2801.FieldNName	FieldNTy	Nein
RemoveDeviceActionRequest	StringID		WS2801	Device.StringID	String	Ja
RemoveDeviceActionRequest	ID		1	DeviceAction_WS2801.ID	Integer	Ja
GetActionGroupMemberRequest	GroupID		1	ActionGroupMember.GroupID	Integer	Nein
AddActionGroupMemberRequest	GroupID		1	ActionGroupMember.GroupID	Integer	Nein
AddActionGroupMemberRequest	DeviceID		1	ActionGroupMember.DeviceID	Integer	Nein
AddActionGroupMemberRequest	ActionID		1	ActionGroupMember.ActionID	Integer	Nein
AddActionGroupMemberRequest	Offset		600	ActionGroupMember.Offset	Integer	Nein
RemoveActionGroupMemberRequest	GroupID		1	ActionGroupMember.GroupID	Integer	Nein
RemoveActionGroupMemberRequest	DeviceID	x	1	ActionGroupMember.ID	Integer	Nein
ActivateActionGroupRequest	GroupID		1	ActionGroupMember.GroupID	Integer	Nein
DisableActionGroupRequest	GroupID		1	ActionGroupMember.GroupID	Integer	Nein
ActivateNightLightRequest	StringID		WS2801	Device.StringID	String	Ja
ActivateNightLightRequest	ID		1	DeviceAction_WS2801.ID	Integer	Ja
DisableNightLightRequest	StringID		WS2801	Device.StringID	String	Ja
DisableNightLightRequest	ID		1	DeviceAction_WS2801.ID	Integer	Ja

Web Service Responses

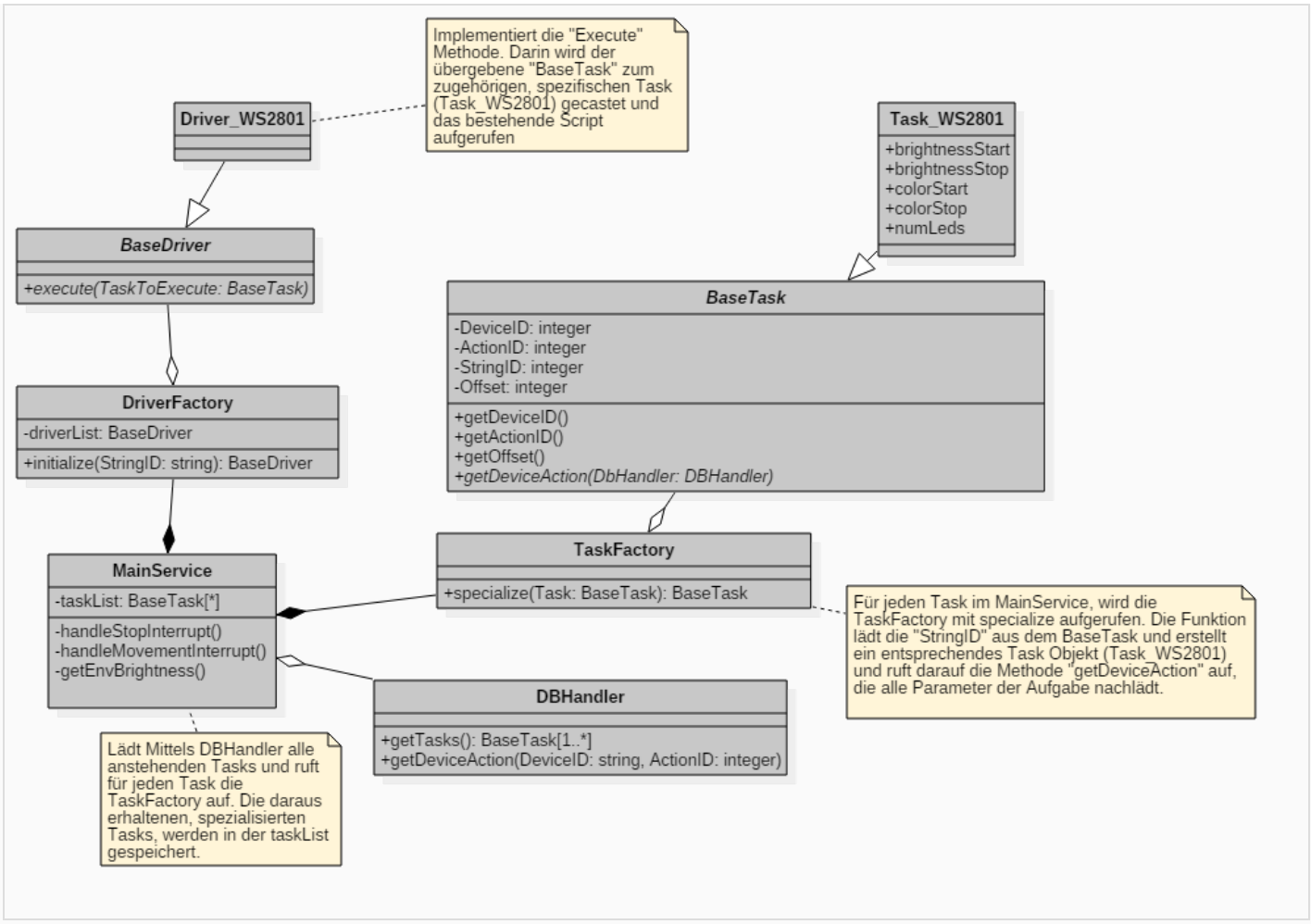
Nachfolgend ist eine Übersicht der zu den Operations gehörigen Responses abgebildet. Das Bild ist ein statisches Beispiel. Die Dokumentation wird in der Projektablage in der Excel-Datei nachgeführt.

Operation	Feld	Anzah	Beispiel	DB-Feld	Typ
GetDeviceResponse	Name	n	WS2801	Device.Name	String
GetDeviceResponse	StringID	n	WS2801_1	Device.StringID	String
GetDeviceResponse	GPIO	n	1	Device.GPIO	Boolean
AddDeviceResponse	Anzahl	1	2		Integer
AddDeviceResponse	Result	1	1		Boolean
RemoveDeviceResponse	Anzahl	1	2		Integer
RemoveDeviceResponse	Result	1	1		Boolean
GetAlarmResponse	ID	n	1	Task.ID	Integer
GetAlarmResponse	StartTime	n	9:00:00 vorm.	Task.StartTime	Time
GetAlarmResponse	Offset	n	600	Task.Offset	Integer
GetAlarmResponse	ActionGroupID	n	1	Task.ActionGroup	Integer
GetAlarmResponse	RepeatPattern	n	0100100	Task.RepeatPattern	String
GetAlarmResponse	Enabled	n	1	Task.Active	Boolean
AddAlarmResponse	Anzahl	1	1		Integer
AddAlarmResponse	Result	1	1		Boolean
RemoveAlarmResponse	Anzahl	1	1		Integer
RemoveAlarmResponse	Result	1	1		Boolean
GetDeviceActionResponse	StringID	1	1	Device.StringID	String
GetDeviceActionResponse	ID	1	1	DeviceAction_StringID.ID	Integer
GetDeviceActionResponse	FieldNName	N	LuxStart		
GetDeviceActionResponse	FieldNType	N	Integer		
GetDeviceActionResponse	FieldNValue	N	100	DeviceAction_WS2801.Fiel	FieldNType
AddDeviceActionResponse	Anzahl	1	1		Integer
AddDeviceActionResponse	Result	1	1		Boolean
RemoveDeviceActionResponse	Anzahl	1	1		Integer

Klassendiagramme

Die nachfolgend gezeigten Klassendiagramme basieren auf dem oben dargestellten Datenmodell sowie der Web Service Spezifikation.

Treiberlayer

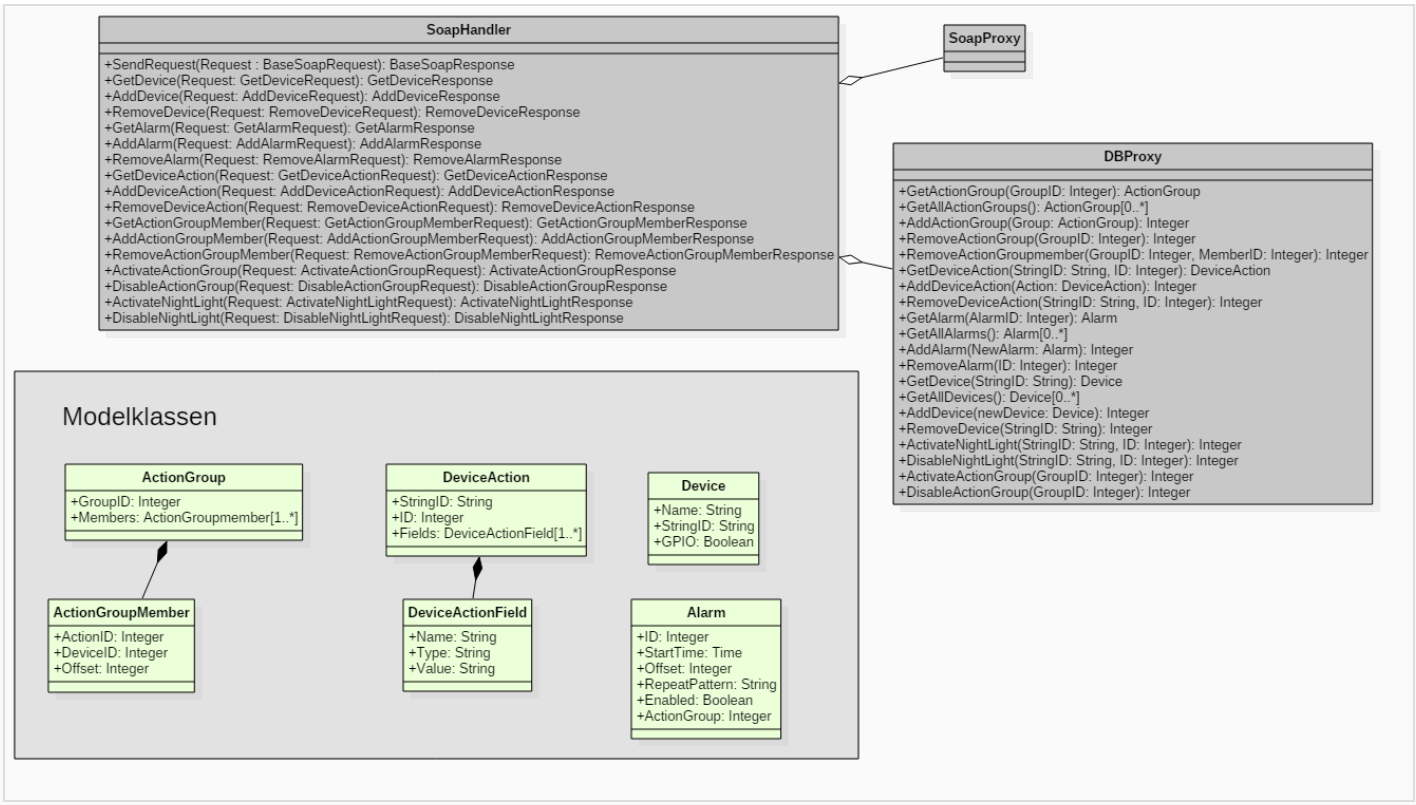


Das Klassendiagramm sieht einen Linux-Daemon vor, der die Hauptlogik enthält. Dieser erstellt einen DBHandler, der regelmässig alle Aufgaben aus der Datenbank lädt. Der DBHandler selektiert alle Tasks die

1. Aktiv sind,
2. Die Uhrzeit erreicht ist,
3. Das RepeatingPattern erfüllt ist und
4. Deren Devices in der ActionGroup GPIO relevant sind

und schickt diese an den MainService als «BaseTask» zurück. Jetzt wird ein Task für jedes anzusprechende Device erstellt. Der MainService ruft für jeden so erstellten Task, die TaskFactory mit «specialize» auf. «Specialize» versucht anhand der StringID, das richtige POCO-Objekt zu erstellen (Task_WS2801) und gibt dieses zurück. Dieses Objekt wird nun im MainService in der «taskList» abgespeichert. Für jeden Task in der taskList wird nun die DriverFactory mit der StringID des Tasks aufgerufen. Die DriverFactory versucht das richtige Driver-Objekt zu erstellen («Driver_WS2801») und gibt dieses als BaseDriver Objekt zurück. Der MainService ruft nun auf dem BaseDriver-Objekt mittels Polymorphismus die «execute» Funktion auf. Die Execute-Funktion ist in jedem expliziten Driver «Driver_WS2801» implementiert und enthält den Scriptaufruf mit den Angaben aus dem jeweiligen Task (Task_WS2801) Objekt.

Middlewarelayer



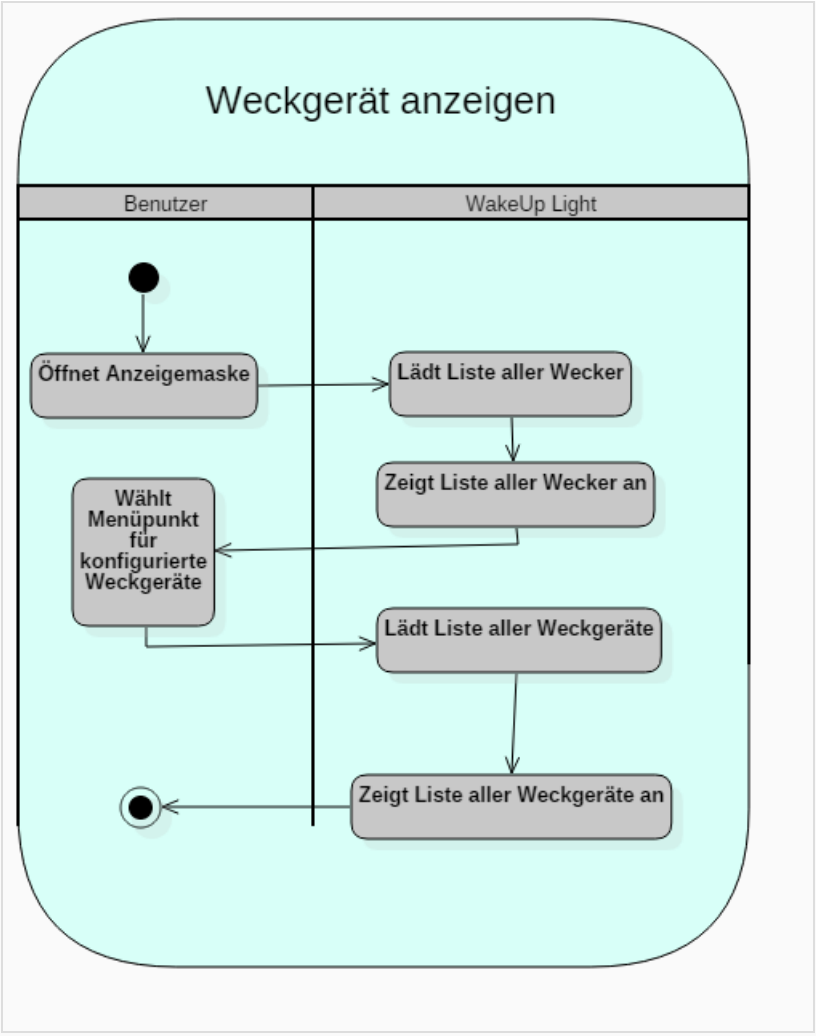
Der SoapHandler schickt bei Bedarf WebService Requests an Komponenten die per Web Service angebunden sind (LIFX) und empfängt WebService Requests, die für das WakeUp-Light gedacht sind. Er implementiert die oben spezifizierten WebService Operationen.

Der DBProxy übernimmt die Kommunikation zur Datenbank. Der SoapHandler ist dafür zuständig, dass er seine Requests richtig interpretiert und die richtige Funktion auf dem DBProxy aufruft.

Der SoapProxy übernimmt die tatsächlichen Verbindungsdetails und Netzwerktechnischen Details. Dieser wird hier nicht weiter ausgeführt, da er für die Funktionsweise der Endsoftware irrelevant ist.

Beispiel Sequenzdiagramm

Das nachfolgende Sequenzdiagramm wurde als Beispiel erstellt.



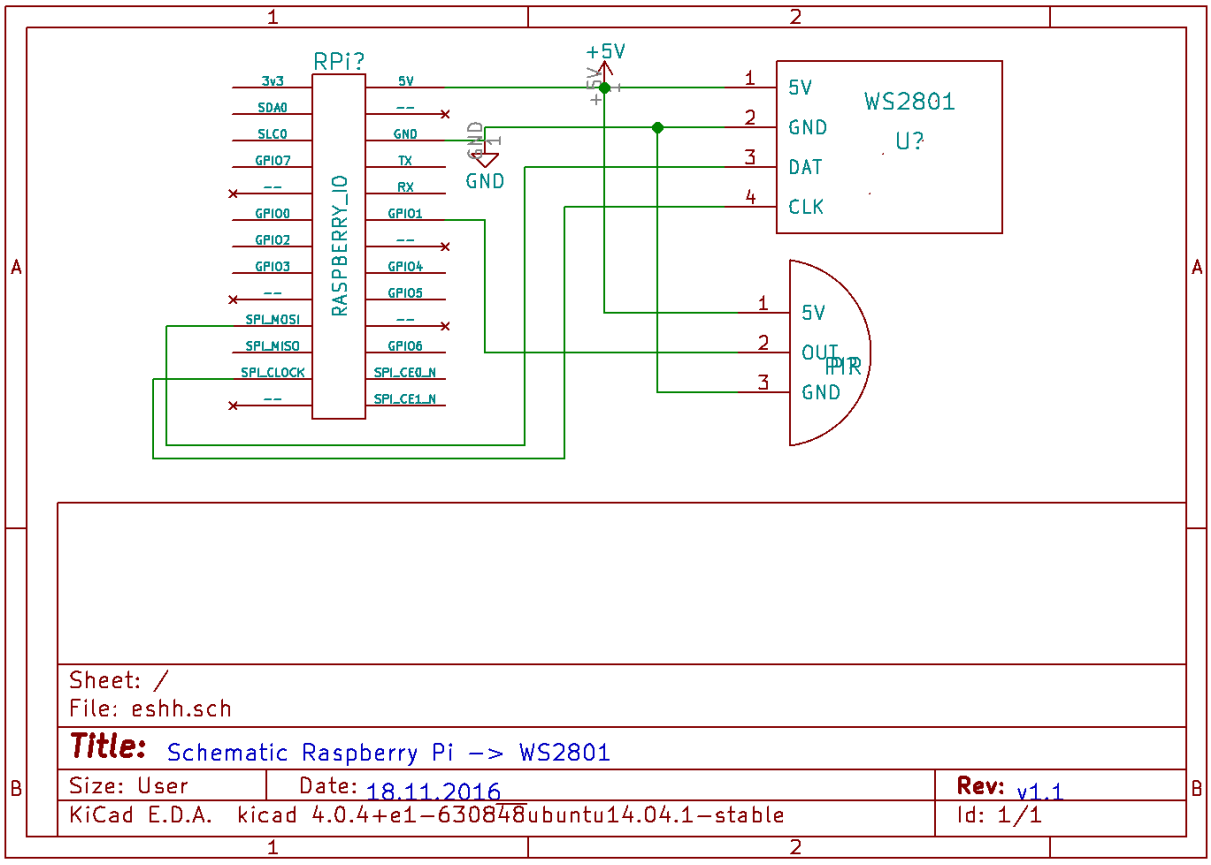
Testplan

Die im Design ausgearbeitete Spezifikation beinhaltet bereits einiges an Funktionalität. Um diese Funktionalität testen zu können, wurde ein spezifischer Testplan erstellt, der die in der Analyse und dem Design ausgearbeiteten Features abdecken soll. Der Testplan wird im Projektrepository als Excel-Datei geführt und ist hier nur auszugsweise als Beispiel aufgeführt.

Zu Testendes Feature	Bemerkung	Ausgangskriterien	zu testende Hand
Startkriterien	Initialisierung	Raspberry pi wird neu gestartet -> Programm wird gestarted	LED's dürfen keine
alle LED's können auf grün geschaltet werden	Treiber	der LED-Strip ist ausgeschaltet	LED's werden auf g
alle LED's können auf rot geschaltet werden	Treiber	der LED-Strip ist ausgeschaltet	LED's werden auf r
alle LED's können auf blau geschaltet werden	Treiber	der LED-Strip ist ausgeschaltet	LED's werden auf b
alle LED's können einzeln auf rot geschaltet werd	Treiber	der LED-Strip ist ausgeschaltet	einzelnes Ansprech
alle LED's können einzeln auf grün geschaltet wer	Treiber	der LED-Strip ist ausgeschaltet	einzelnes Ansprech
alle LED's können einzeln auf blau geschaltet wer	Treiber	der LED-Strip ist ausgeschaltet	einzelnes Ansprech
LED's können einzeln angesprochen werden	Treiber	alle LED's sind ausgeschaltet	LED's können einze
Bewegungslicht	Anwendung	Alle LED's sind ausgeschaltet, der Bewegungssensor hat keine	Bewegungssensor
Bewegungslicht	Anwendung	Bewegungssensor ist aktiviert, LED's eingeschalten	Timer ist abgelaufe
Timer	Anwendung	Alle LED's sind ausgeschaltet, der Bewegungssensor hat keine	Zeit stimmt mit Tir
Timer	Anwendung	Der Timer ist aktiv und beendet sich	Endzeit stimmt mit
Priorität, wenn mehrere Aktionen gleichzeitig aus	Anwendung	Timer ist aktiv, Bewegungsmelder hat keine Bewegung erkannt	Timer hat eine höh
Dimmer zunehmend	Anwendung	Timer ist aktiv, hat gerade eingeschaltet und ist auf "immer he	Led's sollten imme
Dimmer abnehmend	Anwendung	Timer ist aktiv, hat gerade eingeschaltet und ist auf "immer dü	Led's sollten imme

Schaltungsentwurf

Die Schaltung zeigt, wie das Hauptweckmedium (die LED-Pixelkette WS2801) an den Raspberry PI angeschlossen wird. Die Applikation sieht vor, dass auch andere Geräte angeschlossen und angesteuert werden können.


RPi?
RASPBERRY_IO
3v3
SDA0
SLC0
GPIO7
--
GPIO0
GPIO2
GPIO3
--
SPI_MOSI
SPI_MISO
SPI_CLOCK
--
5V
--
GND
TX
RX
GPIO1
--
GPIO4
GPIO5
--
GPIO6
SPI_CE0_N
SPI_CE1_N
+5V
GND
WS2801
U?
5V
GND
DAT
CLK
PIR
5V
OUT
GND
Sheet: /
File: eshh.sch
Title: Schematic Raspberry Pi -> WS2801
Size: User
Date: 18.11.2016
Rev: v1.1
KiCad E.D.A. kicad 4.0.4+e1-630848ubuntu14.04.1-stable
Id: 1/1