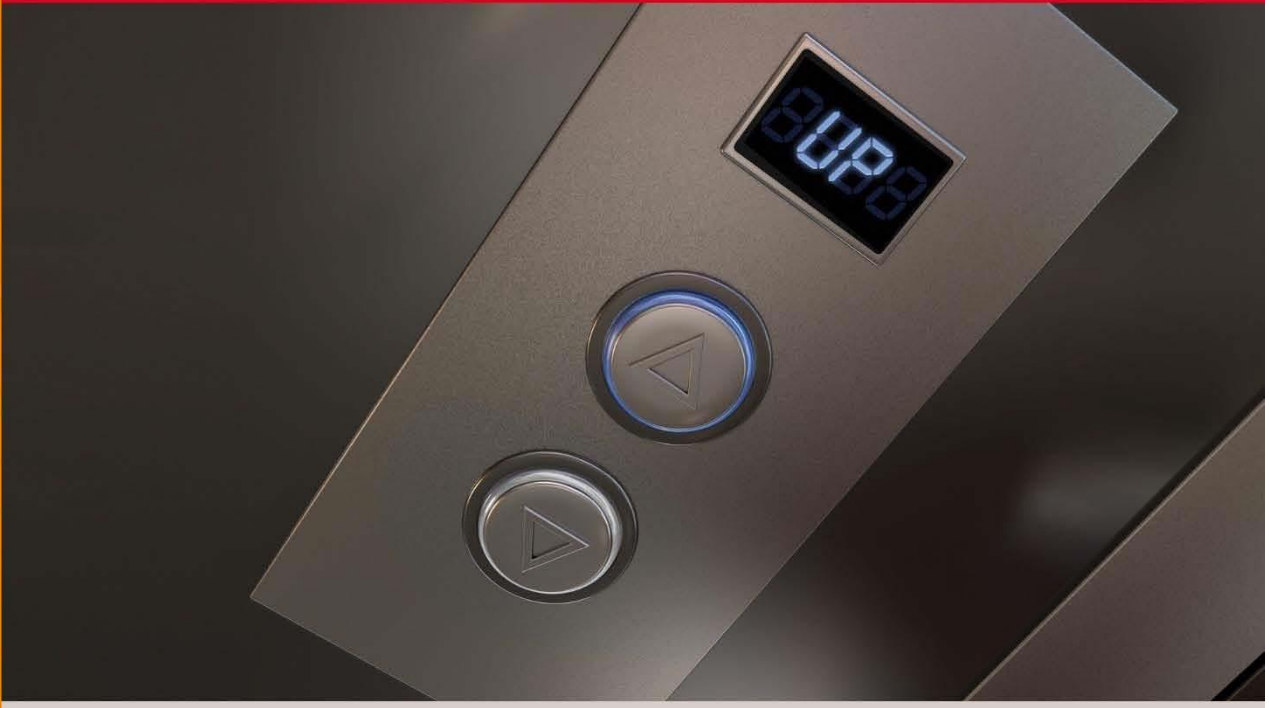
Hue bridge-lighting automation engine : investigate lighting control alternatives for the existing rule-engine

4TU.School for Technological Design STAN ACKERMANS INSTITUTE

Hue Bridge – Lighting Automation Engine

Spyridon Skoumpakis
September 2016



Hue Bridge – Lighting Automation Engine

Investigate lighting control alternatives for the existing rule-engine

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

Philips Lighting

Eindhoven University of Technology

Steering Group Spyridon Skoumpakis
George Yianni
Daniel Goergen
Walter Slegers
Tanir Ozcelebi
Ad Aerts

Date September 2016

Document Status Public

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.097B, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31402474334

Published by
Eindhoven University of Technology
Stan Ackermans Institute

Printed by
Eindhoven University of Technology
UniversiteitsDrukkerij

SAI report no.
Eindverslagen Stan Ackermans Instituut ; 2016/036

Abstract

Philips Hue is an internet connected (wireless) lighting system designed to transform how users experience light inside their homes. It is one of the leading and most installed smart home / Internet of Things products in the world. Philips Hue enables color tunable lights to be controlled from smartphones, web services or other control logic and devices running in the system.

The brain of the Hue system is an embedded device called Hue bridge. The Philips Hue bridge controls and monitors ZigBee lights, sensors, and switches; it acts as local home lighting controller. The bridge communicates both in IP and ZigBee networks and actually facilitates the message translation from one to another.

The bridge uses a rule engine, which receives switch, sensor, or timer triggers and then sets a specific lighting scene as result. This engine is a software module responsible for the automation logic of the bridge. With an increasing complexity of home lighting control use cases the need of exploring more sophisticated automation engines was imperative. As the first step, an investigation of alternative engines is conducted having a comparison table as main output. Furthermore, a formal specification for a future Lighting Automation Engine is developed, and coupled with a prototype. The specification describes ways to transcend the strict limitations of the existing engine introducing the power of scripting languages. The project also lists several suggestions for future improvements.

Keywords
Home automation, lighting automation, smart lighting, rule engine, scripting engine, automation engine, Philips Lighting, Philips Hue, TUE, Software Technology, PDEng

Preferred reference
Hue Bridge – Lighting Automation Engine: Investigate lighting control alternatives for the existing rule-engine. SAI Technical Report, September 2016. (Eindverslagen Stan Ackermans Instituut ; 2016/036)

Partnership
This project was supported by Eindhoven University of Technology and Philips Lighting.

Foreword

The Internet of Things and your whole home connected and automated is a predicted future for which we see the first steps taking place with connected consumer devices. Did you ever wonder what is the most common electronic device that is part of that, one that is present in all your rooms? Your lights. A connected home needs connected lighting. Control your lights with your smart phone. Define automated lighting behavior with your smart phone, e.g., which lights to switch on when you enter a room. And also if these lights should emit a warm white light, cold white light, or have a color. The latter might depend on the time of day, the weather, your personal agenda, and so on. We probably haven’t imagined the possibilities.

Philips Hue is a connected home lighting system, on a journey to discover the possibilities. And to allow the world to discover the possibilities, the Philips Hue system is an open system, it allows 3rd party application developers to build apps for Hue. Apps to control your lights, but also to define and create lighting automation behavior. This automation has limitations, some intentionally, some due to resource constraints. The time has come to enable more.

How should a next step in lighting automation be designed? It should be open for other app developers, but not cause users to be surprised on unexpected lighting behavior. A home user should be in control and understand what’s happening. This document from Spyridon Skoumpakis describes requirements for home lighting automation. It includes an inventory of what exists in the world today, to learn from others. It suggests a direction for the Philips Hue system on how to improve its current lighting automation into one that allows more flexibility to handle the needs of the emerging IoT world while preventing a loss of control. It serves as input on how to proceed with lighting automation for Philips Hue.

W. Slegers
September 2016

Preface

This report offers a detailed account of the graduation project for the PDEng (Professional Doctorate in Engineering) Software Technology program on behalf of the Eindhoven University of Technology and the Stan Ackermans Institute. This project was carried out in Philips Lighting, a company that designs, develops, and produces lighting solutions and applications both for professional and consumer markets, over a period of nine months from January until September 2016.

The project's goal is to evaluate the author as a software designer, while providing Philips Lighting with a modern specification proposal for a future Lighting Automation Engine that will be used in the Hue bridge v2. The original need was to investigate engine alternatives in order to support sophisticated lighting automation use cases. This report contains insights, the design as well as the description of the process that led there. Therefore, in addition to the new design, the domain, project management, conclusions, and retrospective are explained in corresponding chapters.

This report is primarily intended for readers with a technical background. However, certain chapters may be interesting for non-technical readers, such as project managers and home automation enthusiasts.

Spyridon Skoumpakis,
September 2016.

Acknowledgements

A number of people have contributed to the successful completion of this project. First, I would like to express my appreciation to Daniel Goergen and Walter Slegers my supervisors from Philips Lighting. In the context of our weekly synchronization meetings they provided guidance and support throughout all the phases of this project. Furthermore, I would also like to thank Tanir Ozcelebi, my TU/e supervisor, for his valuable contribution and meaningful feedback.

This project would not have been a success without the support from domain experts, and there were many. The two best examples of such people are Ino Dekker and Michel van de Wetering who despite their really busy agendas were willing to answer every single of my questions, and there were many. I thank you for your patience, critical thinking, advices, and continuous support during this project.

Additionally, I would also like to thank everyone involved in the PDEng program, especially our program director Ad Aerts. Thank you for our conversations, your understanding, your advices and everything; this program and my experience would not be the same without you.

Last but not least, I would like to express my gratitude, from the bottom of my heart, to three people for their continuous support during this program, and for believing in me. First and foremost, a special thank you goes to my mother, Eleni, for her incredible patience and support. Second, a special thank you goes to my best friend, Christos, for everything he has done for me and most importantly for being physically present, and finally a special thank you goes to my cousin John for his support in difficult times.

Spyridon Skoumpakis,
September 2016.

Executive Summary

Philips Hue is a personal wireless lighting system designed to transform how users experience light inside their homes. It is one of the leading and most installed smart home / Internet of Things products in the world. Philips Hue transforms how users can experience light by enabling color tunable lights controlled from smartphones, web services or other control logic and devices running in the system. Furthermore, it is an open system, i.e., via standardized or published interfaces other suppliers or developers can add components.

The brain of the Hue system is an embedded device called Hue bridge. The Philips Hue bridge controls and monitors ZigBee lights (Hue lamps), sensors, and switches; it acts as local home lighting controller. The bridge communicates both in IP and ZigBee networks and actually facilitates the message translation from one to another.

The bridge uses a rule engine, which receives switch, sensor, or timer triggers and then sets a specific lighting scene (predefined light attributes state) as result. This engine is a software module responsible for the automation logic of the bridge. With an increasing complexity of home lighting control use cases the need of exploring more sophisticated automation engines was imperative.

Improving home lighting experience is of great importance to Philips Lighting. The home automation environment is rapidly evolving and companies need to follow and satisfy the new trends. Focusing on lighting, an important relevant criterion is being able to explain to a user of the system why a light changed. The existing Hue engine offers almost no information for its internal activities. Additionally, as the lighting automation community grows the need of having abstraction layers for different user categories is continuously emerging. Developers ask for flexibility and functionality and end users ask for usability and simplicity. Besides these fundamental examples, there are many more that Philips Lighting wants to achieve in order to maintain its leading position as a provider of lighting solutions and applications.

To address these challenges Philips Hue department initiated this project. In the context of the project in question, a thorough investigation for existing Lighting Automation Engines conducted and useful insights extracted. A comparison table of open-source alternatives created and some important design decisions made and documented.

A formal specification for a future Lighting Automation Engine is developed using JSON Schema. The specification describes ways to transcend the strict limitations of the existing engine introducing the power of scripting languages and abstraction layers. The specification is separated in three parts implementation, instances and interface. The implementation scripts offer generic reusable behavior to native and third-party applications, the instances offer real-world examples of behavior and the interface provides a contract-bridge between the two. These three pillars constitute the formal description of an engine capable to satisfy the most important requirements and to serve as guideline for future implementation.

The architecture defined is modular and works in parallel with the existing system as long as the interfaces between them remain the same. The benefits of scripting are countless and as long as the internal complexity is hidden and controlled within the designed abstraction layers the end-user experience will be the desirable. Many important features that the existing engine lacked such as the usage of variables, full Boolean logic, grouping and logging come practically out-of-the-box.

Last but not least, the project lists several suggestions for future improvements.

Table of Contents

List of Figures

List of Tables

1.Introduction

In this chapter, the context of the project is presented alongside with a brief reference to the involved parties, followed by the outline of the project per chapter.

1.1 Context

The assignment described in this report is part of nine-month collaboration between Eindhoven University of Technology and Philips Lighting B.V. under the auspices of the Software Technology designer program. This Professional Doctorate in Engineering (PDEng) program is offered by Stan Ackermans Institute 4TU.School for Technological Design.

Stan Ackermans Institute (SAI) is a federation of four leading Dutch technical universities: TU Delft, TU Eindhoven, University of Twente, and Wageningen University. The federation aims at maximizing innovation by concentrating the strengths in research, education and knowledge transfer of all technical universities in the Netherlands. The SAI manages more than twenty post-graduate technical designer programs across the four technical universities. Each designer program is intended to teach the skills needed to design the complex systems needed in the high tech industry to new master's graduates who are starting their careers.

The goal of a PDEng program is to provide an additional dimension to a full master's program by extending it and adding new elements to it. A PDEng trainee further develops skills for synthesis and interdisciplinary work, acquiring the competencies to create innovative technological solutions for products, processes, and systems. The solutions are based on functional requirements as well as on business and market requirements, within the context of society as a whole. The technological designer program takes two years to complete. During the first year, extensive knowledge and experience of the latest design methods and their applications gained through in-house team projects. The second year of the program is spent in industry where the PDEng trainee works on an individual assignment. [1]

1.2 Outline

This report is organized in the following chapters:

- **Chapter 2 – Stakeholder Analysis:** presents the identified stakeholders. The main stakeholders for this project were identified in the early phase of this project based on different points of interest for the Lighting Automation Engine.

- **Chapter 3 – Domain Analysis:** describes the context of Home Automation around Hue Lighting and important information about lighting control in general. This information aims to provide the terms that will give the reader a better understanding of the rest of the report.

- **Chapter 4 – Problem Analysis:** gives the overview of the problem at hand, a more detailed analysis of the Hue ecosystem, and the expected outcome of this project, i.e. a Lighting Automation Engine.

- **Chapter 5 – Feasibility Analysis**: presents the feasibility analysis of the problem at hand. It shows the relation to the challenges and risks that were identified in the early stages of the project.

- **Chapter 6 – System Requirements**: shows the process used to gather the project's requirements. Following this, the important scenarios that concern the Lighting Engine together with the functional and non-functional requirements are described. Finally, the identified design criteria that are important for the success of this project conclude this chapter.

- **Chapter 7 – Design Alternatives**: presents the investigation of alternative engines and home automation systems along with its outcome. Additionally, this chapter discusses some important design decisions. These decisions are the foundations of the new Lighting Engine's architecture.

- **Chapter 8 – Engine Specification:** focuses on the creation of a formal specification for the new engine. First, it describes the specification, starting from defining what JSON is and then what JSON Schema is. Second, it continues with Schema's implementation for the new Lighting Automation Engine which is one of the two major outputs of the project.

- **Chapter 9 – Schema Validation:** This chapter provides an account of the suitability of the created JSON Schema to meet the system requirements as listed in Chapter 6. This chapter discusses the various techniques for validation.

- **Chapter 10 – Conclusions - Results:** shows the results of this project. The two main achievements are the outcome of the alternatives investigation and the JSON Schema specification prototype.

- **Chapter 11 – Project Management**: presents an overview of the project management techniques used in this project. The initial planning and how it evolved is also explained using the Work-Breakdown Structure. Finally, the planning methodology and timeline of the project are presented accompanied with some explanatory information for its execution.

- **Chapter 12 – Project Retrospective:** offers a reflection on the project, looking back into what proved to be good practices and what could have been improved. Furthermore, the design criteria are revisited and their role on the outcome of the project is reexamined.

2.Stakeholder Analysis

In this chapter, the identified stakeholders are mentioned. Detailed description of the involved parties is given and a list of the main stakeholders per party follows accompanied with a table of all affiliates and a visual representation of the analysis.

2.1 Eindhoven University of Technology (TU/e)

The Eindhoven University of Technology is responsible for the educational aspect of this project and fulfilling the requirements for a project of this type. That means certain standards need to be met. The TU/e is concerned with the design process, project management, and implementation.

2.2 Software Technology Program

The Professional Doctorate in Engineering (PDEng) degree program in Software Technology is provided by the Department of Mathematics and Computer Science of Eindhoven University of Technology (TU/e) in the context of the 4TU.School for Technological Design, Stan Ackermans Institute.

This Professional Doctorate in Engineering program (PDEng) is an accredited and challenging two-year, third-cycle (doctorate-level) engineering degree program during which its trainees focus on strengthening their technical and non-technical competencies related to the effective and efficient design and development of software for resource-constrained software-intensive systems, such as real-time embedded or distributed systems, in an industrial setting. In particular, the focus is large-scale project-based design and development of this kind of software.

The Software Technology program is designed to prepare people for an industrial career as a technological designer, and later on as a software or system architect. It starts with 15 months of advanced training and education, including four small, industry driven training projects, followed by a major design project of nine months in a company. [2]

2.3 Philips

In 1891, Gerard Philips, together with his father Frederik Philips, founded the firm Philips & Co. The company was established in empty business premises in Eindhoven. There was already considerable competition in the lamp market at that time. Gerard’s distinctive approach was to concentrate fully on the mass production of incandescent lamps. In 1895 Gerard’s brother Anton Philips joined the firm to look after sales, a move that proves successful. In 1895, 200,000 incandescent lamps were sold, and three years later more than 1,000,000. By the end of the 1890s Philips & Co. was one of the largest producers in the Netherlands and, with 1,000 employees, the country’s largest industrial employer. [3]

Today, Koninklijke Philips N.V. (Royal Philips or the ‘Company’) is the parent company of the Philips Group (‘Philips’ or the ‘Group’). The Company is managed by the members of the Board of Management and Executive Committee under the supervision of the Supervisory Board. The Executive Committee operates under the chairmanship of the Chief Executive Officer and shares responsibility for the deployment of Philips’ strategy and policies, and the achievement of its objectives and results.

Headquartered in Amsterdam, the Netherlands, Philips employs over 112,000 employees (December 2015) with sales and services in more than 100 countries worldwide. With sales of EUR 24.2 billion in 2015, the company is a market leader in cardiac care, acute care and home healthcare, energy efficient lighting solutions and new lighting applications, as well as lifestyle products for personal well-being and pleasure (Annual Report 2015).[4]

In September 2014, Philips announced its plan to sharpen its strategic focus by establishing two standalone companies focused on the HealthTech and Lighting opportunities respectively. A stand-alone structure for Philips Lighting has been established within the Philips Group, effective February 1, 2016.

2.4 Philips Lighting

As of February 2016, two standalone operating companies emerged within Royal Philips, focused on the HealthTech and Lighting opportunities respectively. While the businesses in Lighting and HealthTech operate independently, Lighting remains a wholly-owned Philips business and will be until the board of directors identifies and executes the right strategic option for its future. The Royal Philips Executive Committee, of which both Frans van Houten (CEO of Royal Philips) and Eric Rondolat (CEO of Philips Lighting) remain members, continues to oversee both businesses in Lighting and HealthTech. Functional reporting lines into Royal Philips for Finance, Legal, Communications, and HR will remain in place until full separation is achieved.

Philips Lighting Solutions B.V. is the leading provider of lighting solutions and applications for both professional and consumer markets, pioneering in how lighting is used to enhance the human experience in the places where people live and work. Whether being at home, on the road, in the city, shopping, at work or at school, Philips Lighting is creating lighting solutions that transform environments, create experiences, and help shape identities. Philips Lighting serves its customers through a market segment approach, which encompasses Homes, Office and Outdoor, Industry, Retail, Hospitality, Entertainment, Healthcare and Automotive. The company employed approximately 33,600 people worldwide with sales of EUR 7.4 billion in 2015. In 2015, Philips Lighting spanned a full-service lighting value chain – from lamps, luminaires, electronics and controls to connected and application-specific systems and services.

Philips Lighting is a global market leader with recognized expertise in the development, manufacture and application of innovative, energy-efficient lighting products, systems and services that improve people's lives. The company has pioneered many of the key breakthroughs in lighting over the past 125 years, laying the basis for its current strength and leading position in the digital transformation.

As of **February 2016,** the structure of the company changed considerably. Lighting is a rapidly evolving environment and Philips as a global leader is dynamically shaping its future. One of the main focuses of the company is home lighting automation. In the context of this domain, Philips Lighting has many projects following the interest and needs of the market. The current project is about a home Lighting Automation Engine.

2.5 Main stakeholders

The main group of people who were involved in the steering process of the project are presented below:

2.5.1. TU/e

Ad Aerts (ST Program Director)

He is the general director of Software Technology PDEng program since 2008 and thus he is responsible for supervising the collaboration of the two parties of each design project.

Tanir Ozcelebi (TU/e supervisor)

He is an assistant professor in Security and Embedded Networked Systems at the Department of Mathematics and Computer Science and the research program manager for the Bright Environments research program of TU/e Intelligent Lighting Institute since 2013. Besides his mentorship, his role included making sure that the design and documentation met the standard of a PDEng project.

Spiros Skoumpakis (PDEng Philips Trainee)

He is a PDEng candidate responsible for the implementation of the project.

2.5.2. Philips Lighting

George Yianni (Project Owner)

He is the head of technology and creator of Philips Hue, responsible for the technology choices made in the connected lighting business of Philips, which includes the hue product. This ranges from design of new features, architectures and products to choices of which technology standards and platforms to adopt. He is the initiator and owner of the project in question.

Daniel Goergen (Project Manager)

He is a System Architect in the Philips Hue department.

Walter Slegers (Project Mentor)

He is a Software Architect in the Philips Hue department. Walter, Daniel and Spiros were the three main responsible people for the outcome of the project being in close collaboration since its beginning.

2.6 Stakeholder Analysis

Different people were involved in different phases of the project and influenced its outcome in various ways. An effort of visualizing their contribution is presented in the following diagram.

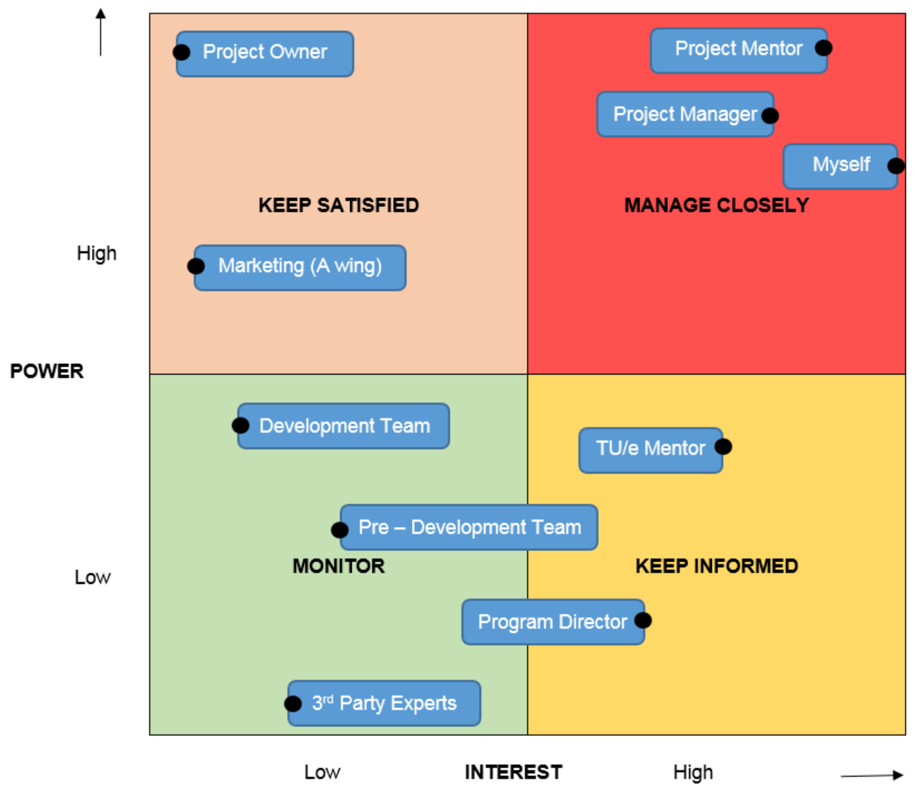


Figure 1 Stakeholder power-interest diagram

- **High power, interested people:** these are the people you must fully engage and make the greatest efforts with e.g., the direct supervisors of the project who are actively steering the process.
- **High power, less interested people:** provide sufficient information to these people to ensure that they are up to date but not overwhelmed with data e.g., the head of the department who initiated the project.
- **Low power, interested people:** keep these people adequately informed, talk to them to ensure that no major issues arise. These people can help with the detail of the project e.g., End Users, other Project Managers, and Business Community.
- **Low power, less interested people:** provide these people with minimal communication to prevent boredom e.g., other departmental members, teams unaffected by the change.

3.Domain Analysis

In this chapter, we focus our interest inside the structure of Philips Lighting. The domain around the Lighting Automation Engine is described and analyzed focusing incrementally on the important constituents. To be more specific, we begin with a small historical overview of lighting and we continue with explaining the concept of Home Automation, some milestones, some related systems and protocols. Finally, we scratch the surface of lighting control, the issue at hand.

3.1 Introduction

Since its creation in 1891, Philips has been a product-oriented company, selling thousands of conventional lighting products such as incandescent, halogen, and fluorescent technology bulbs around the world. However, new technologies, such as light emitting diode (LED) devices, have completely changed the market and, therefore, the business strategy. Since LED diodes are essentially a product from the semiconductor (chips) industry, new entrants are coming to the lighting market, and lighting installations are expected to become more intelligent, dynamic, and personalized. To overcome these challenges, Philips Lighting is transforming from being a largely product-focused company to a solution-oriented company in which products are seen as building blocks.

As one can easily understand, a big company like Philips Lighting is involved in many activities. In fact, the company serves a large and attractive market that is driven by the need for more light, the need for energy-efficient lighting, and the need for digital and connected lighting. The world's population is forecast to grow from 7 billion today to over 9 billion by 2050. At the same time, we are witnessing rapid urbanization, with over 70% of the world's population expected to live in urban areas by 2050. These trends will increase demand for light. In addition, in the face of resource constraints and climate change, the world needs that light to be energy efficient; at the same time, the lighting industry is moving from conventional to LED lighting, which is changing the way people use, experience and interact with light. Digital technologies enable connectivity and seamless integration in software architectures, systems, and services. Connected lighting allows light points to be used as information pathways opening up new functionalities and services based on the transmission and analysis of data. [4]

The lighting market is expected to grow by 2 to 4% per annum between 2015 and 2019 (source: BCG). The majority of this growth will be driven by LED-based solutions and applications – heading towards a 60 to 65% share by 2018.

Figure 2 History of Light Bulbs (oil, incandescent, fluorescent, LED) [7]

The latest (2016) structure of the company presented in Chapter 1 illustrates the wide span of areas of interest inside the Lighting domain. The specific domain of interest for this project is the **Home Business Group**. This group inside Philips Lighting is responsible for – among others – research and development of Home Automation products. Within the home group, there are different business units and the one that encompasses the scope of this project is the **Home Systems Business Unit** also known as **Philips Hue**.

3.2 Home Automation

Home Automation is the use and control of home appliances remotely or automatically. The same concept can be found with various names such as smart home, digital home, e-home, intelligent household and domotics.

The idea of Home Automation is not a recent concept in any way but it has been more of a case of technology catching up with the idea. Home Automation was a topic of science fiction for many years and in Ray Bradbury's short story "There Will Come Soft Rains" (1950), he wrote about an automated home that continues to work despite no one living in it. [8]

A brief list of milestones in the history of Home Automation follows:

- **Remote controls** – It all started with the wireless remote control, which was first unveiled by Nikola Tesla in 1898 when he controlled a miniature boat by sending radio waves.
- **Domestic Appliances** – The 20th century started with the boom in home appliances such as the vacuum cleaner engine in 1901 and the electric powered vacuum six years later. Throughout the next two decades was the revolution in home appliances with refrigerators, clothes dryers, washing machines, irons, and toasters. However, these were expensive and only afforded as a luxury for the wealthy.
- **ECHO IV** – The idea of Home Automation was flirted within the 1930's when the earliest working prototypes of automated houses debuted at the World's Fairs in Chicago and New York City, but those homes were never intended to be commercially available. It was not until 1966 that Jim Sutherland developed the first Home Automation system "Echo IV", which would make a shopping list, control temperature and turn appliances on and off, but this was also never commercially sold.
- **Kitchen Computer** – 1969 saw the Honeywell Kitchen Computer, which was a computer that would create recipes, although this had no commercial success due to the price.
- **Microcontroller** – The microprocessor came in 1971 and this meant a rapid price fall in electronics; consequently, technologies became more accessible to everyone.
- **Smart Home** – This term was first coined by the American Association of Home Builders in 1984.
- **Ubiquitous Computing –** is a term coined around 1988 and refers to a software engineering concept where computing is made to appear anytime, everywhere using any device in any format.
- **Gerontechnology** – Through the 1990's there was a new focus on combining gerontology with technology to help improve the lives of the elderly and less able.
- **Ambient Intelligence** – (Aml) refers to electronic environments that are sensitive and responsive to the presence of people. It was a vision on the future of consumer electronics, telecommunications, and computing; it was originally developed in the late 1990s for the time frame 2010-2020.

- **Domotics** – By the end of the century, this term was commonly used to describe how domestic appliances were now being combined with computers and robots. Despite this development in making this technology more accessible, it was still very expensive and lacked any widespread uptake, and was left for the rich.
- **Integer millennium house** – Opened in 1998, this demonstration home in Watford, England showcased how Home Automation could be integrated to it a home with heating systems, automatic garden controlling soil, security systems, lights and doors.
- **Start of the technology revolution** – Gradually as technology became more affordable, these technologies slowly became integrated in our homes. As these became more popular, there was more investment into making them efficient, cheaper, and thus more accessible.

Figure 3 Home Automation Services [9]

- **Internet of Things** – (IoT) as a part of this revolution, this term is the new domain "hype" and was coined in 1999 by a British entrepreneur. It is essentially the network of physical objects (Things) – devices, vehicles, buildings and other items – embedded with electronics, software, sensors, and network connectivity that enables these objects to collect and exchange data.
- **Now** – Nowadays Home Automation is everywhere, and we are not always aware of it. We can now control our TVs, heating, lights, alarms and doors all via our smart phones and controllers.
- **The future** – Our imagination is our only limitation with technology as advanced as is it is today we can make almost anything such as mirrors that are TVs, smart wardrobes, and smart ovens. [8]

In this context, Philips made a **strategic choice** to be the expert in Lighting (Automation) and not the center of smart homes. The existing knowledge of the company in the specific domain was the main drive for that choice. Additionally, another main reason was that lighting control is the most common usage scenario of a Home Automation system. Lighting is ubiquitous and the main gate of IoT (Internet of Things) penetration in contemporary households.

3.2.1. Home Automation Systems

Besides the abovementioned examples, there are several recent Home Automation systems that are worth mentioning in the IoT context. The list is for sure not exhaustive but the following systems came up repetitively during the meetings of the author with various Philips' experts. The systems are separated in two categories the commercial and the Open Source. In a later section we will put more focus on the Open Source solutions; here we give brief descriptions for both.

Commercial Systems

SmartThings

SmartThings is a company founded in 2012 and a proprietary Home Automation system. SmartThings' primary products include a free SmartThings app, a SmartThings Hub, as well as various sensors and smart devices.

The SmartThings native mobile application allows users to control, automate, and monitor their home environment via mobile devices. Customers can use the app to connect multiple devices at once or follow a dedicated path to configure one device at a time.

The hub connects directly to a home's internet router and is compatible with communication protocols such as ZigBee, Z-Wave, and IP-accessible devices. It serves to connect sensors and devices to one another and to the cloud, allowing them to communicate with the SmartThings native app.

Last but not least, the company was acquired by Samsung in August 2014 [10].

HomeKit

Apple has no smart home devices of its own but it has HomeKit technology, a proprietary software framework for controlling and interconnecting devices around a household. The frontend of the system is both Siri and the Home app in iOS. HomeKit provides integration between accessories that support Apple's Home Automation Protocol and iOS devices. A public API is offered for configuring and communicating with those devices. HomeKit was first introduced in 2014 as a part of iOS 8 [11].

Fibaro

Fibar Group began as a spin-off from Poland in 2010 and they now have one of the most powerful proprietary home automation platforms. The system offers its own hub, smart accessories and applications for a household, but it can also be integrated with other systems such as Philips Hue. Fibaro uses mainly Z-Wave protocol for its communication. Additionally, it offers different abstraction layers for different user categories such as full Lua functionality for experienced users and a visual block interface for less technical users [12].

Wink

Wink is a brand of software and hardware products that connect with and control smart home devices from a consolidated user interface. Wink was founded in 2014 as a spin-off but now is an independent subsidiary of another company named Flex. As of 2016, Wink is connected to 1.3 million devices.

Furthermore, it connects with smart home devices associated with the Internet of Things, such as thermostats and Wi-Fi-enabled lights, to provide a single user interface on a mobile app or via a wall-mounted screen, called Relay. The mobile app is free, while consumers pay for a Wink Hub, or Wink Relay, which connects with smart devices in the home. The hubs integrate with competing software standards used by different manufacturers. All the processing activities are implemented in the Cloud. In February 2016, new features were introduced to allow Wink to operate on the local network, in case a user's internet connection is down. In June 2016, compatibility with Uber, Fitbit, and IFTTT, was added to the Relay product. [13]

WeMo

WeMo is a series of commercial products from Belkin International first launched in 2012. WeMo enables users to control home electronics from anywhere. The product suite includes a switch, motion sensor, Insight Switch, light switch, camera and app. The WeMo Switch can be plugged into any home outlet, which can then be controlled from an iOS or Android smartphone running the WeMo App, via home Wi-Fi or mobile phone network.

- The WeMo Motion Sensor can be placed anywhere, as long as it can access the same Wi-Fi network as the WeMo devices it is intended to control. It can then turn on and off any of the WeMo devices connected to the Wi-Fi network as people pass by.
- The WeMo Insight Switch provides information on power usage and cost estimation for devices plugged into the switch.
- The WeMo Light Switch is for use where a light is controlled by a single light switch. Multi-way switching is not supported at this time.
- The WeMo App controls the WeMo devices from anywhere in the world as long as the WeMo devices' wireless network is connected to the Internet. WeMo devices can also be controlled using IFTTT technology. WeMo devices can also be controlled by voice through the Amazon Echo. [14]

Open Source Systems

OpenHAB

The Open Home Automation Bus is an open source, technology-agnostic home automation platform written in pure Java with an OSGi architecture for modularization. Being hardware/protocol agnostic, OpenHAB allows users to integrate and connect a variety of devices from classical home automation systems, such as KNX, Z-Wave, Insteon, EnOcean,, to new Internet of Things (IoT) gadgets and devices, such as Koubachi, Sonos, Nest Labs, Philips Hue, GE Link and custom built Arduino nodes and sensors.

OpenHAB is controlled by a single user interface accessible from a standard web browser or user created Android and iOS applications. Through this interface, users can manage all aspects of their smart home by creating automation rules or scenes and leveraging data from RESTful API's to control everything from lighting to irrigation, and more.

OpenHAB was initially released in 2010 and today is a part of Eclipse Smart Home project, the most active open source home automation community. [15]

OpenRemote

OpenRemote is an open source project, started in 2009, with the ambition to overcome the challenges of integration between many different protocols and solutions available for home automation, and offer visualization tools. OpenRemote Inc. was created, to enable the sponsorship of the OpenRemote open source project – in the vein of JBoss.

OpenRemote is software integration platform for residential and commercial building automation. OpenRemote platform is automation protocol agnostic, operates on off-the-shelf hardware and is freely available under an open source license. OpenRemote's architecture enables fully autonomous and user-independent intelligent buildings. End-user control interfaces are available for iOS and Android devices, and for devices with modern web browsers. User interface design, installation management and configuration can be handled remotely with OpenRemote cloud-based design tools. [16]

Home Assistant

Home Assistant is a relatively new (December 2014) open-source home automation platform running on Python 3. The goal of Home Assistant is to be able to track and control all devices at home and offer a platform for automating control. Home Assistant can be extended by components. Each component is responsible for a specific domain within Home Assistant. Components can listen for or trigger events, offer services and maintain states. Exactly like OpenHAB Home Assistant, aims to be protocol agnostic and thus supports the integration of different technologies and devices controlled by one application. [17]

There are various other relevant systems open source, proprietary or even independent projects (e.g., GitHub and npm) that one can find online. Some of them were documented separately due to lack of space and can be provided upon request.

3.2.2. Home Automation Protocols

There are several different Home Automation protocols (some of them already mentioned above) and, based on the specific set up requirements, there is always one best choice. The list is not exhaustive a brief description follows each one: [18]

- **X10**, developed in the 1970s, is the oldest Home Automation protocol. X10 is a simple system that uses the power lines in a home to allow communication between devices and appliances. Since X10 uses the power lines, it is very reliable but subject to interference from other electrical devices in the circuit. Special noise filters can mitigate this interference. X10 is a primitive system and can only perform about 16 commands, sent one at a time.
- **Insteon** combines wired and wireless communication into a single system that offers great reliability and flexibility. The power line is typically used as a backup to the RF frequency used by the system. This allows commands to reach the proper destination with little to no interference. Insteon supports over 65,000 different commands and is one of the best options for upgrading the light switches in a home. Insteon offers limited compatibility with X10 devices, but with the proper equipment, one can streamline an older X10 system with Insteon technology.
- **UPB** (Universal Powerline Bus) is a wired system developed in the late 1990s as an improvement to the technology that undergirds X10. UPB reduces the interference that sometimes plagues X10 by using high-power pulses to send its commands over power line circuits. UPB sends commands faster and can handle greater voltage loads than X10, enabling a broader range of applications. UPB is fully programmable beyond the simple commands of X10.
- **KNX** appeared in Europe in the late 1990s and early 2000s and spread from there to over 100 countries. The system operates in much the same way as Insteon, except that in addition to power lines and RF frequencies, the standard system also supports the transmission of commands over wireless infrared, twisted pair wiring and Ethernet cables. KNX is normally installed in a twisted pair wiring setup, which effectively eliminates electrical interference.
- **ZigBee** was conceived in 1998, standardized in 2003, and revised in 2006. The name refers to the waggle dance of honey bees after their return to the beehive. ZigBee is an IEEE 802.15.4-based specification for a suite of high-level communication protocols used to create personal area networks with small, low-power digital radios. It is a type of wireless mesh network that is completely unlike any of its predecessors. In a mesh network, every device acts as a relay to send and receive information. Commands travel by relay through the network of devices until they reach their intended destination. Due to the nature of a mesh network's relay system, the wireless network can become larger, stronger, and more reliable with each additional device added. Its low power consumption limits transmission distances to 10–100 meters line-of-sight, depending on power output and environmental characteristics. ZigBee 3.0 is on the verge of being released at this writing.

- **Z-Wave** alliance was established in early 2005. This protocol uses the same mesh networking strategy as ZigBee; devices can communicate to one another by using intermediate nodes to actively route around and circumvent household obstacles or radio dead spots that might occur in the multipath environment of a house. It is oriented to the residential control and automation market and is intended to provide a simple and reliable method to wirelessly control devices.

3.3 Lighting Control

It is the most common usage scenario of a Home Automation system based on the omnipresence of Lighting. Lighting control is fairly easy to both explain and set up. The simplest example and its main components follow: [19]

- A hardware controller, or central control unit
- An actuator
- A lamp

The actuator in this case is a device that controls the flow of current from a wall socket to the lamp in question. It does so by being plugged into both the wall socket, and the lamp. The control unit communicates with the actuator to tell how much current to let through to the lamp. The control unit may be operated through a website, by remote control, or something similar.

The setup is illustrated in Figure 7. The wireless communication between the remote control, the control unit, and the actuator is implemented using a Home Automation communications protocol, e.g., ZigBee or Z-Wave (most common technologies).

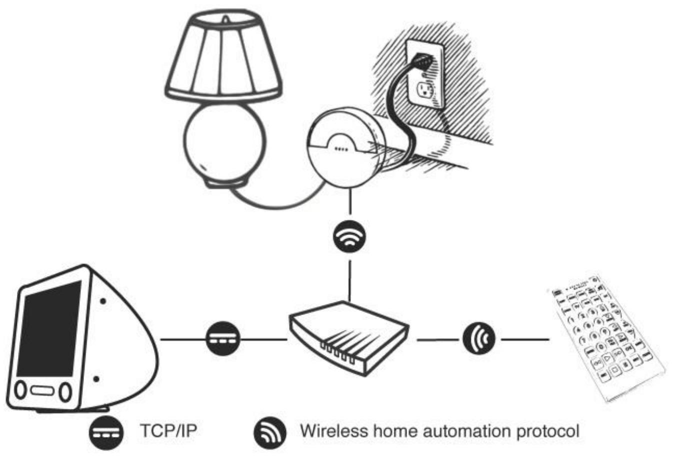


Figure 4 Simple Lighting Control Use Case [19]

4.Problem Analysis

In this chapter, we go deeper inside Lighting Control and we introduce Philips Hue, the low-level context of the problem at hand. By describing the constituents of the Hue ecosystem, we can finally focus on the specific module in question, the Lighting Engine and its existing implementation, the rule engine.

4.1 Context

In the previous chapter, we explained the high level structure of Philips Lighting in Business Groups and we mentioned that our focus is the Home Business Group and inside that the Home Systems Business Unit. This business unit is publicly known as Philips Hue.

This assignment mainly involves three people from Philips who are responsible for creating and supervising the assignment in question. The structure inside the Philips Hue department is as depicted on the following page.

The people who are responsible for this assignment are:

- **George Yianni,** Project Owner
- **Daniel Goergen**, Project Manager
- **Walter Slegers**, Project Mentor

Their roles and involvement are explained in the next chapter, Stakeholder Analysis.

Philips Hue is an internet connected (wireless) lighting system designed to transform how users experience light inside their homes. It is one of the leading and most installed smart home / Internet of Things products in the world. Philips Hue transforms how users can experience light by enabling color tunable lights to be controlled from smartphones, web services or other control logic and devices running in the system. Furthermore, it is an open system, i.e., via standardized or published interfaces other suppliers can add components.

Hue lamps communicate via a standardized ZigBee Light Link protocol allowing integration with ZigBee Light Link based devices such as sensors and light switches. Via a Hue bridge the ZigBee Light Link network is connected to the Internet. On the Internet side, there are smart phones, web browsers, third-party services (like IFTTT) and a Hue portal. All these components have software and use Hue interfaces or SDKs provided for Hue application development. [20]

4.2 Hue System

In 2012, Philips launched the Philips Hue system. It was a set of three intelligent lights that, along with the bridge connected to any home Wi-Fi system, allowed for color and brightness control of the lights from a smartphone or tablet. Since then a lot of things have been changed. The basic changes and components are described in the following sections.

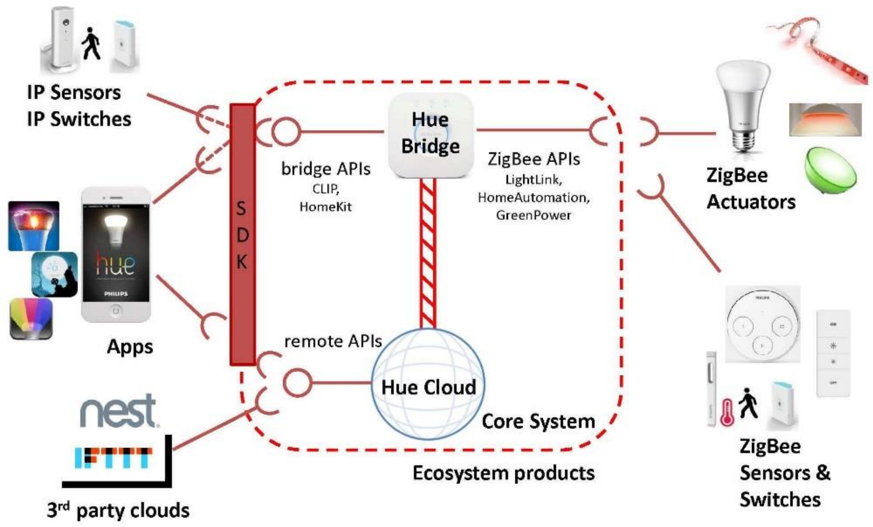


Figure 5 Hue system overview [21]

4.2.1. The Bridge

A Hue bridge is an interconnection between the ZigBee Light Link network and the local residential network. Near the end of 2015, a new bridge was introduced (v2) by Philips along with quite a few changes.

Within the Hue system, the bridge is responsible for Home Automation (e.g., lights go on when you operate a Hue switch, or lights go off at a specified time), for operation of the lights via IP (e.g., mobile applications or cloud services), for connection to Hue Cloud/Portal, for software updates and last but not least for data logging. The Hue lights and switches communicate via ZigBee, so the bridge interconnects between IP and ZigBee networks.

Figure 6 Hue Bridge v2 (rectangular) and v1 (circular) respectively

From a bird's-eye view, the Hue bridge v2.0 is a budget router enhanced with Apple HomeKit communication and ZigBee. The implications for software development are significant.

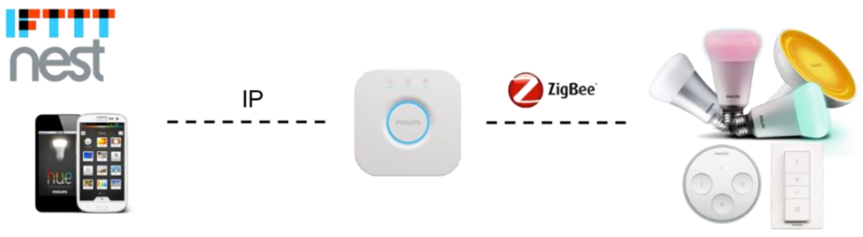


Figure 7 Hue Bridge as a gateway [22]

Hue system assumes there is only one bridge in a ZigBee Light Link network. The bridge acts as a gateway for Internet communication (forwarding commands and replies), acts as a programmable control center (execute commands based on schedules and rules), and distributes software updates on the ZigBee network. Via the Home Internet, it can also communicate via the Hue portal on the public Internet. The bridge offers a publicly documented interface via the local Home Internet for (third party) apps. **Externally it is called Hue-API**.

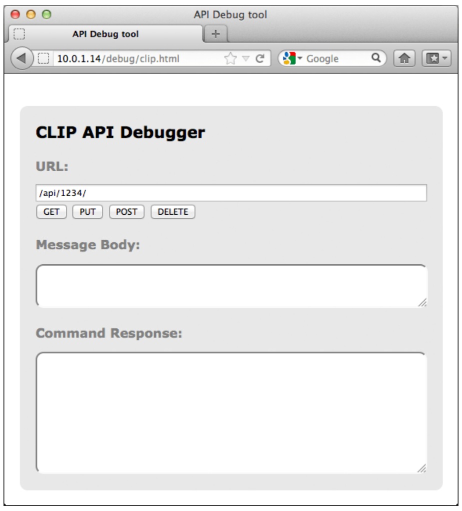
4.2.2. Hue API

The Hue API interface allows developers to interface with and make use of the functionality of the Philips Hue system. Using this interface, they can find information about the available devices in their local network, control these devices, and do much more.

The Hue API is a RESTful JSON interface in which clients interact with resources in the Philips Hue system. What this means is that every resource such as devices, groups and lights in the Philips Hue system is represented by a unique URI that is interacted with. This allows a user to control a resource (e.g., light) by sending a new value to the corresponding URL. [24]

Philips offers a simple test web app built into every bridge. Once we know the Hue bridge address we can load the test app by visiting the following address in our web browser.

- http://<bridge IP address>/debug/clip.html



CLIP API Debugger

URL:

/api/1234/

Message Body:

Command Response:

Figure 8 Testing tool – web application [25]

Using this debugger utility we can populate the components of an HTTP call – the basis of all web traffic and of the Hue RESTful interface. [25]

1. **URL**: this is actually the local address of a specific resource (thing) inside the Hue system. It could be light, a group of lights or many more things. This is the object we will be interacting with in this command.
2. **A body**: this is the part of the message which describes what we want to change and how. Here we enter, in JSON format, the resource name and value we would like to change/add.
3. **A method**: here we have a choice of the four HTTP methods the Hue call can use.
   - **GET**: this is the command to fetch all information about the addressed resource
   - **PUT**: this is the command to modify an addressed resource
   - **POST**: this is the command to create a new resource inside the addressed resource
   - **DELETE**: this is the command to deleted the addressed resource
4. **Response**: In this area we will see the response to our command. Also in JSON format.

4.2.3. Lights – Lamps

This is the output of the system. The lights or lamps are ZigBee Light Link nodes producing light in a range of colors and intensities. There are different kind of lights in the current Hue system.

Lamps form a ZigBee Light Link mesh network for communication. ZigBee Light Link is an open standard protocol, which means that other non-Hue nodes might be part of the network, like another light or a remote control changing the color or intensity of a lamp bypassing the Hue bridge. For the lamp, the Hue bridge is one of the ZigBee Light Link nodes. [24]

4.2.4. Apps

Apps are (smartphone) applications to control the lights via Hue bridge and portal interfaces. Multiple apps can control the same bridge and lights. Apps are not limited to the Philips Hue app, although the latter might offer more features by using interfaces that are not released to the public (e.g., Hue portal interfaces).

Apps can be used to configure Hue. In addition, they can be used for the bridge, among others, to add and remove lamps and sensors/switches. Furthermore, they can be used for the lamps, among others, to change light scheme definitions. Apps are not restricted to smart phones or tablets they could also equally be a website or an Arduino board. To support development of Apps on a smart phone, two SDKs are offered, an iOS and Android SDK. [24]

4.2.5. SDK

The Hue SDK (Software Development Kit) is a tool for third-parties to use to access the Hue system. It is provided in iOS and Android versions and builds a software layer in front of the Hue API. This software layer provides an object based interface in the native language (e.g., Objective-C) of the mobile device.

It is always possible to create apps purely using the Hue API. The aim of the Hue SDK is to hide system complexity, ensure compliance with technical requirements and make it easier to construct new Hue apps. [24]

4.2.6. Hue portal

The portal is a set of Hue Internet services running in the cloud. There is only one (distributed) Hue portal for all users, apps, and bridges. The portal offers access to a Hue bridge for a Hue app not connected via local network, a browser based “app” for access to one’s Hue system, and software distribution to Hue systems. The portal also offers services to stimulate Hue system usage such as light scene sharing, user FAQ, Hue blog, app developer information, marketing and sales information. The portal offers interfaces to trusted parties for access to Hue systems. [24]

4.2.7. Hue sensors and switches

Hue sensors and switches are ZigBee Light Link nodes providing their state to the bridge. State changes from Hue sensors and switches, such as a switch toggle, can be used to trigger actions, such as switching the lights on or off. A regular ZigBee Light Link sensor or switch can directly send a command to a light. In order to be integrated into a Hue system, the sensors and switches should be able to report their state to the Hue bridge allowing the bridge to decide what happens with the event. [24]

4.2.8. More ZLL nodes

More ZigBee Light Link compliant nodes exist that do not commercially explicitly target Hue. The ZigBee Light Link compliant nodes meant here are commercially not part of the Hue system. If it is a ZLL switch, it might directly change the state of a light independent of a Hue bridge. If it is a ZLL light, it could be controlled by a bridge but it might have less functionality than a Hue light. These nodes typically do not participate in Hue software updates. [24]

4.2.9. Third-party services

Third-party services refer to Hue aware cloud based functionality developed by others. A third-party service typically communicates with the Hue portal to get access to bridges in order to control lights or detect events in the Hue system. An example is IFTTT, which can be used to connect a Hue system to other Internet services. Currently, IFTTT and Nest are the only third-party services supported by Hue system. [24]

4.2.10. Browsers

This section is about the HTML browser on any device (PC, TV, tablet, mobile phone, and so on). Via a browser, one can access the user interface of the Hue portal. Users can see and control their light state; developers/maintainers can get access to debugging logs, and so on. If the browser is used via the local home network, it can also access the bridge. [24]

4.3 Problem Description

Now that we have a clearer view of the Hue system and its constituents, we need to focus on the Hue bridge. Inside the Hue bridge there are different modules.

The bridge software module that is responsible for all lighting automation functionality is the rule engine. In other words, the rule engine is currently the main way to add smart behavior in the Hue system. The rule engine was created a few years ago and served its purpose quite well until recently.

Lighting control as a part of Home Automation domain is an ever-changing environment. In 2016, the market needs for smart configurable behavior are dramatically different and still increasing in scope and complexity. **Those observations led Philips Lighting to start investigating lighting automation alternatives**.

The existing engine is quite efficient in memory and execution but it is at the same time inherently limited. There are known design limitations. These limitations were essentially inevitable because at the creation time, it was impossible to foresee the evolution of the lighting automation needs of the coming years

The term rule engine is quite ambiguous simply because it can be any system that uses rules, in any form that can be applied to data to produce outcomes. Next, we define what a rule engine is inside the context of Hue system using the presented functional view of the bridge as a basis.

4.3.1. Problem Statement in a nutshell

Facts

- The Philips Hue bridge controls and monitors ZigBee lights, sensors, and switches; it acts as local home lighting controller.
- The bridge uses a rule engine, which receives switch, sensor, or timer triggers and then sets a specific lighting scene as result.
- The capabilities range from switching on lights on a button press, to delayed switch off after sunrise, to small lighting state machines.

Goal

- ➢ With an increasing complexity of home lighting control use cases, the need to explore more sophisticated Lighting Automation Engines is becoming imperative.

The assignment includes

1. Investigate alternatives for the currently used rule engine for lighting control,
2. Prototype one or more alternatives on the Hue bridge and suggest one. Options can be found in scripting or other rule based systems.

Challenges

An important criterion is being able to **explain to a user of the system why a light changed**.

Other evaluation criteria include:

- Capabilities of the solution
- Complexity for developers/users using the solution
- Effort and fit with current Hue bridge hardware and software

4.4 Design Opportunities

After the end of the initial phase of the project, four alternative routes were identified as part of the follow up investigation and design phases. These four routes are more or less the possible approaches to tackle the challenge at hand. Some preliminary observations and accompanying comments are presented as follows:

1. **Improve the current engine**
   - People in Philips Lighting have already been working on that for quite some time. This group of people is dynamic and is called Feature Team(s). On the one hand, it is acknowledged as inefficient and (time & resource) expensive to continuously improve the current rule engine which is rather limited by design. From the research and development point of view, it seems complicated and not such a good trade-off to keep the same simple design and architecture and try to follow the ever-changing environment of Home (Lighting) Automation solutions. On the other hand, compatibility is an advantage for the continuous support and enhancement of the current solution.

2. **Create a new one from scratch**
   - It seems to be the ideal solution in terms of following faithfully the requirements but it is at the same time very time consuming and risky (for the scope of this project).

3. **Find available alternatives online and use an off-the-shelf solution or modify it accordingly**
   - Searching for the existing related works and picking the best fit for this specific case seems the best solution at first glance time-wise and implementation-wise.

4. **Hybrid solution**
   - In practice, things are rarely black and white; in that sense, the final solution can be a combination of the above. The existing engine can play the role of an interface to something new. Another example approach can be a mix of enhancing the current rule engine with a subset of an off-the-shelf solution.

5.Feasibility Analysis

In this chapter, the feasibility analysis of the problem in question is presented. The issues and the risks identified in the early stages of the project are listed below together with some brief explanation. Furthermore, a supportive table is depicted combining the hierarchy of the risks and challenges (i.e., the impact) with some mitigation strategies.

5.1 Challenges & Risks

Risks are ubiquitous in any domain or discipline. A crucial part of every project is to identify them as early as possible and try to mitigate their impact. Even from the initial project description, one could identify some possible challenges and risks and of course, many more were expected. Our effort in this section is focused on listing the most important risks and challenges followed by a short explanation. On the one hand, the goal is to give enough context for the reader to follow and on the other hand, to use this as future improvement reference.

5.1.1. Lack of domain knowledge

The domain of the project was not obvious from the original description; only after the first interview, it was more or less clear that embedded world knowledge would be needed. The author had only basic academic knowledge about the specific field but no real working experience. That was the main reason why extra effort was needed at the early stages of the project in order to acquire the sufficient level of understanding to proceed in designing a prototype. During that initial research phase, many meetings were organized and domain experts were involved to help in this direction. That process continued until the end of the project in a less intense manner.

5.1.2. Distributed expert knowledge

As already explained due to the lack of author's domain knowledge, the need for domain experts was imperative from the beginning. Many brainstorming meetings with various people from different departments were organized in order to acquire useful information and tips. The focus of the project was a module inside the main embedded device (i.e. the bridge) but in order to understand how the module works one should have sufficient understanding of the whole system. The knowledge about the specific module was distributed amongst many people and even more people were involved in the embedded device as a whole. Some of them were not even working for Philips Lighting at the time of research and some were working for different modules during the years. All in all, it was quite difficult to gather all the information needed from all these people and merge it in one non-contradictory, useful knowledge stream.

5.1.3. Complexity of legacy system/code

The main system has different versions and different people from different backgrounds were involved in its development and construction. The legacy module (i.e., rule engine) was also not new and relatively complex. The complexity of the whole system lengthened the author's learning curve and consequently the complexity of the important sub-module. The main focus was to identify what the current rule engine could do and what it could not do and at the same time to find the right places to put changes. Due to the size of the code-base and the lack of documentation, this identification process was quite long.

5.1.4. Lack of documentation

As already mentioned the technical complexity of the system was based on its size but also on the lack of supportive documents. There were different repositories and very few documents. For a new developer walkthrough guides, architectural documents/diagrams, and configuration and installation manuals can be invaluable especially as a starting point. Unfortunately, the existing documentation was scarce and outdated in many cases.

5.1.5. Lack of resources

The deployment and installation of the development environment was delayed based on two reasons. First was the lack of proper stepwise documentation and contradictory expert advice (different people were using different set-ups) and second was the lack of hardware resources. To be more specific, originally the laptop that was provided from the company was intended for managerial use and was by no means capable to support software development. The process of applying for new equipment was quite long and caused an almost inevitable bottleneck.

Furthermore, in different stages of the project, different extra resources were needed, such as a router, a switch, a debug cable and an extra bridge but the process of getting them was not always straightforward.

Last but not least, sometimes lack of resources was combined with issues of malfunctioning or wrong equipment. One of the best examples was that the bridge given to the author was meant for production usage and not for development (which was not communicated). On top of that, at some point in time, someone changed the IP part of the bridge to development version and the ZigBee part remained as production version. Consequently, a lot of inconsistencies and issues were caused by that problematic combination until the point of actual realization and resolution.

5.1.6. Time shortage – Converging vs. Diverging

The nominal time span of the project was nine months. The actual working time was much less if one would consider extra time such as holidays, meetings and come-back days. The very purpose of those projects is to build a prototype beginning from an ill-defined problem. The diverging vs. converging trade-off or in other words implementation vs. research was almost inherent. We cannot do the one without the other and for sure we cannot do both perfectly. The golden ratio is project-specific and requires extensive and iterative deliberation with all involved parties.

5.1.7. Conflicting requirements and use-cases

The author participated in more than 20 meetings during the process of requirements elicitation. The document that was produced (including mainly non-functional, functional requirements and use-cases) was quite extensive resulting in an increased risk of conflicting requirements and user-stories. Prioritization had to be made and that procedure was not straightforward introducing an obvious time overhead.

5.1.8. Contradictory user categories

Besides the conflicts introduced generally by the large list of stakeholders there was another major source of contradictions. The prototype in question had basically two main user-categories, the developers and the end-users. Those two categories introduce many design trade-offs. The requirements and the use-cases that are connected with each category are quite different and in many cases conflicting, introducing **trade-offs**. For example, the developer needs a flexible Lighting Engine but this flexibility inevitably introduces complexity for the end user. This complexity conflicts with the need of (end) user-friendliness and so on and so forth.

5.1.9. Dependency – Configuration issues

One of the biggest challenges in this specific project were the dependencies of the module in question combined with the configuration issues. As already explained, the Lighting Automation Engine is merely a small part of a bigger system and in order to build something similar (on top, in parallel, or in any way) one should understand basic things for the system as a whole. Additionally, one should take care of all the dependencies of the specific module with other modules that it communicates with. Some of them should be modified or notified when something new was introduced.

Furthermore, those dependencies were accompanied with configuration issues. In order to build even small changes, one should re-build the entire system, manage all the dependencies, and make sure that everything is linked in the right way. A lot of things can go wrong and lack of modularity was making things even worse. Rather simple tasks like memory usage measurement, addition of external libraries, and use of older repository branches were introducing errors and thus experts were needed.

All in all, configuration issues took approximately one fourth of the total time of the project and were intertwined with many if not all the above mentioned risks and challenges.

5.2 *Risk Management*

The above mentioned risks & challenges are gathered in one table in combination with their potential impact, probability and mitigation strategy.

Table 1 Risk Management

Risk - Challenge	Impact	Probability	Mitigation Strategy
Lack of domain knowledge	High	High	Read documentation, consult experts
Distributed expert knowledge	Medium	High	Arrange meetings, keep minutes
Complexity of legacy system/code	High	High	Start implementation early, gather questions, involve experts
Lack of documentation	High	High	Combine documents, create my own, pair programming
Lack of resources	Medium	High	Involve project owner, be persistent, use workarounds, backup solutions
Time shortage – Converging vs. Diverging	High	High	Involve supervisors in decisions and make a good time management plan
Conflicting requirements and use-cases	Medium	High	Stakeholder analysis, prioritize inputs and use hierarchical scale, involve company supervisors at all stages
Contradictory user - categories	High	High	Create abstraction layers for different user categories exposing different parts of the solution.
Dependency – Configuration issues	High	High	Pole experts, expose to supervisors, create own documentation for future reference

6.System Requirements

After the analysis of the domain and its problems, a set of requirements is extracted and formulated, that have to be satisfied for this project. This chapter presents these requirements, both functional and non-functional accompanied with relevant use cases.

6.1 Introduction

6.1.1. Purpose and scope

This section provides some concepts, which gradually will evolve into requirements combined with some use cases which the future proposed Home Automation Lighting Engine should comply with.

- The part of the legacy system that we aim to improve is called rule engine.
- The rule engine is a software component intended to translate sensor events into actions in the Hue system.
- This component is event driven and events are modeled as sensors events.

6.1.2. Requirement Elicitation Process

In the beginning of the project, a list of people that would be involved had to be established. In order to create this list, some white board meetings were organized with the help of the Philips Lighting supervisors. People who had any kind of interest or relation with the current Lighting Engine (i.e., rule engine) were invited to discuss their ideas about its past, present and possible future.

The goal was to have as many people as possible and eventually narrow the list down to the most important stakeholders who could provide the best contribution. After the first brainstorming sessions, a small list of concepts (requirements) was composed as the first draft and presented below. The lists of affiliated and stakeholders were presented in previous section.

In the context of the requirement elicitation phase, multiple meetings with the affiliates were organized. Preferably, small group meetings took place in order to gather as much information as possible in the area of expertise of each person. The goals of these meetings were to:

- Create a first version of a requirement list that could be reviewed by all affiliates and lead to further refinement and discussion.
- Increase the familiarity between the author and the people. This would lead to more information sources and assistance during the project and also higher chances of adoption of the final solution.
- Establish a better understanding of the context in which the engine is used from different points of view.

After this first round of meetings, a concept (requirement) list was created. This list included most of the concepts (requirements) that were mentioned during the meetings. The main aim of this phase was to create a concept list that would eventually lead to a requirement list that would satisfy the most important stakeholders and allow the author to move forward to the design phase of the project. The list was not meant to be final but relatively stable, since the research nature and duration of this project were expected to cause changes. The non-functional requirements were given more emphasis during this phase. Most of the functional requirements were refined later during the iterative prototype phase (creation of the specification).

6.2 Requirements

6.2.1. Non-Functional

- Understandability (end user)

The engine shall provide the proper metadata, explaining why something changed, in order to facilitate app monitoring from the end user perspective.

- Traceability (developer)

The engine shall provide functionality on the bridge such that mechanisms will be available for the UI (User Interface) developers to create (block-view) control and monitoring features.

- Compatibility

The engine shall at least support all basic functionality (CRUD, etc.) of Hue interface.

- Modularity

The lighting engine should be comprised of decoupled units with proper interfaces.
It shall facilitate creating new individual modules and linking them together.

6.2.2. Functional

The following list is presented in hierarchical order. The hierarchy is not strict but it reflects, in a way, the importance to include this specific functionality in the future Lighting Automation Engine and its specification. Most of these requirements were inspired by use cases which are described afterwards but formulated in advance.

1. The engine should be event-driven (included as a solution in Appendix A)
2. The engine should be able to create templates and reuse behavior. Behavior that is commonly used (e.g., scene cycling, toggling, dim up/down) should be offered in a generic format to other users.
3. It shall provide means to configure behavior. This is related with the above. The behavior offered should not be static but to include variables and other means of configurability.
4. It should offer different abstraction layers for different user categories. In other words, different behavior should be exposed to different users. The end user, the third-party app developer, the external script developer and the Philips developer are some basic examples.
5. The engine shall provide the means to identify behavior by gathering metadata (owner (app, user), unique identifier, name, description, timestamp, etc.)
6. The engine shall be able to enable/disable behavior.

6.3 Use Cases

A first attempt to list user scenarios and needs for lighting automation is presented as follows. The use cases were grouped in various categories. The first grouping level includes three different user perspectives the end-user perspective, the developer perspective and the Hue system perspective. The second level is applied when needed and it is grouping per lighting control topic. The current list consists of two examples for each group.

The Requirements document is a separate document accompanied with related design questions which should be answered during the second and third phase of the project or act as a reference for future implementers. The diagram 19 reflects the grouping and the different perspectives. The # symbol is used to signify a group (multiple) of requirements instead of a single one.

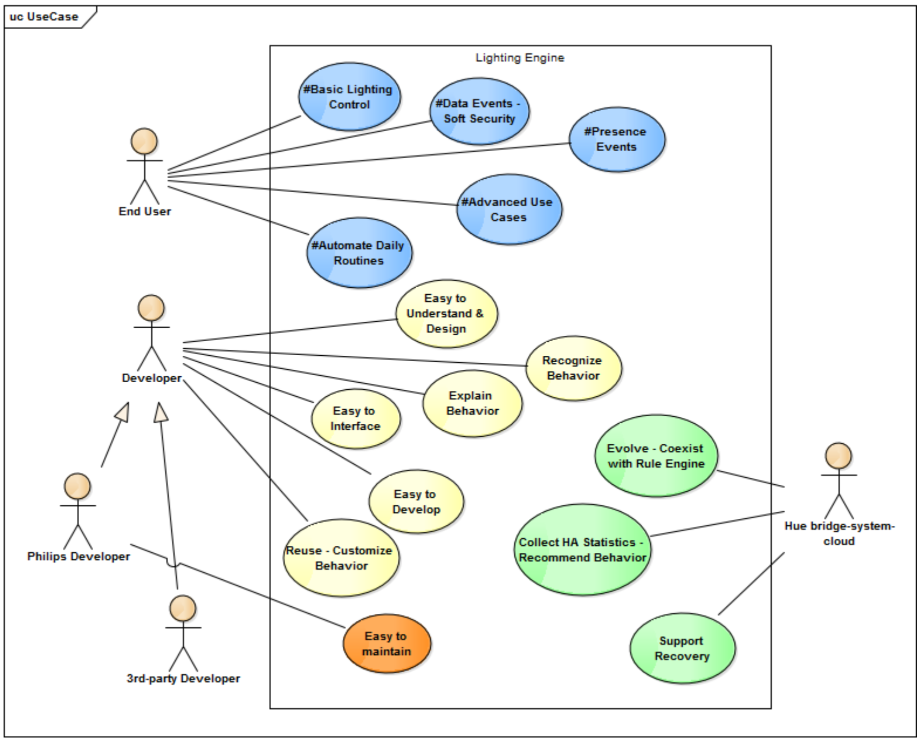


Figure 9 High level Use Case diagram

6.3.1. User perspective

Basic Lighting Control

1. As a user I would like a switch on the ground floor and a switch on the 1st floor both toggle (on/off) the light on the 1st floor (e.g., Hotel switch).
2. As a user I would like by pressing a button once to browse between different scenes or light recipes (relax, energize, etc.), and by pressing it again after x (=10) seconds to switch off.

Automate daily routines

1. As a user I don't like to be woken up during the night so I like to automatically get a more dimmed light when I switch on a light during that time.
2. As a user I want on weekdays the bedroom light to gradually dim up at 7:00 am, in the weekend not.

Data events – Soft Security

1. As a user I want to have a light bulb near the umbrella appear as blue light in the morning when it might rain (Internet weather service).
2. As a user I want to be notified if door/window is open during night

Presence Events

1. As a user when I am approaching home (geolocation by phone) I want the light in the hallway to switch on between sunset and sunrise.
2. As a user I want a garage presence sensor to trigger the garage lights to go on when I am there and after my departure fade them out in 5 min.

Advanced Use Cases

1. As a user I want lighting control to include gradual dynamic effects.
2. As a user I want the lights in the house to follow the circadian rhythm.

6.3.2. Mobile app / Internet service developer perspective

1. As a developer I want to have an engine easy to understand (15 min of reading) and easy to design the first home automation app for it. [to increase adoption rate]
2. As a developer I want to have the means to explain to a user which behavior is/has installed (in which room, for which lights, etc.), also if a second instance of the app is running on a second phone.

6.3.3. Hue (bridge/cloud/system) perspective

1. As a system I want the new engine to (be able to) become an evolution (compatible with previous generation) or coexist nicely in parallel to the incumbent lighting automation solution.
2. As a system I want an engine that could facilitate the support of a (Hue) app that can “recover”, e.g., disable behavior installed by buggy app to help users recover.

6.4 Design Criteria

TU/e provides a list of criteria as a measure of assessment of a technological design.

6.4.1. Introduction

Design criteria can be used for many different purposes. First, they can be used to distinguish valid design assignments from invalid ones. This is important in selecting and formulating design assignments – prior to their actual execution. One crucial aspect is, that there should be a clearly identifiable artefact that is to be designed or redesigned. Second, the criteria can be used in assessing the design assignment after completion. As an interesting side-effect, criteria will help shape the actual realization of design assignments since, during the design process, both the designer and the supervisors will aim for high scores in the upcoming assessment.

First and foremost we want to define what it is that is being designed. This we call an *artefact*. An artefact can be either a product or a process that brings forward products. A product can be a physical product, but also a software product or even a service.

After deliberation with both Philips and TU/e parties a choice of criteria was made. Three that apply and two that do not apply were picked in order to reflect on them during different stages of the project. The following aspects of the artefact in question (i.e., current project) need consideration:

6.4.2. Criteria that apply

The following criteria were carefully picked during the 2nd month of the project because they were considered relevant at that point in time. In the analysis paragraph we will explain why. [28]

- **Functionality:** Which are the functions to be fulfilled by the artefact, and how effective shall it be? Most often, these requirements are initially vague; the role of the artefact in its context is usually described in a merely global way. Together this forms the set of requirements. To a large extent, the designer determines the functionality in the form of specifications, staying within the envelope determined by the requirements. In case of a re-design, finding the part that needs adjustment may form the main challenge.
- **Complexity:** Designing a complex artefact requires the knowledge of methods and techniques from various disciplines. A truly complex artefact, in general, will be only realizable by a design team.
- **Documentation and presentation:** Is the description of the artefact sufficient to check that the design has been carried out according to the rules; are the models sufficient to demonstrate essential features of the artefact?

6.4.3. Criteria that do not apply

The following criteria were carefully picked during the 2nd month of the project because they were considered irrelevant at that point in time. In the analysis paragraph we will explain why. [28]

- **Impact:** What is the economical and societal relevance of the artefact? Which revenues are expected, and for whom? What is the societal purpose of the artefact? Which risks are implied by the production, use and disassembly of the artefact? In what respect does the artefact contribute to sustainable society?
- **Inventiveness:** To what extent is the solution novel? 'Novel' may mean the deployment of a novel technology, or an innovative combination of existing technologies. In both cases, there can be the case of a creative invention; it can also be a trivial compilation of existing elements. Inventiveness is therefore partially determined by the complexity of the artefact.

6.4.4. Criteria Analysis

Functionality

The functions to be fulfilled by the artefact are described in the Requirements document. That document is quite extensive so we have extracted some generic topics with pivotal role in the context of functionality. These topics are groups of use-cases and can be named as pillars:

1) Basic Lighting Control
2) Automate daily routines
3) Data events
4) Presence events
5) Advanced (Dynamic) functionality

In addition, the various specifications of the artefact can be considered part of the functionality. For example, we have already defined certain RAM & Flash memory specifications-limitations presented (in detail) in the technology choice memory requirements document.

Complexity

The artefact in question (lighting engine) is a very small part of a big system (Hue bridge). The complexity of the entire system affects the complexity of its sub-modules. In order to build something that is meant to be a natural part of the hue eco-system and in particular become a sub-module of the hue bridge you have to have at least a vague idea of how things work from inside. The collaboration-communication of the artefact with the rest of the system even in the ideal scenario of complete modularity is imperative.

Furthermore, the artefact itself should be considered complex based on the domain knowledge needed. The embedded world requires experience and a lot of patience in order to achieve tasks that in the "normal pc world" are considered easy. Expertise is scarce and multi-disciplinary, methods, techniques and requirements from different departments should be combined towards creating the final product. The existing system (current rule engine) was created by a team and is currently considered limited, the new engine is theoretically one's man project thus the outcome will be a prototype. Even though there is no team for the current project the goal is to improve the already present functionality and that is increasing the complexity drastically.

Documentation

All the participants of this project agreed on the importance of documentation. Both parties, TU/e and Philips have their own reasons for supporting the detailed documentation as one of the most crucial outputs.

- TU/e requires a thesis; this thesis should be focused on the design of the artefact. This design document should be backed up by sufficient number of models demonstrating essential features of that artefact.
- Philips Lighting is investing on a prototype project that will be future-proof. In other words the company values an orthodox step-wise development process following the entire software development cycle. Every step should be well documented and reasoned in order to be available for future usage. Every design decision, every requirement, every architectural or technological choice matters.

Documentation is usually underestimated in big organizations comparing to functionality. In our case a well-documented artefact that is not perfect is preferred than a poor-documented artefact that is fully functional especially considering the fact that (updated) documentation is truly scarce in our department.

Impact

This project is considered as a prototype and as such belongs to the pre-development team of Hue department. Consequently, it is not difficult to concur that there is no immediate societal or economical relevance. It is currently undefined whether the artefact will be put to production or used just an example for future development. There is no obvious business value for realizing this specific artefact from the marketing department's point of view.

Inventiveness

Building something entirely new or from scratch cannot be considered a design goal. It could be a welcomed result but that is not the goal. The original goal of the current project was to re-use existing knowledge (preferably off-the-shelf) modifying it to Philips Lighting needs and ultimately build on top of that. Generally, innovation is valued but it is not the top priority in the artefact design level. In that sense the solution cannot be considered novel per se.

7. Design Alternatives

In this chapter, some important design decisions are discussed. These decisions could serve as the foundations of the new engine's architecture. Furthermore, information for other relevant existing systems as well as the implementation language decision can be found.

7.1 Introduction

Early in the project, the design opportunities and the context were defined (section 4.4 Design Opportunities). The main goal was to investigate and redesign the Hue Lighting Automation Engine given the freedom to think outside the box. This gave the project an exploratory nature and a broad range of choices. A large number of alternatives would have to be investigated and few most suitable features kept in the final solution as inspirations.

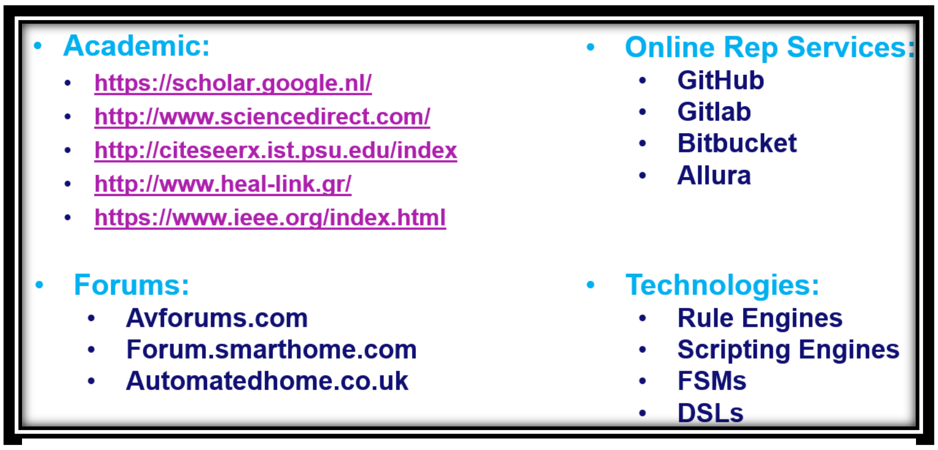
The original direction was to investigate open-source home automation systems or engines that could ideally be modified to our needs.

The process of evaluating alternatives and deciding in favor of few was carried out in two phases. The first phase was more theoretical and included documentation reading, meetings with relevant stakeholders and thinking of applicability of similar solutions from other domains. The second phase of evaluation was the creation of a large scale comparison table based on a group of important criteria. The created table would have some earlier decisions as foundations and incorporate more during the process. Some alternatives were dependent on others, so that pointed the order of the investigation.

In this chapter, the decisions that acted as the foundations for the final direction are described. This way the reader can have a better impression and understanding about the rationale behind choices in the final design.

7.2 First Phase

A plethora of information sources were identified and presented in the following picture. Home Automation is an evolving domain and as such there were many solutions provided. The open-source software requirement ruled out many of the alternatives but still the list was quite extensive.



- Academic:
  - https://scholar.google.nl/
  - http://www.sciencedirect.com/
  - http://citeseerx.ist.psu.edu/index
  - http://www.heal-link.gr/
  - https://www.ieee.org/index.html
- Online Rep Services:
  - GitHub
  - Gitlab
  - Bitbucket
  - Allura
- Forums:
  - Avforums.com
  - Forum.smarthome.com
  - Automatedhome.co.uk
- Technologies:
  - Rule Engines
  - Scripting Engines
  - FSMs
  - DSLs

Figure 10 Information sources for alternative technologies

A quite extensive research phase was initiated and all these sources were used with different combinations of key words and phrases (e.g., rule engine, inference engine, automation engine, scripting engine, lighting engine, finite state machine (FSM), and domain specific language). Parts of the sources were also online communities (forums) and individuals (independent developers) which were asked to provide ideas and suggestions for possible solutions.

Furthermore, personal research was accompanied with various meetings with domain experts inside Philips Lighting and thus became more focused. The first draft list of alternatives is shown below (more information can be found in separate document).

7.2.1. OpenHAB

The Open Home Automation Bus (OpenHAB) project aims at providing a universal integration platform for all things around home automation. It is a pure Java solution, fully based on OSGi[55]. As mentioned in the section 3.2.1, OpenHAB's automation engine is only a small part of an entire home automation system and its focus is far beyond lighting.

Through an extensive email discussion with the founder of the project (Kai Kreuzer) it is understood that at the time of writing OpenHAB is part of Eclipse SmartHome framework and moves towards a second version.

OpenHAB version 1 included a rule engine, using a domain specific language based on java, but had various constraints:

- It required EMF/Xtext/Xbase on the runtime, which is not ideal for constrained devices
- It did not allow to build rule GUIs on top as all rules are purely textually defined.
- It did not allow to reuse and share rules or logic blocks between users / solutions.

They therefore introduced a new rule concept into Eclipse SmartHome (OpenHAB version 2). The overall idea was to have reusable rule components that are formally described, so that they can be handled in GUIs and combined into rules. These had to be provided as a kind of template, that could easily be shared and instantiated by a user through a UI. That very idea served as inspiration for our proposed solution. [29]

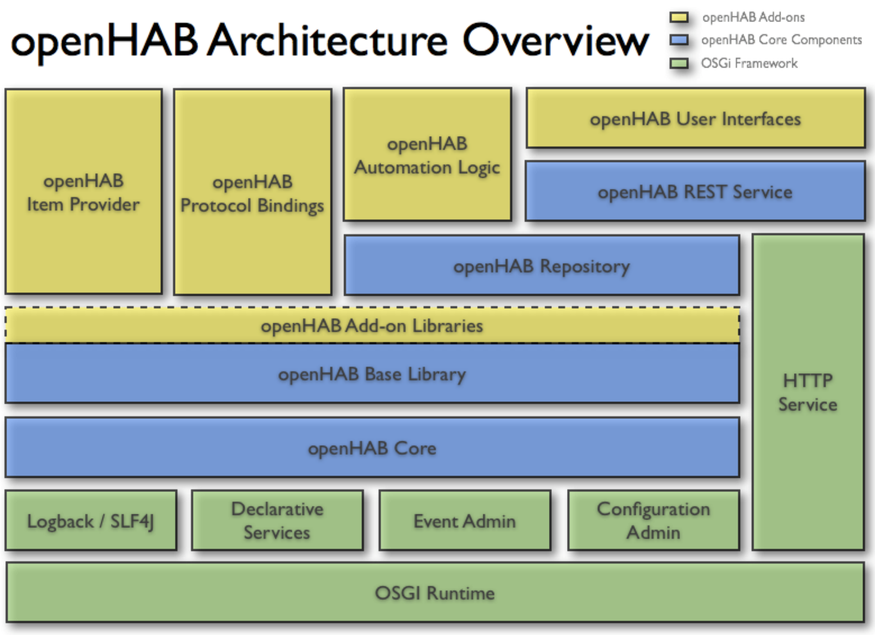


Figure 11 OpenHAB architecture overview – automation logic module [29]

7.2.2. OpenRemote

OpenRemote is a software integration platform for residential and commercial building automation. In addition to the open source variant there is also a commercial version of OpenRemote available.

The OpenRemote platform consists of three software components:

- The **OpenRemote controller**, an always-on (24/7) Linux, Windows or OS X server application, which connects the mobile control devices (smartphones, tablets) to building automation systems and devices under control. Control devices can be building infrastructure (light switches, power outlets etc.), consumer electronic devices, or home appliances. The OpenRemote controller can also run scripts, which are called rules. These rules are automation sequences, which are implemented based on the open Drools event processing language.
- The second component consists of the **OpenRemote mobile clients** (OpenRemote Panels) for iOS or Android. Graphical user interface and functionality of these apps can be fully customized using the third component of OpenRemote, the OpenRemote Designer.
- **OpenRemote Designer** is an online, cloud based application, providing a graphical user interface for crafting the mobile client interface and the related commands, sensors, and switches. Once user interface and control functions are designed, the OpenRemote Designer configuration files are synchronized with the local controller installation. The smartphone client application is updated automatically, when connecting to the controller, immediately reflecting changes or updates made in the OpenRemote Designer project.

In the context of this system, an extensive way of describing behavior can be done using Rules (system module). This is very useful for time-based actions, such as switching on and off lights while on vacation; for activity-based actions reacting to the presence of people detected by sensors; or for a combination of those such as smart thermostats. The rules make use of the Drools language. Since Drools is also an alternative we will continue the analysis there. [30]

7.2.3. Vera – OpenLuup

Luup (Lua-UPnP) is Mi Casa Verde’s software engine which incorporates Lua, a popular scripting language, and UPnP, the industry standard way to control devices.

Mi Casa Verde's core product, **Vera**, is a complete (proprietary) home automation solution with a focus on energy conservation, yet powerful and flexible for the smarthome enthusiast. Vera is built with Luup, running on a modified Wi-Fi access point.

On top of Lua, Vera has developed an extension they have called Luup. Luup adds functionality to Lua to allow users to interact with Vera.

OpenLuup is a pure-Lua **open-source** emulation of the Vera Luup environment. **OpenLuup** is an environment which supports the running of some MiOS (Vera) plugins on generic UNIX systems (or, indeed, Windows systems.) Processors such as Raspberry Pi and BeagleBone Black are ideal for running this environment, although it can also run on Apple Mac, Microsoft Windows PCs, anything, in fact, which can run Lua code (most things can - even an Arduino Yún board.) The intention is to offload processing (CPU and memory use) from a running Vera to a remote machine to increase system reliability. [31]

7.2.4. Domoticz

Domoticz is an open source Home Automation system that lets users monitor and configure various devices like: Lights, Switches, various sensors/meters like Temperature, Rain, Wind, UV, Electra, Gas, Water and much more. Notifications/Alerts can be sent to any mobile device. It is written in C++.

This system is designed to operate in various operating systems (Linux/Windows/Embedded Devices). The user-interface is a scalable HTML5 web frontend, and is automatically adapted for Desktop and Mobile Devices. It is compatible with all recent browsers.

Domoticz helps users to add some intelligence to their home. A great deal of this functionality can be reached with the available program options like timers and notifications. When things get more complicated scripting can be the solution. In Domoticz two kinds of scripts are commonly used: **Lua scripts** and Bash shell scripts. The Lua interpreter is integrated by the Domoticz developers (so is also available in Domoticz on Windows) and the interpreter for the Bash shell scripts is built in to the Linux OS.

Domoticz provides one of the most detailed Lua implementation documentations.

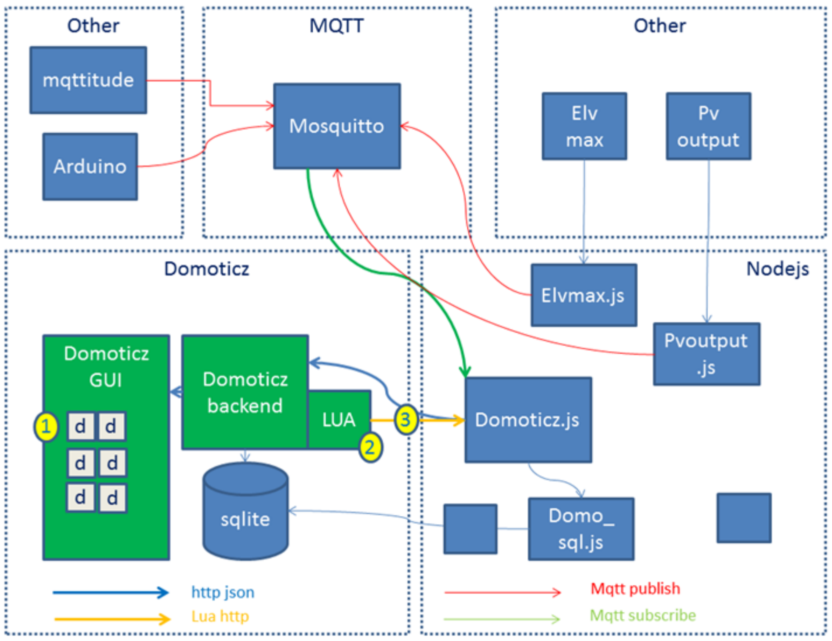


Figure 12 Domoticz – Architecture Overview [32]

- **MQTT** is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. MQTT provides a publish/subscribe message pattern to provide one-to-many message distribution and decoupling of applications.
- **Node.js** is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- **Node-RED** is a tool running in the Node.js platform providing a browser-based flow editor that makes it easy to wire together "flows". Flows can be then deployed to the runtime in a single-click.

Domoticz supports a number of hardware devices natively (e.g., rfxtrx433, z-wave, smartmeter). Using its native MQTT interface Domoticz can **publish** events from inside to the outside world. Domoticz can also respond to actions requested by anyone (and passed on by the MQTT-broker).

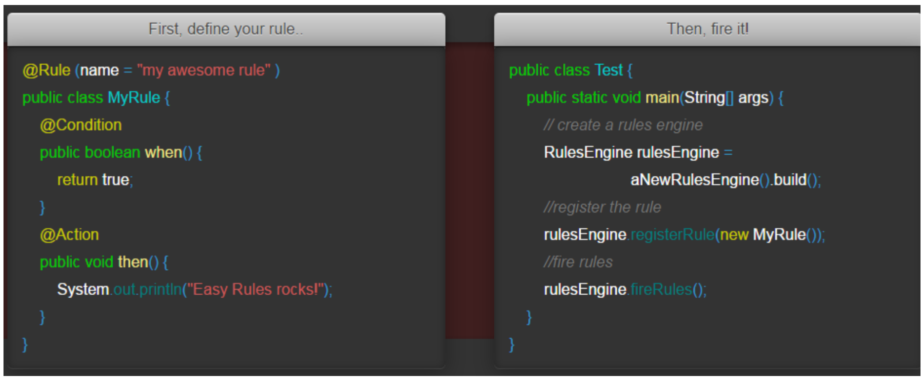
The Node-RED tool provides an alternative way to creating little programs (flows) to interface with anything user wants. But maybe the user application can create MQTT-messages on its own that can be understood by Domoticz. Or maybe user likes to create programs in Node.js itself. So using Node-RED is not mandatory, it does however provide an attractive way to handle messages. An alternative of using MQTT is **the internal LUA-engine.** [32]

7.2.5. Easy Rules

Easy Rules is a Java rules engine inspired by an article called "Should I use a Rules Engine?" by Martin Fowler in which he says:
“You can build a simple rules engine yourself. All you need is to create a bunch of objects with conditions and actions, store them in a collection, and run through them to evaluate the conditions and execute the actions.”
This is exactly what Easy Rules does, it provides the Rule abstraction to create rules with conditions and actions, and the Rules Engine API that runs through a set of rules to evaluate conditions and execute actions.

Core features:

- Lightweight library and easy to learn API (20k jar)
- Embeddable (in an application server, a servlet container or a dependency injection container)
- POJO based development with annotation programming model
- Useful abstractions to define business rules and apply them easily with Java
- The ability to create composite rules from primitive ones
- Dynamic rule configuration at runtime using JMX



First, define your rule..

```
@Rule (name = "my awesome rule" )
public class MyRule {
  @Condition
  public boolean when() {
    return true;
  }
  @Action
  public void then() {
    System.out.println("Easy Rules rocks!");
  }
}
```

Then, fire it!

```
public class Test {
  public static void main(String[] args) {
    // create a rules engine
    RulesEngine rulesEngine =
               aNewRulesEngine().build();
    //register the rule
    rulesEngine.registerRule(new MyRule());
    //fire rules
    rulesEngine.fireRules();
  }
}
```

Figure 13 Example structure of Easy Rules Engine [33]

Why is Easy Rules called the "The stupid Java rules engine"?

The goal behind Easy Rules is to provide a lightweight rules engine without features that 80% of application do not need. The term "stupid" is actually the perfect term to describe how the engine works: It iterates over a set of ordered rules and execute rules when their conditions are met. This what makes it easy to learn use and use following the KISS (Keep It Simple, Stupid) principle. [33]

7.2.6. Home Assistant

Home Assistant is an open-source home automation platform running on Python 3. The goal of Home Assistant is to be able to track and control all devices at home and offer a platform for automating control. [17]

The main parts of the system:

- **Home Control** is responsible for collecting information on- and controlling devices.
- **Home Automation** triggers commands based on user configurations.
- **Smart Home** triggers commands based on previous behavior.

In order to better understand how this system works it would be nice to see the bigger picture of Home Automation landscape and how Home Assistant fits in it.

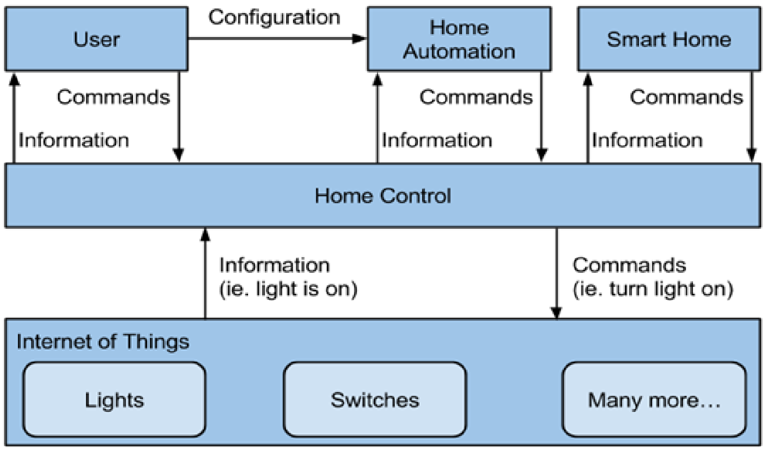


Figure 14 Overview of the Home Automation landscape [17]

The Home Assistant core is responsible for Home Control. It has four parts to make this possible:

- The Event Bus facilitates the firing and listening of events. This is the beating heart of Home Assistant.
- The State Machine keeps track of the states of things. Fires a state_changed event when a state has been changed.
- The Service Registry listens on the event bus for call_service events and allows other code to register services.
- The Timer will send a time_changed event every 1 second on the event bus.

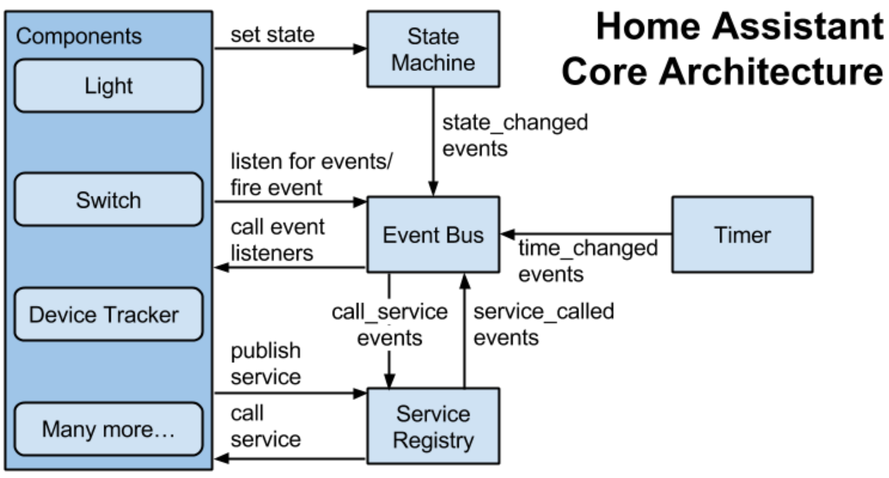


Figure 15 Home Assistant core architecture [17]

Home Assistant offers a few built-in automations but mainly you'll be using the automation component to set up your own rules.
The basics of automation: [17]

- Every automation rule consists of triggers, an action to be performed and optional conditions.
- Triggers can be anything observed in Home Assistant. For example, it can be a certain point in time or a person coming home, which can be observed by the state changing from not_home to home.
- Actions will call services within Home Assistant. For example, turn a light on, set the temperature on your thermostat or activate a scene.
- Conditions are used to prevent actions from firing unless certain conditions are met. For example, it is possible to only turn on the light if someone comes home and it is after a certain point in time.
- The difference between a condition and a trigger can be confusing. The difference is that the trigger looks at the event that is happening, e.g., a car engine turning on. Conditions looks at the current state of the system, e.g., is the car engine on.

7.2.7. Drools

Drools is an open source Business Rules Management System (BRMS) solution. It provides a core Business Rules Engine (BRE), a web authoring and rules management application (Drools Workbench) and an Eclipse IDE plugin for core development.

Drools is a BRMS with a forward and backward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm [34].

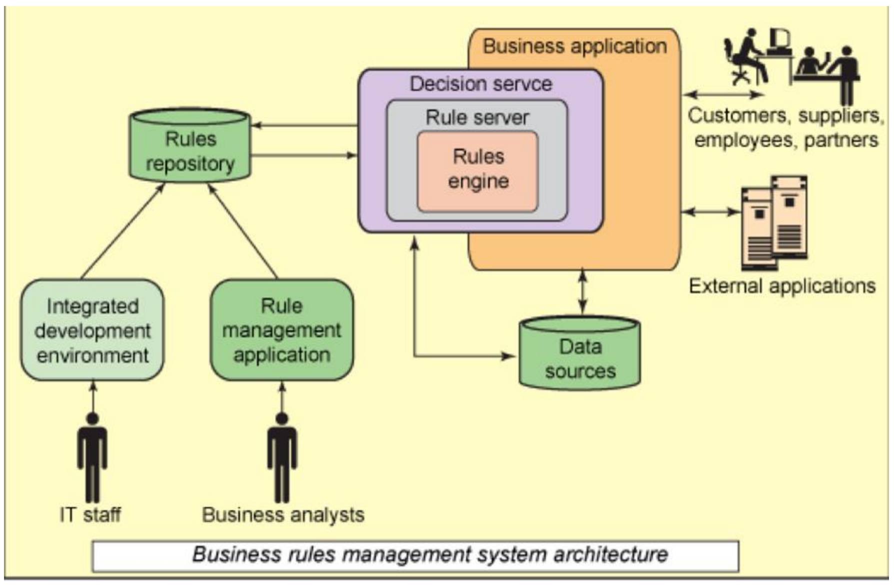

Business application
Decision servce
Rule server
Rules engine
Rules repository
Customers, suppliers, employees, partners
External applications
Integrated development environment
Rule management application
Data sources
IT staff
Business analysts
Business rules management system architecture

Figure 16 Generic BRMS architecture [34]

Drools Expert

Drools Expert is a component/subproject of the umbrella KIE (Knowledge Is Everything) project. It is a Java based rule engine DSL. The rule engine is the computer program that delivers Knowledge Representation and Reasoning (KRR) functionality to the developer. At a high level it has three components:

- Ontology ("Things" e.g. java Classes/Beans)
- Rules
- Data

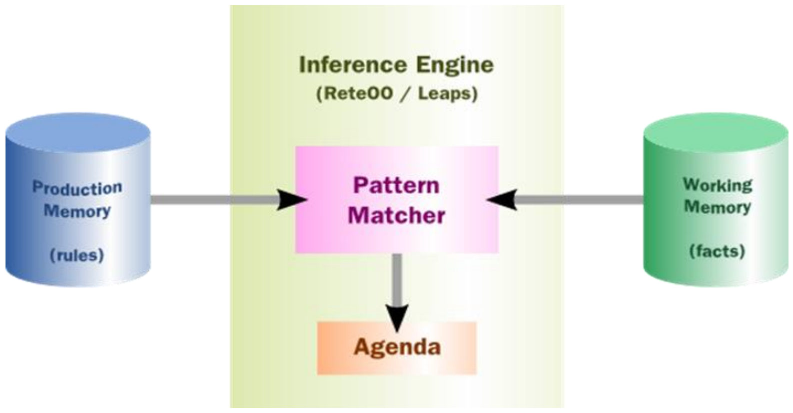


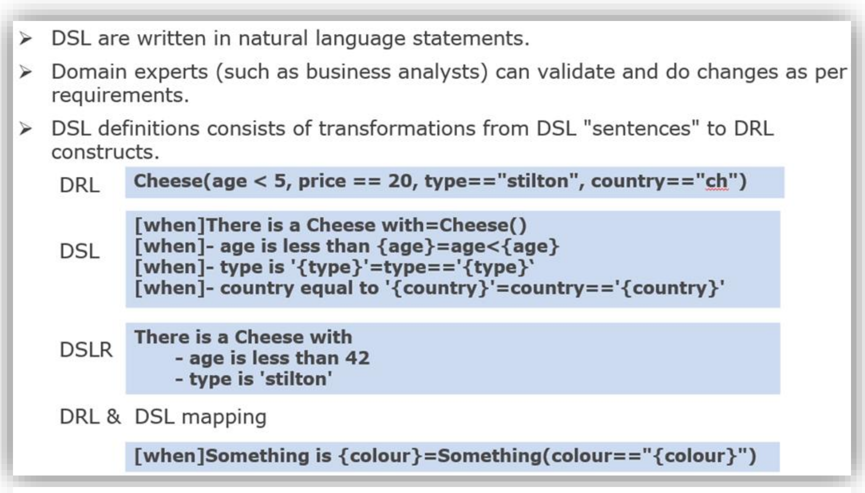
Figure 17 High-level View of a Rule Engine [34]

- The rules are loaded into production memory and are available at all times
- Facts are asserted into the Working Memory where they may then be modified or retracted.
- The Agenda manages the execution order of the conflicting rules using a conflict resolution strategy.
- The rules might be in conflict when more than 1 rule matches the same set of facts in working memory

Drools 6 has an enhanced and optimized implementation of the Rete algorithm for object oriented systems called as ReteOO (at the time of writing it is called PHREAK)

Drools Expert is a declarative, rule based, coding environment. Drools Rule Formats:

- Drools Rule Language (DRL)
- Domain-specific language (DSL)
- Decision tables
- Guided rule editor
- XML



➢ DSL are written in natural language statements.
➢ Domain experts (such as business analysts) can validate and do changes as per requirements.
➢ DSL definitions consists of transformations from DSL "sentences" to DRL constructs.

DRL
```
Cheese(age < 5, price == 20, type=="stilton", country=="ch")
```

DSL
```
[when]There is a Cheese with=Cheese()
[when]- age is less than {age}=age<{age}
[when]- type is '{type}'=type=='{type}`
[when]- country equal to '{country}'=country=='{country}'
```

DSLR
```
There is a Cheese with
      - age is less than 42
      - type is 'stilton'
```

DRL & DSL mapping
```
[when]Something is {colour}=Something(colour=="{colour}")
```

Figure 18 Drools Rule Formats [34]

Domain Specific Languages:

Domain Specific Languages (or DSLs) are a way of creating a rule language that is dedicated to user's problem domain. A set of DSL definitions consists of transformations from DSL "sentences" to DRL constructs, which lets user use of all the underlying rule language and engine features. Given a DSL, user writes rules in DSL rule (or DSLR) files, which will be translated into DRL files.

DSL and DSLR files are plain text files, and user can use any text editor to create and modify them. But there are also DSL and DSLR editors, both in the IDE as well as in the web based BRMS, and the user can use those as well, although they may not provide you with the full DSL functionality.

When to use a DSL:

DSLs can serve as a layer of separation between rule authoring (and rule authors) and the technical intricacies resulting from the modeling of domain object and the rule engine's native language and methods. If one's rules need to be read and validated by domain experts (such as business analysts, for instance) who are not programmers, one should consider using a DSL; it hides implementation details and focuses on the rule logic proper. DSL sentences can also act as "templates" for conditional elements and consequence actions that are used repeatedly in one's rules, possibly with minor variations. One may define DSL sentences as being mapped to these repeated phrases, with parameters providing a means for accommodating those variations.

DSLs have no impact on the rule engine at runtime, they are just a compile time feature, requiring a special parser and transformer.

Drools is considered to be one of the most elaborate and powerful automation description mechanisms. OpenHAB first implementation was using drools but due to its complexity was abandoned. [34]

7.2.8. Z-Way Home Automation

It is an extensible modular home automation system, based on JavaScript (using v8) intended to be used within the Z-Wave.Me home automation controllers.

Z-Wave.Me is a proprietary company founded by a group of engineers with a vision to provide stable, easy to use and highly powerful building blocks for a Z-Wave based network to control light, climate, heating, security and safety in homes and offices [35].

RaZberry project:

RaZberry brings Z-Wave protocol to the Raspberry PI board. The Razberry platform adds all the components needed to turn a Raspberry PI board into a fully operational and inexpensive Z-Wave gateway.

JavaScript Engine:

Z-Way uses the JavaScript engine provided by Google referred to as V8. V8 implements JavaScript according to the specification ECMA 5 1. Z-Way extends the basic functionality provided by V8 with plenty of application specific functions.

The Z-Way Software Architecture:

Z-Way is the portion of RaZberry that runs on the Rasberry Pi operating system level. The code comes as Linux executable with libraries and is using certain configuration and translation files that are described later.

Z-Way is a fully featured home automation controller supporting Z-Wave as communication technology. It allows to:

- Include and exclude devices and configure these devices, manage the network configuration and stability by visualizing the configuration and routing within the network
- Switch actuators such as electrical switches, dimmers, motor controls for sun blind, garage doors or venetian blind, door looks, heating thermostats and many more

- Access sensor data such as motion detection, temperature, CO2, smoke etc.
- Visualization of all functions of the Z-Wave network mapped to the floor plan or as tables simple to read
- Create logical connection between events created by sensors and actions performed by actuators

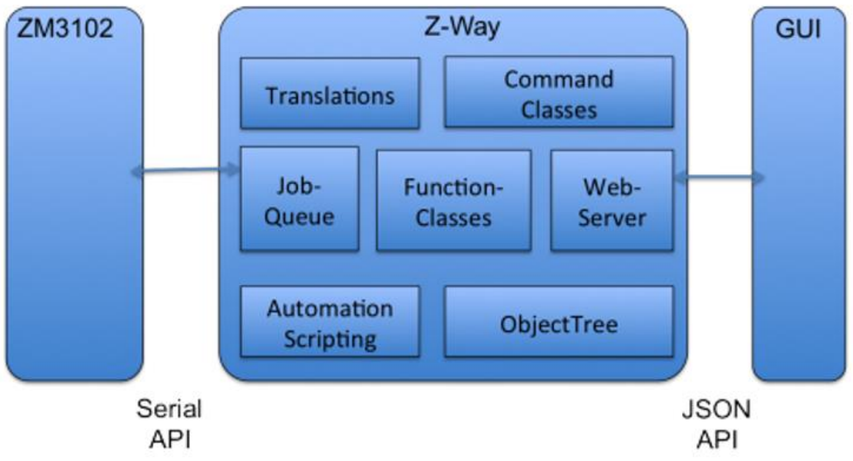


Figure 19 Z-Way Software structure [35]

Z-Way consists of several function blocks:
- The Job Queue: This is the core of Z-Way
- Function Classes: The implementation of all the commands to control the Z-Wave transceiver chip and the Z-Wave network
- Command Classes: The application level commands used to control Z-Wave devices in the network
- The JSON web server: It implements the application programmers interface Translation Functions: They help to translate machine readable tokens into human-readable strings
- The automation and scripting engine: This is the way to get the intelligence into the system.
- For more information about Z-Way such as the user interface, the JSON API structure refer to the Z-Way User and Developers Documentation available online.

The automation sub-system:

The automation subsystem allows writing automation scripts using JavaScript. It uses the ECMA compatible JavaScript Engine described previously. All the code realizing the automation engine is written in JavaScript itself and is available as open source for further study and modification.

The automation engine performs different actions based on events. The actions are either signal commands or scripts that can add additional logic and conditions. Events are either generated from the Z-Wave network or from an outside sources such as the Internet or even from a user interaction is causing certain actions, either within the Z-Wave network (e.g. switching a light) or outside Z-Wave (e.g., sending an email). In Z-Way all automation is organized in so called modules. [35]

It is important to note here that the core engine of this specific system is open-source but the system itself is proprietary.

7.2.9. Independent GitHub projects

GitHub is an endless source of valuable information. Various relevant independent projects found but due to space limitations we will only name the most distinctive.

logic4mqtt

It is an open source logic and scripting engine for Smart Home automation, based around MQTT as a central message bus.

It uses Java's generalized scripting interface (JSR-223) so scripts can be implemented in any script language supported by this interface. By default, the JVM ships with a JavaScript scripting engine (Rhino with Java 7, Nashorn with Java 8), but a variety of other interfaces is available for languages like Groovy, Jython and others.
Logic4mqtt provides a scripting host and a support API which provides

- MQTT access
- Event handling, based on incoming MQTT messages
- Versatile Timer support with both Cron-alike and natural language syntax
- Support for Sunrise/Sunset calculations, also tied into timer support
- Utility functions for network access, Wake-On-Lan etc.

The API is organized in classes. [36]

7.2.10. Independent npm projects

Npm is the (online) package manager for JavaScript. To be more specific, npm is a NodeJS package manager. As its name would imply, one can use it to install node programs. Also, if one use it in development, it makes it easier to specify and link dependencies.

Various relevant projects were found using Node.js but again we will mention the most distinctive due to space limitations.

mqtt-scripts

It is a Node.js based script runner for use in MQTT based Smart Home environments. It is intended to be used as the "logic layer" in your smart home, and offers a zero-boilerplate, straight forward scripting environment.

It follows the mqtt-smarthome architecture. Mqtt-scripts is quite similar to logic4mqtt - but allows the usage of modules via the require command. Mqtt-scripts could also be seen as something like "Node-RED without GUI". [37]

```
+---------------+
|               |                       +----------------------+
| Logic Engine  +------+          +-----+ Hardware Interface A |
|               |      |          |     | (e.g. KNX)           |
+---------------+   +--+----------+-+   +----------------------+
                    |               |
                    |  MQTT-Broker  |
                    |               |
+---------------+   +--+----------+-+   +----------------------+
|               |      |          |     | Hardware Interface B |
| Visualization +------+          +-----+ (e.g. Homematic)     |
|               |                       +----------------------+
+---------------+
```

Figure 20 Mqtt-smarthome architecture [38]

7.3 *Second Phase*

After the first phase which was basically a thorough research of alternatives and knowledge acquisition followed by a filtering process the second phase was initiated.

The systems described above present various common features such as language choice, architectural patterns, design patterns and general technology choices, which led to further investigation and to the extraction of useful conclusions.

The original goal was to find an off-the-shelf solution and possibly modify it to fit the needs of the current project. In order to find that solution a comparison table had to be created including all the filtered alternatives. For a table to make sense relevant comparison criteria had to be found. After careful deliberation with the project's stakeholders and personal research the table of section 7.3.2 was created.

7.3.1. Comparison Criteria

- #Commits

One quantitative metric was the number of commits (in GitHub) for the specific system in question (or more importantly for its automation engine when the number was available). This number shows the amount of work of a project.

- Last commit date

It is an indicator for how active is a project. The numbers shown in the table are just snapshots.

- Need of resources

Indicator for whether each engine is embeddable and on which platforms.

- Code Size (zipped)

Related with the above, the size of the code gives an initial indication of the resources needed.

- Technology size in the bridge

Each engine uses different technologies (e.g., programming languages). The languages' impact was tested in the bridge for OpenWrt. The outcome was compared against our project's memory requirements. As it is shown in a later section this is the decisive criterion for the final choice.

- Maturity

Maturity can be measured with different ways; in this case each system's or engine's creation date was picked. The earlier the better (time in the market).

- Hue compatibility

Each system's compatibility with Hue is examined. The main focus of the comparison was each engine and not each entire system thus this criterion was not so meaningful. The compatibility of the engine with ZigBee or OpenWrt could serve as better examples.

- Use of variables

One of the basic requirements for the future lighting engine is to support the usage of variables so it is used as a metric against all alternatives.

- Security

It is quite broad as a criterion but some systems do not even consider this aspect. Security (e.g., sandboxing, filtering) is an important requirement for the future engine.

- Size of community

Based mostly on the number of (developers) contributors and size of online forums.

- Modularity

Some of the alternatives were entire systems, some standalone engines or a combination (i.e. a modular part of a bigger system). Very important criterion since the goal was to be able to extract the engine and embedded to the Hue bridge.

- Documentation

How well documented is an engine or a system plays an important role on the understandability by a third-party user and thus on the final choice. That can be measured from things such as user guides, manuals, wikis, faqs, live chats and api docs.

- Deployment – Portability

What is the intended deployment platform for the system or the engine and how easy it is to embed it to another system.

- Learning Curve

That was a mainly personal metric for the author. Since the author will be the creator of the system his familiarity with each technology is important. Additionally, the generic user friendliness of each system is crucial.

- Scalability

A generic software criterion based on each system's review and documentation. Not of high importance. In other words, this is the ability of each engine to withstand growth.

- Extensibility

The ability to add functionality to the existing engine/system is measured.

- Type of license

Since the goal was to re-use existing software the type of license was really important. Even if all the systems were initially identified as open-source the different licenses have quite different limitations on usage, sharing and publication.

- Email communication

Last but not least, the author of the current project contacted every single creator of each of the systems/engines in order to elicit more information. The speed of the response and whether there was a response at all serve as indicators for picking the final solution.

7.3.2. Comparison table

Table 2 Comparison table of Alternatives

	OpenHAB	Vera openluup	Domoticz	Easy Rules	Home Assistant	Drools	Z-Way HA	Logic4mqtt
#Commits	8731 openhab, 2123 smarthome (sys)	60	3007(sys), 85(dzVents)	263	5310 (sys)	10249 (expert & fusion)	1.642	78
Last Commit Date	dec 2015 (old engine), may 2016 (new engine)	march 2016	May 2016 (lua scripts)	may 2016	may 2016 (sys)	may 2016	apr-16	feb-16
Need of Resources	resourceful devices mostly (PCs, ARMs, rasberry pis)	not promising, not much info, vera alikes I guess	generally resourceful, lua part embeddable	very small	most likely a PC or a raspberry Pi	high resource requirements	mostly for resourceful devices	seems embeddable
Code Size (zipped)	?	~KBs	?	100 KB	?	9 MBs	1 MB	53 KBs
Technology size in Bridge's OpenWrt (7,3 MBs)	xtend - DSL (java based)	**Lua (0,3 MBs)**	**Lua (0,3 MBs)**	Java	**Python (2+ MBs)**	Java - DSL	Javascript	Java - JavaScript
Maturity	Feb 2010 (sys)	Oct 2015 (for Openluup)	Dec 2012 (sys)	feb-15	dec-14	2001 or 2005	>1year	1 year
Hue Compatibility	yes - native (binding)	yes (?)	yes(!)	N/A	Yes	Not out of the box but OpenRemote did	Not natively but possible	Yes, gateway present
Use of Variables	yes	yes (lua)	yes (lua)	?	yes	yes	yes	yes (script)
Security	yes, https & authentication	upnp serious security issues	some kind of lua sandbox	?	?	yes, quite detailed	has some issues	implicit
Size of community	second largest after drools	small (forum & 1 contr)	medium (71 contr for sys)	very small (6 contr)	quite large (>150 contr for the system)	huge and active (97 contr for drools expert)	quite big (21 contr for engine)	really small (2 contr)
Modularity	new engine yes, but needs work	Not much info, maybe	The Lua implementation is not stand-alone. It is a combination of the Domoticz core, and the lua interpreter - The Lua scripts are reusable yes.	yes, standalone	probably not, does not seem quite modular	if we consider OpenRemote & OpenHAB then yes	seems modular but has licensing issues	by design
Documentation	quite detailed	there is no official doc for devs, limited for openluup	quite detailed (especially for lua)	fair	detailed	probably the most elaborate of all (860 pages documentation)	quite detailed and comprehensive	limited
Deployment – portability	similar to need of resources	similar to need of resources	similar to need of resources	similar to need of resources	similar to need of resources	similar to need of resources	similar to need of resources	similar to need of resources
Learning Curve	long and steep (dsl+drl)	considerably short	considerably short	very easy	medium (python)	the most complex engine	medium to difficult	very short
Scalability	yes, on Rules	?	multi-scripts	not significantly	multiple connected instances master-slave(system)	yes	possible	possible
Extensibility	bindings (high)	?	seems possible	//	possible	yes	possible	by design
Type of License	EPL v1	Not present (emulator of proprietary system)	**GNU GPL v3** (dzVents r 3rdparty)	MIT	MIT	Apache Soft. License 2.0	???	MIT 2014
Email Communication	Really fast by Founder	N/A	Fast but brief by founder	late but yes	No	No	No	Detailed but late

7.4 Conclusions

Some of the systems were eliminated before reaching the table. OpenRemote was skipped due to the fact that is using Drools (as its engine) and it would be a duplication. The npm project was quite similar with logic4mqtt so again in order to avoid overlap it was omitted.

Reading the table is not an easy task for someone not involved in the project; that is why specific and generic conclusions are listed below.

7.4.1. Specific

- OpenHAB is an entire system with an engine that it is not quite modular and thus cannot be extracted and used out of the box. Nevertheless, it was together with drools the most active and well documented system of all.
- Vera – OpenLuup is an emulator of a proprietary system, not well supported and not modular so a lot of complications can arise.
- Domoticz (engine) licensing together with the lack of modularity were the most important drawbacks.
- Easy rules served as inspiration but it was too simple for actual usage.
- Home assistant focus was not the engine and thus did not seem modular at all. The functionality served by the rules seemed quite primitive too.
- Drools size and complexity were the main drawbacks.
- Z-Way turned out to be proprietary system using an open source foundation for its engine.
- Logic4mqtt seemed promising but its support, documentation and language were the main obstacles.

7.4.2. Generic

None of the alternatives can be used out of the box. The original goal to find an off-the-shelf solution either as standalone engine or as a modular part of an entire system and modify that to our project's needs turned out to be unreachable. Some of the reasons are mentioned as follows:

Most of the alternatives were home automation systems with no or small modularity. Since, all the engines were focusing on the home level they were not lighting specific. Consequently, they lacked functionality or they were so complex and bundled with their parent system that could not be separated.

Furthermore, most of the alternatives were requiring many more resources (mostly memory) than those of Hue bridge v2 (Raspberry Pi resources were most of the times the lowest limit).

After an extensive meeting with a representative of IP&S (Intellectual Property & Standards) department of Philips it was obvious that the licensing would be one of the most decisive criterion in the case of using third-party software. For example, the GPL v3 license introduces problems with Philips patents and thus solutions with this license could not be used (for the final product).

During the process of evaluating the abovementioned criteria the need of measuring the programming language impact became imperative. The need of choosing a language for the future Lighting Automation Engine goes beyond the comparison of alternatives. All in all, it is quite clear that some possible languages should be picked and compared in the real environment i.e., the Hue bridge v2.

8.Engine Specification

In this chapter, we are focusing on the creation of a formal specification for the new engine. This is one of the two major outputs of this project (the other one was the alternatives investigation). Having just explained the general architecture we will here describe the mechanism of JSON Schema, starting from what is JSON and then what is JSON Schema. We will continue with its implementation for the new Lighting Automation Engine.

8.1 Introduction

Hue bridge software is mainly written in C and is exposed through the app for end users and through Hue API for third-party application developers. As was explained, in Hue API, all responses and new values are sent and returned in JSON with UTF8 encoding so it is easy to generate or parse.

The Hue interface allows developers to interface with and make use of the functionality of the Philips Hue system. Using this interface they can find information about the available devices in their local network, control these devices and other.

The Hue API is a RESTful JSON interface in which clients interact with resources in the Philips Hue system. What this means is that every device, group etc. in the Philips Hue system is represented by a unique URI which is interacted with.

In other words, if a new third-party developer wants to interface his application with Hue he needs to read the Hue API specification.

Up until now, Philips Lighting has offered a text-based, online specification which is used by external developers. Internally, there is a similar document called Hue API specification.

This specification describes the whole system and thus the rule engine too. It would make a big difference to have something more than a human readable document. JSON Schema serves exactly this purpose as it is explained in the following sections. [25]

8.2 JSON

To define what JSON Schema is, we should probably first define what JSON is. JSON stands for “JavaScript Object Notation”, a simple data interchange format. It began as a notation for the World Wide Web. Since JavaScript exists in most web browsers, and JSON is based on JavaScript, it is s very easy to support there. However, it has proven useful enough and simple enough that it is now used in many other contexts that do not involve web surfing. [43]

Additionally, JSON is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is the most common data format used for asynchronous browser/server communication (Asynchronous JavaScript and JSON - AJAJ), largely replacing XML which is used by AJAX (Asynchronous JavaScript and XML).

Furthermore, JSON is a language-independent data format. It derives from JavaScript, but as of 2016, code to generate and parse JSON-format data is available in many programming languages. [44]

At its heart, JSON is built on the following **data structures**:

- object:
    - { "key1": "value1", "key2": "value2" }
- array:
    - ["first", "second", "third"]
- number:
    - 42
    - 3.1415926
- string:
    - “This is a string”
- boolean:
    - true
    - false
- null:
    - null

With these simple data types, all kinds of structured data can be represented. With that great flexibility comes great responsibility, however, as the same concept could be represented in myriad ways. For example, you could imagine representing information about a person in JSON in different ways: [43]

```
{
  "name": "George Washington",
  "birthday": "February 22, 1732",
  "address": "Mount Vernon, Virginia, United States"
}

{
  "first_name": "George",
  "last_name": "Washington",
  "birthday": "1732-02-22",
  "address": {
    "street_address": "3200 Mount Vernon Memorial Highway",
    "city": "Mount Vernon",
    "state": "Virginia",
    "country": "United States"
  }
}
```

Both representations are equally valid, though one is clearly more formal than the other. The design of a record will largely depend on its intended use within the application, so there’s no right or wrong answer here. However, when an application says “give me a JSON record for a person”, it is important to know exactly how that record should be organized. For example, we need to know what fields are expected, and how the values are represented. **That is where JSON Schema comes in.**

8.3 JSON Schema and metadata

JSON Schema is a JSON media type for defining the structure of JSON data. JSON Schema provides **a contract** for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. [44]

The following JSON Schema fragment describes how the second example above is structured. [44]

```
{
  "type": "object",
  "properties": {
    "first_name": { "type": "string" },
    "last_name": { "type": "string" },
    "birthday": { "type": "string", "format": "date-time" },
    "address": {
      "type": "object",
      "properties": {
        "street_address": { "type": "string" },
        "city": { "type": "string" },
        "state": { "type": "string" },
        "country": { "type" : "string" }
      }
    }
  }
}
```

By “validating” the first example against this schema, you can see that it fails. However, the second example passes. Schema validation is a topic thoroughly explained in the next chapter.

8.3.1. Approach

The orthodox way to create a JSON Schema is to first include all the information that is needed and then incrementally start adding constraints. **Not all constraints can be expressed.** JSON Schema limits itself to describing the structure of JSON data, it cannot express functional constraints. [45]

8.3.2. JSON Schema primitive types

JSON Schema defines seven primitive types for JSON values: [46]

1. array
    - A JSON array.
2. boolean
    - A JSON boolean.
3. integer
    - A JSON number without a fraction or exponent part.
4. number
    - Any JSON number. Number includes integer.
5. null
    - The JSON null value.
6. object
    - A JSON object.
7. string
    - A JSON string.

It is impossible to describe the full capabilities (specification) of JSON Schema in the context of this document. For more definitive information one can search online. Nevertheless, the best way to present schema’s power is via examples.

8.3.3. Example – Product API

Another more elaborate example follows. Pretend we are interacting with a JSON based product catalog. This catalog has a product which has an id, a name, a price, and an optional set of tags. [45]

Example JSON data for a product API
An example product in this API is:

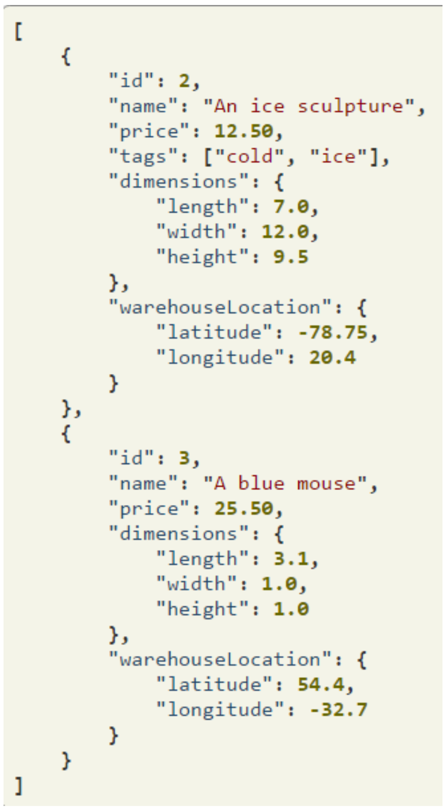
```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

While generally straightforward, that example leaves some open questions. One may ask:

- What is id?
- Is name required?
- Can price be 0?
- Are all tags strings?

When we are talking about a data format, we want to have **metadata** about what fields mean, and what valid inputs for those fields are. JSON schema is a specification for standardizing how to answer those questions for JSON data.

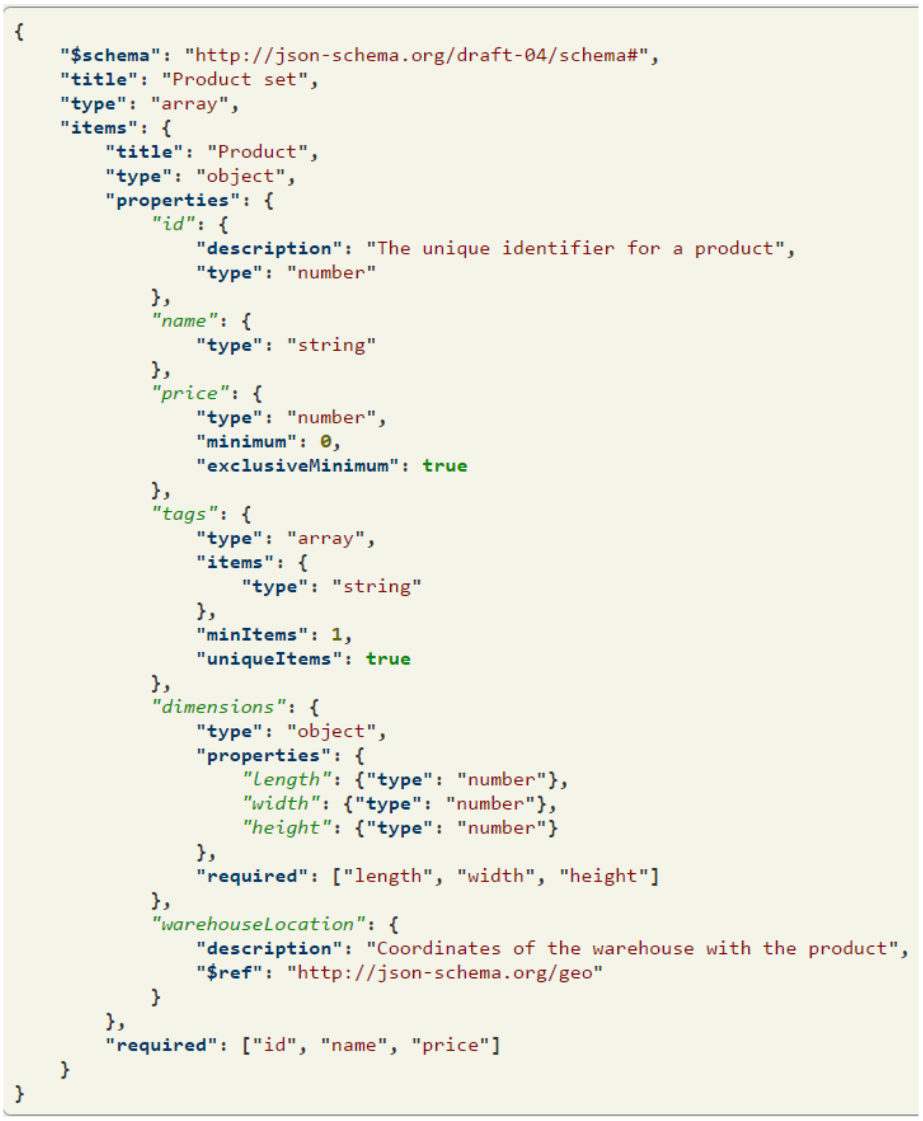
A specification for an array of products follows, with the products now having 2 new properties. The first is a dimensions property for the size of the product, and the second is a warehouseLocation field for where the warehouse that stores them is geographically located.



```
[
    {
        "id": 2,
        "name": "An ice sculpture",
        "price": 12.50,
        "tags": ["cold", "ice"],
        "dimensions": {
            "length": 7.0,
            "width": 12.0,
            "height": 9.5
        },
        "warehouseLocation": {
            "latitude": -78.75,
            "longitude": 20.4
        }
    },
    {
        "id": 3,
        "name": "A blue mouse",
        "price": 25.50,
        "dimensions": {
            "length": 3.1,
            "width": 1.0,
            "height": 1.0
        },
        "warehouseLocation": {
            "latitude": 54.4,
            "longitude": -32.7
        }
    }
]
```

Figure 21 Instances of products [45]

These are our data and now we want to create a formal description for them that could be both machine and human readable. The outcome is shown in the figure 41.



```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product set",
    "type": "array",
    "items": {
        "title": "Product",
        "type": "object",
        "properties": {
            "id": {
                "description": "The unique identifier for a product",
                "type": "number"
            },
            "name": {
                "type": "string"
            },
            "price": {
                "type": "number",
                "minimum": 0,
                "exclusiveMinimum": true
            },
            "tags": {
                "type": "array",
                "items": {
                    "type": "string"
                },
                "minItems": 1,
                "uniqueItems": true
            },
            "dimensions": {
                "type": "object",
                "properties": {
                    "length": {"type": "number"},
                    "width": {"type": "number"},
                    "height": {"type": "number"}
                },
                "required": ["length", "width", "height"]
            },
            "warehouseLocation": {
                "description": "Coordinates of the warehouse with the product",
                "$ref": "http://json-schema.org/geo"
            }
        },
        "required": ["id", "name", "price"]
    }
}
```

Figure 22 Set of Products Schema [45]

Schema notation explained:

- The above schema has four properties called keywords. The title and description keywords are descriptive only, in that they do not add constraints to the data being validated. The intent of the schema is stated with these two keywords (that is, this schema describes a product set).
- The type keyword defines the first constraint on our JSON data: it has to be a JSON array.
- Finally, the $schema keyword states that this schema is written according to the draft v4 specification.
- Id is a numeric value that uniquely identifies a product. Since this is the canonical identifier for a product, it does not make sense to have a product without one, so it is required (without it the Schema will not validate).
- Name is a string value that describes a product. Since there isn't much to a product without a name, it also is required.
- There are no free products. In JSON schema a number can have a minimum. By default this minimum is inclusive, so we need to specify exclusiveMinimum.

- Unlike the previous properties, tags have many values, and are represented as a JSON array. According to our imagination, all tags must be strings, but we are not required to specify tags. We simply leave tags out of the list of required properties. However, we have to add two constraints:
    - There must be at least one tag,
    - All tags must be unique.
        - The first constraint can be added with minItems, and the second one by specifying uniqueItems as being true.
- And also, since JSON Schema defines a reference schema for a geographic location, instead of coming up with our own, we will reference an existing one using the pointer notation “$ref”.

The above example is by no means definitive of all the types of data JSON schema can define. For more definitive information see the full standard draft. [47]

8.3.4. Why JSON Schema?

Having understood the usage of JSON Schema through the above mentioned examples now is the right time to explicitly list some of the main reasons why to use this media type in the first place. [48]

- It describes your existing data format
- It is clear, human- and machine-readable documentation
- It offers complete structural validation, useful for
    - automated testing
    - validating client-submitted data
- It has the widest adoption among all standards for JSON validation
- It is very mature (current version is 4, there are proposals for version 5)
- It covers a big part of validation scenarios
- It uses easy-to-parse JSON documents for schemas
- It is platform independent
- It is easily extensible
- It has 30+ validators for different languages, including 10+ for JavaScript, so no need for coding

Last but not least:

- It improves the existing text-based Hue API specification

8.4 Implementation

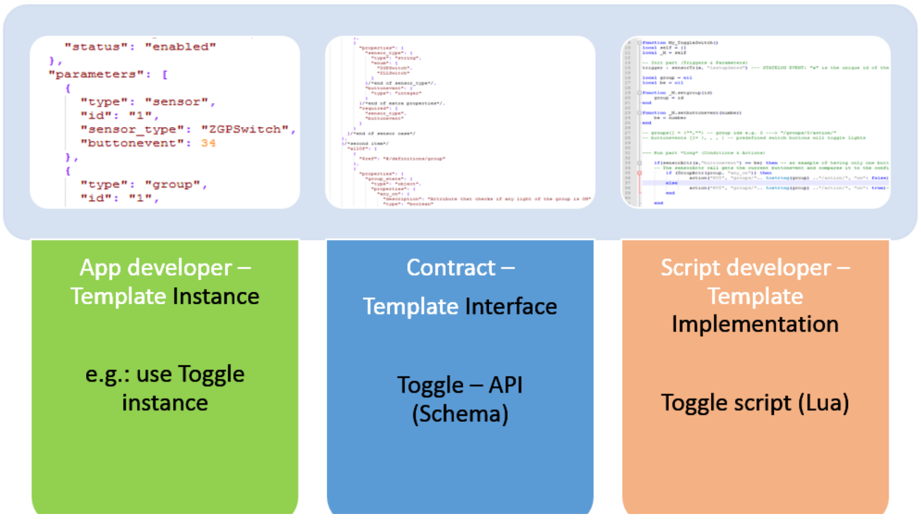
The fundamentals of JSON and JSON Schema were just explained so we are now ready to dive in to the actual implementation in the context of this project. This section presents the implementation of a formal specification describing a future Lighting Automation Engine. This specification is represented as JSON Schemas.

We would like to expose certain common behavior by offering templates in script format and allowing the addition of new ones. We identified three characteristic, common behaviors:

- Toggle (Switch ON/OFF)
- Scene-Cycling
- Dim Up/Down

For these cases, we created three Lua scripts as proposals for the template behavior layer (second from the bottom).

In combination with the common behavior (templates) identification we also identified two basic views of each behavior and a contract between them.



```
   "status": "enabled"
},
"parameters": [
  {
    "type": "sensor",
    "id": "1",
    "sensor_type": "ZGPSwitch",
    "buttonevent": 34
  },
  {
    "type": "group",
    "id": "1",
```

```
[,
{
  "properties": {
    "sensor_type": {
      "type": "string",
      "enum": [
        "ZGPSwitch",
        "ZLLSwitch"
      ]
    }/*end of sensor_type*/,
    "buttonevent": {
      "type": "integer"
    }
  }/*end of extra properties*/,
  "required": [
    "sensor_type",
    "buttonevent"
  ]
}
]/*end of sensor case*/
},
{/*second item*/
  "allOf": [
    {
      "$ref": "#/definitions/group"
    },
    {
      "properties": {
        "group_state": {
          "type": "object",
          "properties": {
            "any_on": {
              "description": "Attribute that checks if any light of the group is ON"
              "type": "boolean"
```

```
function My_ToggleSwitch()
local self = {}
local _M = self

-- Init part (Triggers & Parameters)
trigger : sensorTr(x, "lastupdated") --- STATELOG EVENT: "x" is the unique id of the

local group = nil
local be = nil

function _M.setgroup(id)
    group = id
end

function _M.setbuttonevent(number)
    be = number
end

-- groups[] = {"",""} -- group ids e.g. 0 ---> "/groups/0/action/"
-- buttonevents []= {, , , } -- predefined switch buttons will toggle lights


--- Run part "Long" (Conditions & Actions)

    if(sensorAttr(x,"buttonevent") == be) then -- an example of having only one butt
    -- The sensorAttr call gets the current buttonevent and compares it to the confi
        if (GroupAttr(group, "any_on")) then
                action("PUT", "groups/".. tostring(group) .."/action/", "on": false)
        else
                action("PUT", "groups/".. tostring(group) .."/action/", "on": true)-
        end

    end
```

App developer – Template Instance
e.g.: use Toggle instance
Contract – Template Interface
Toggle – API (Schema)
Script developer – Template Implementation
Toggle script (Lua)

Figure 23 Basic views of behaviors

1. The template implementation view
   - This is the view of a script developer; the creators of template Lua scripts are viewing the system from this perspective (e.g., toggle script creation). They are allowed to create and upload template scripts. Only authorized developers will be able to perform these actions.
   - The (lighting) behaviors that can be useful in a home are limitless. Philips would like to offer the opportunity to third-party audience to create new template behavior for variety and for increasing the adoption of the system.
   - Philips will expose to these developers (behavior experts, see Figure 39) a Lua API which they will use to create their templates.
   - The same developers upon creating their template behavior they will also create a JSON Schema for it. Ideally, there will be a mechanism to create Schemas from the Lua API (e.g., by parsing text manually put in the beginning of each template or programmatically extract information from each template and convert that to JSON Schema structure).

2. The template instance view
    - This is the view of the third-party app developers who want to use our automation system but completely ignore the existence of the underlying engine. For them the engine is a black box (e.g., toggle instance usage).
    - They are potentially UI creators and they want to instantiate the template behavior offered. They will also be able to create or modify instances.
    - The Schemas will be exposed to them hiding the complexity of the scripting engine. The Schemas will guide them describing the important information which will be exposed.
3. The template interface
    - This is the “bridge” of the two previous views. The template implementation (Lua scripts) includes all functionality and information but it cannot be exposed “as-is” to app developers.
    - There are various reasons to the previous choice such as to avoid exposing unnecessary complexity, to make creation of a UI easier, to avoid abusage of the system and so on and so forth.
    - We need a contract (interface) to formally define what kind of data should be exposed and how they will look like. In other words, we need JSON Schemas (e.g., toggle Schema).
    - JSON Schema is both machine and human readable.

9. Schema Validation

This chapter provides an account of the suitability of the created JSON Schema to meet the system requirements as listed in Chapter 6. This chapter discusses the various techniques for validation.

JSON Schema allows applications to validate instances, either non-interactively or interactively. For instance, an application may collect JSON data and check that this data matches a given set of constraints; another application may use a JSON Schema to build an interactive interface in order to collect user input according to constraints described by JSON Schema. [50]

The validation of the Schema itself can be made against the actual code. In an ideal scenario the future scripting engine will have on top of it a scripting API. From that API we will generate programmatically the necessary Schemas. In this project, we created the Schemas manually and thus we used online validators to validate instances against the Schemas.

Before going into the validation of the created JSON Schema it is worth remembering the example of the previous section. There we had two person-information JSON instances and one Schema. The first instance did not validate but the second did.

One may notice that the JSON Schema itself is written in JSON. It is data itself, not a computer program. It is just a declarative format for "describing the structure of other data". This is both its strength and its weakness (which it shares with other similar schema languages). It is easy to concisely describe the surface structure of data, and automate validating data against it. However, since a JSON Schema cannot contain arbitrary code, there are certain constraints on the relationships between data elements that cannot be expressed. Any "validation tool" for a sufficiently complex data format, therefore, will likely have two phases of validation: one at the schema (or structural) level, and one at the semantic level. The latter check will likely need to be implemented using a more general-purpose programming language. [43]

In the context of this project we will focus only on the structural validation of the created JSON Schema which will verify that the product was built in the right way.

The (online) validators are checking two things:

- If the instance and the Schema are (JSON) syntactically correct.
- If the instance is validating against the JSON Schema.

The former is considered to be relatively straightforward but the latter requires certain amount of experiments. Given the high degree of freedom provided inherently by the JSON Schema we need to be very careful in the validation process.

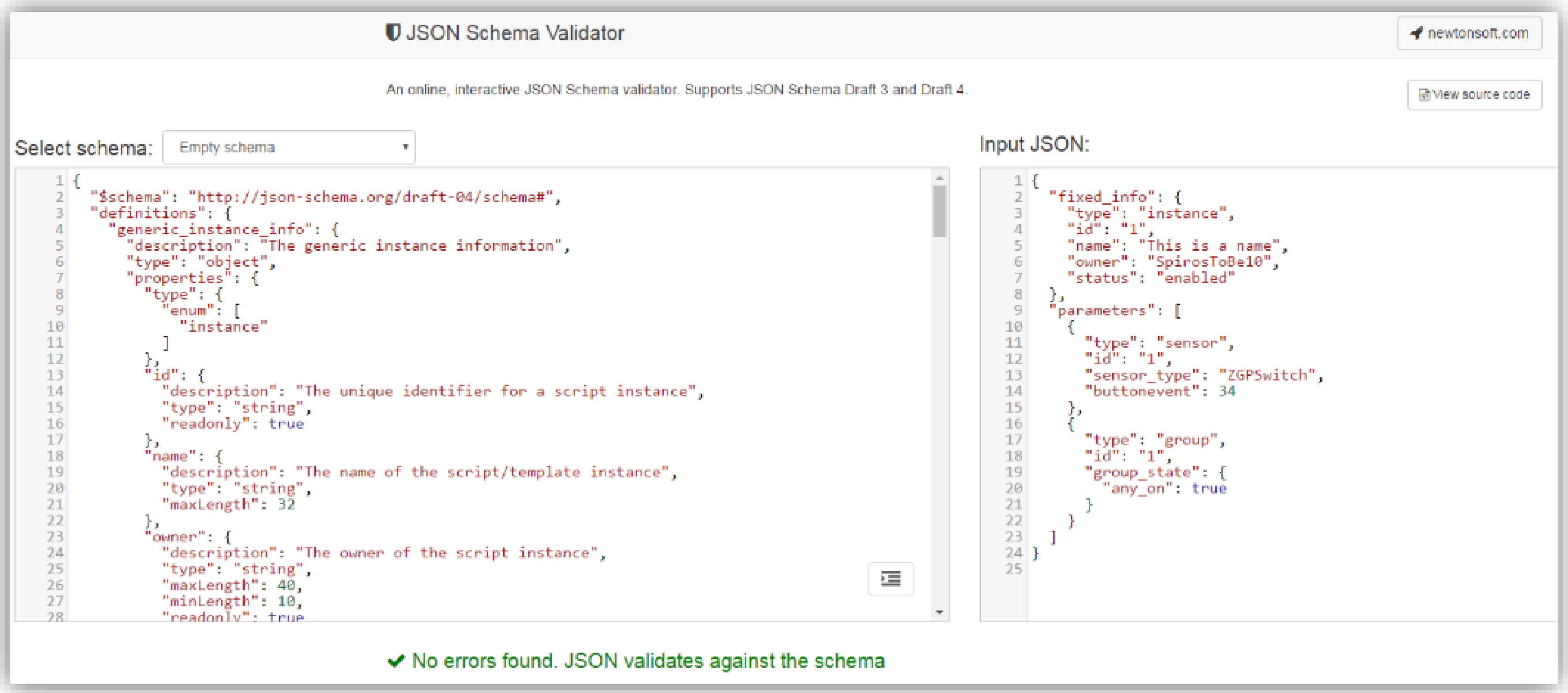
JSON Schema is just a proposal for JSON structure; this means that as long as we create instances that are syntactically correct and they are not included in any of the Schema's constraints they will validate successfully.

The point is to have enough constraints in order to cover the most important scenarios of instances and achieve the proper level of validation.

The design of the Lighting Automation Engine JSON Schema depends largely on its intended use within the Hue bridge, so there is no right or wrong answer here. However, when an application says "give me the resources that are required for a specific behavior", it is important to know exactly how that resources should be organized.

The process followed was to create multiple common instances and try to validate them against the Schema. When mismatches were encountered meant that the Schema was not generic enough and should be modified (usually by adding more constraints). At the time of writing the point of full coverage is reached and thus the proper level of genericity is achieved.

To be more specific, we refer to the specification created as the JSON Schema but in reality it consists of several sub-Schemas. So for all the Schemas we created separate example instances and we validated them against the corresponding Schema.



An online, interactive JSON Schema validator. Supports JSON Schema Draft 3 and Draft 4.

Select schema: Empty schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {
    "generic_instance_info": {
      "description": "The generic instance information",
      "type": "object",
      "properties": {
        "type": {
          "enum": [
            "instance"
          ]
        },
        "id": {
          "description": "The unique identifier for a script instance",
          "type": "string",
          "readonly": true
        },
        "name": {
          "description": "The name of the script/template instance",
          "type": "string",
          "maxLength": 32
        },
        "owner": {
          "description": "The owner of the script instance",
          "type": "string",
          "maxLength": 40,
          "minLength": 10,
          "readonly": true
```

Input JSON:

```
{
  "fixed_info": {
    "type": "instance",
    "id": "1",
    "name": "This is a name",
    "owner": "SpirosToBe10",
    "status": "enabled"
  },
  "parameters": [
    {
      "type": "sensor",
      "id": "1",
      "sensor_type": "ZGPSwitch",
      "buttonevent": 34
    },
    {
      "type": "group",
      "id": "1",
      "group_state": {
        "any_on": true
      }
    }
  ]
}
```

✔ No errors found. JSON validates against the schema

Figure 24 Validation of an example Instance [51]

10. Conclusions - Results

This chapter elaborates the results achieved by this project and the added value to the stakeholders. The two main achievements are the outcome of the alternatives investigation and the JSON Schema specification. Furthermore, future steps that Philips Lighting can follow are listed as proposals.

Improving home lighting experience is of great importance to Philips Lighting. The home automation environment is rapidly evolving and companies need to follow and satisfy the new trends.

Focusing on lighting, a fundamental relevant criterion is being able to explain to a user of the system why a light changed. The existing Hue engine offers almost no information for its internal activities. Additionally, as the lighting automation community grows the need of having abstraction layers for different user categories is continuously emerging. Developers ask for flexibility and functionality and end users ask for usability and simplicity. Besides these examples, there are many more that Philips Lighting wants to achieve in order to maintain its leading position as a provider of lighting solutions and applications. To address these challenges Philips Hue department initiated this project.

This document serves as input on how Philips Hue can further proceed in smart lighting. The main results produced by the author follow:

A long series of meetings with Philips experts and online communities resulted in **a list of requirements and use cases** capturing the needs and directions towards a future Lighting Automation solution.

Moreover, no solution can be designed and applied before acquiring enough knowledge for the question "what else is out there" (related projects). Consequently, one of the two major outputs of this thesis was the **exploration of sophisticated** (lighting) **automation engines** triggered by the increasing complexity of home lighting control use-cases captured. Philips identified this knowledge acquisition imperative in order to decide how a next step in lighting automation should be designed. A comparison table of open-source alternatives was created and good practices were extracted from these engines.

None of the alternatives can be used out of the box. The original goal to find an off-the-shelf solution either as standalone engine or as a modular part of an entire system and modify that to our project's needs turned out to be unreachable.

When the investigation outcome was solidified a conscious choice was made by the main stakeholders of this project. That choice was to focus on the conceptual design and architecture of a future Lighting Automation Engine rather than start implementing directly and blindly. As a result of this choice **a high level architecture of a scripting engine** is presented reflecting the main requirements captured in the earlier stages. This architecture can be a point of reference for a future implementation. It combines good practices of existing alternatives and encapsulates basic functionality which is limited or not even present in the incumbent Hue rule engine.

The benefits of scripting are countless and as long as the internal complexity is hidden the end-user experience will be the desirable. Many important features that the existing engine lacked such as the usage of variables, full Boolean logic, grouping and logging come practically out-of-the-box.

Philips Hue system is an open system. On one hand, it allows third party application developers to build apps for Hue. Apps to control lights, but also to define and create lighting automation behavior. On the other hand, it allows end users to interface with Hue system via these apps. The different user categories introduce design trade-offs and require extra abstraction layers on top of a scripting automation engine.

In order to solve these problems **a formal specification for a future Lighting Automation Engine** is developed using mainly JSON Schema. This specification is both machine and human readable and was recognized by Hue as of great importance. It is essentially a binding layer on top of the proposed scripting engine and describes ways to expose metadata to third-party applications hiding the complexity of the underlying engine. It is the second major output of this project and comes as a significantly improved replacement of the existing textual specification that is publically available.

Philips Lighting is now able to offer valuable information to external developers in order to create apps interfacing with Hue and broaden the openness of the system. Hue is also enabled to easily update and extend the new specification, two features (updatability and extendibility) inherently supported by JSON Schema.

The specification is separated in three parts implementation, instances and interface. The implementation part consists of Lua scripts which offer generic reusable behavior to native and third-party applications under the hood. The instances offer real-world examples of behavior and the interface (the actual Schemas) provides a contract-bridge between the previous two. These three pillars constitute the formal description of an engine capable to satisfy the most important requirements and to serve as guideline for a future implementation.

11. Project Management

This chapter gives an overview of the project management techniques used in the context of this project. The initial planning and how it evolved through discussion and reviewing.

11.1 Introduction

The project management approach used in this project is mainly based on Rationale Unified Process (RUP) [52]. Following RUP using agile extensions, the project period was split into four (iterative) phases.

- Inception
- Elaboration
- Construction.
- Finalization

The agile methodology suggests iterative and incremental development through so called sprints. Sprint is a short time span of one week (in this case), where a deliverable needs to be produced.

RUP was used as the backbone of the management process because it was found to fit best with the Philips Lighting way of working. The stakeholders involved were more positive in interacting with the process if these phases were clearly defined. This included reviewing documents and attending meetings based on a linear plan. For the purpose of planning, the main tool used was Trello. A visual web-based tool for project management providing a Kanban/Scrum board [53].

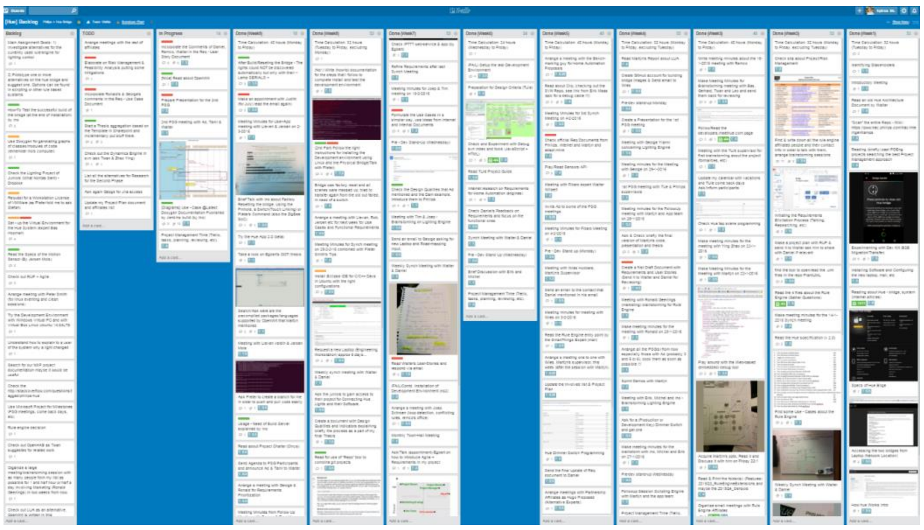


Figure 25 An example of a Trello board

The following sections dive into the details of the project, such as the work-breakdown structure, project plan and execution

11.2 Work-Breakdown Structure (WBS)

The initial WBS for the project was implemented relatively early in the project. It was an estimate of the time each phase would require and what it would include. Since the knowledge of the domain and the project itself was limited at the time, changes were expected to happen to this plan in later stages. The main deliverables for each of the four project phases are depicted in the following figure. [54]

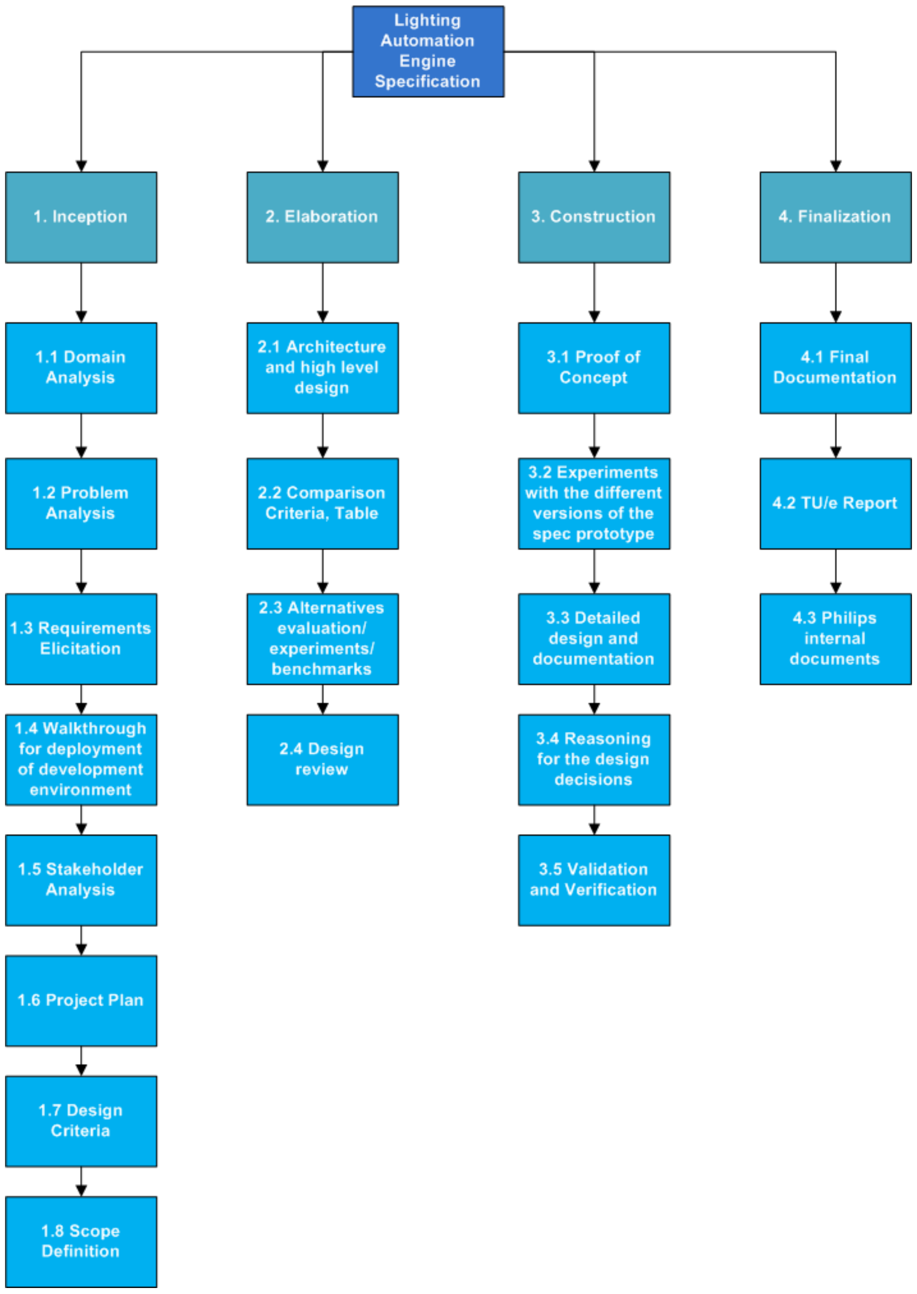

Lighting Automation Engine Specification
1. Inception
2. Elaboration
3. Construction
4. Finalization
1.1 Domain Analysis
1.2 Problem Analysis
1.3 Requirements Elicitation
1.4 Walkthrough for deployment of development environment
1.5 Stakeholder Analysis
1.6 Project Plan
1.7 Design Criteria
1.8 Scope Definition
2.1 Architecture and high level design
2.2 Comparison Criteria, Table
2.3 Alternatives evaluation/ experiments/ benchmarks
2.4 Design review
3.1 Proof of Concept
3.2 Experiments with the different versions of the spec prototype
3.3 Detailed design and documentation
3.4 Reasoning for the design decisions
3.5 Validation and Verification
4.1 Final Documentation
4.2 TU/e Report
4.3 Philips internal documents

Figure 26 Project's Work-Breakdown Structure

The time estimation for the initial plan as well as for the final plan were based on the theory supporting RUP. These plans and the respective changes can be seen in the following sections.

11.2.1. Initial

Table 3 Time allocation - created on 15 January 2016

Phase	Period
Inception	4 – 5 weeks (5th of January)
Elaboration	10 weeks
Construction	10 weeks
Finalization	5 weeks (until mid of September 2016)
Vacations	~ 3 weeks
Total	**33** weeks

11.2.2. Final

The biggest change in the planning was concerning the first phase of the project and that happened because the learning curve of the existing system was quite longer than originally thought. Additionally, the requirements elicitation process considered to be quite important and as such required more time.

Table 4 Time allocation - created on 15 July 2016

Phase	Period
Inception	10 weeks (5th of January)
Elaboration	11 weeks
Construction	9 weeks
Finalization	3 weeks (until mid of September 2016)
Vacations	~ 3 weeks
Total	**36** weeks

11.3 Project Planning and Scheduling

11.3.1. Methodology

The (scrum) method is adjusted to the needs of the project having:

- A unified backlog
- Weekly sprints
- Weekly synch – progress meetings with the two Philips supervisors:
  - Daniel Goergen, System Architecht
  - Walter Slegers, Software Architect
- Monthly PSG (Progress Steering Group) meetings with the two company supervisors and the TU/e mentor:
  - Tanir Ozcelebi, T.Ozcelebi@tue.nl, Assistant Professor, Computer Science
- Performance Evaluations every three months by the PSG members

11.3.2. Timeline

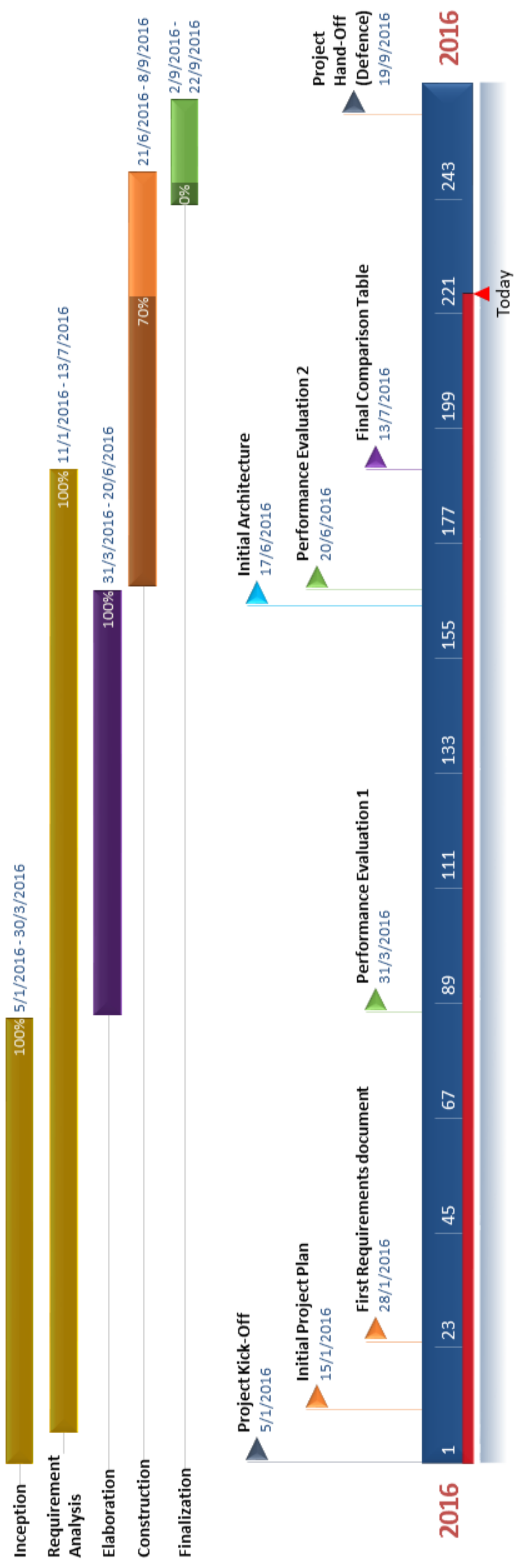


Figure 27 Snapshot of project's timeline

11.4 Execution

The execution of the project followed a structured path based on the project planning. The first two months comprised of meetings with stakeholders and reading domain literature. They also included getting to know the high-level requirements of the stakeholders and refining them.

Following the agile approach of iterative software development, every month a Project Steering Group (PSG) meeting was held where the progress of the project was presented by the trainee and the direction of the project was redefined. During these meetings, the trainee presented the current status as well as parts of the final deliverable. The stakeholders gave feedback and to some extent validation of the deliverable. If it was deemed necessary, additional meetings were scheduled by the trainee in order to obtain extra information regarding lighting specific or embedded domain knowledge. For each meeting, notes were taken by the trainee (meeting minutes), which were later put in documents and occasionally sent back to the stakeholders for review and comments.

By following the project plan, stakeholders could transparently observe how the project is progressing, and if the status satisfied their standards and requirements.

As mentioned in chapter 5, a number of risks were identified throughout the project that caused few changes in the direction of development. These changes were clearly presented to the stakeholders coupled with proposed mitigation strategies from the trainee.

12. Project Retrospective

This chapter presents a reflection account of the author on the course of the project in question. Starting from good practices, followed by improvement points the section ends with revisiting the design criteria first mentioned in System Requirements section.

12.1 Introduction

This project was very challenging and interesting at the same time. Since the author started with limited knowledge of the domain and technologies involved, at every step, he learned something new. In the meantime, he discovered that each of the domains that constitute this project (e.g., home automation, embedded, lighting, scripting) is huge in its own right with many sub-domains. Achieving the right balance between research depth (research into the domain specific) and width (research into a wide range of domains) was a particular challenge.

The most challenging part include working with different departments within Philips Lighting and experts, getting to know the domain, and translating their requirements into tangible results.

As with every project, a sufficient level of understanding of the domain is required to be able to translate requirements into a result. Luckily, the stakeholders from Philips Lighting were more than helpful and provided information and feedback whenever it was requested. This was important because it gave a two-way feedback, for the candidate to understand them, and for them to understand whether the candidate understands the domain.

One of the biggest concerns of the project was the formalization of the requirements into a specification, and their applicability in a system. Significant time was spent on this, especially in the last three months. An additional overhead was created by the fact that the expert knowledge needed was distributed. In the beginning, this distribution was quite an obstacle, but the continuous effort and brainstorming meetings mitigated this problem. The outcome of course was constantly evaluated by the stakeholders from Philips Lighting for preciseness and completeness.

Overall, the project was a great experience and a chance to exercise many skills related to software design. Cooperating with people and managing expectations is crucial and this was repeatedly exercised along the project. On top of that, several new technologies were used that broadened the knowledge and opened new horizons for the future.
■

12.2 Good Practices

The project provided a fertile ground to put the knowledge and skills that were acquired throughout the OOTI program into practice. In addition, it helped the author gain additional experience in a state-of-the-art environment and become a part of a company that has a leading role in the Lighting domain. Some practical tasks were deemed as important throughout the process and are worth mentioning.

- Frequent meetings with company supervisors and domain experts
- Make the most out of Progress Steering Group meetings
- Iterative implementation of the prototype
- Technology Learning

12.3 Improvement Points

When looking back on the project and reflecting upon the experience, there are some aspects that could have been viewed in a way that would benefit the process more. This retrospective is important and reveals the lessons learned and the situations where more attention can be applied in the future.

12.3.1. Project Planning

The creation of a project plan introduced difficulties for the author. The main factors for this were the large duration of the project (nine months) and the unfamiliarity with the embedded domain. These made estimations and work breakdown structure less efficient. Since the duration could not be changed, the domain unfamiliarity issue should have been dealt with more attention. The investigation for the solution started from a very generic point and then focused on more applicable solutions. If this convergence to more specific investigations had been initiated earlier, then the planning would have been more efficient and insightful.

12.3.2. Project Scope and Expectation Management

The expected results of the project were very unclear in the beginning. The scope of the project was broad and covered more than a simple Lighting Automation Engine (the engine was a small part of a big system). This caused significant delay while investigating all the related parts. After discussions the scope was narrowed down and that allowed the author to focus on the engine's specification. This decision could have been taken earlier so that less time would have been spent in investigating other parts.

Additionally, the large number of stakeholders and their different points of view created a frequent shift of focus or even conflicts. This could have been handled slightly more efficiently by earlier prioritizing the expectations and requirements of each stakeholder.

12.4 Design criteria revisited

During the requirements analysis process, three design criteria were identified as applicable and two as not applicable.

Applicable criteria:

- Functionality
- Complexity
- Documentation and presentation

Not Applicable criteria:

- Impact
- Inventiveness

Since the goal of the project was to explore more sophisticated Lighting Automation Engines, the construction phase of the project was steered towards solidifying the design rather than coding. The original choice of the 5 design criteria proved to be correct.

The designer determines the functionality in the form of specifications, staying within the envelope determined by the requirements. The main output of the project is a formal specification (JSON Schemas) describing the functionality (template syntax, instances, interface and implementation) of a future Lighting Automation Engine based on scripting.

The technologies (e.g., embedded C, scripting, JSON) which were used for the development of the formal specification together with the prototype introduce certain complexity. The creation of an engine which meets all the important requirements can only be realizable by a development team.

Last but not least, every step of the process was documented in detail and presented with relevant supportive models when required. The investigation of Lighting Automation Engine alternatives in combination with the requirements and use cases elicitation phase were the longest in time and took proportional space in the documentation.

As expected, being part of the pre-development team, the author did not create any economical or societal impact. The project was about experimenting and investigating consequently the outcome could only be a prototype.

There were some innovative ideas during the formalization of the specification but inventiveness was for sure not one of the main drives or characteristics of the final proposal.

Glossary

This section presents the terminologies used throughout this report along with their explanations.

Table 5 Glossary

Term	Explanation
PDEng	Professional Doctorate in Engineering
TU/e	Eindhoven University of Technology
SAI	Stan Ackermans Institute
OOTI	Onwerpersopleiding Technische Informatica
Cloud	General term for anything that involves delivering hosted services over the Internet.
API	Application Programming Interface
Server	A computing platform whose purpose is to serve other computing platforms.
Web server	Same as server, available over the Internet.
JSON	JavaScript Object Notation
JSON Schema	JSON Schema is a JSON media type for defining the structure of JSON data.
XML	eXtensible Markup Language
HTTP	HyperText Transfer Protocol
REST	Representational State Transfer (architectural style)
RESTful APIs	Interfaces that adhere to the REST style
EDA	Event Driven Architecture (architectural style)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
CLIP	Connected Lighting Interface Protocol
SDK	Software Development Kit
ZLL	ZigBee Light Link
IP	Internet Protocol
OpenHAB	Open Home Automation Bus
npm	Node.js Package Manager
WBS	Work-Breakdown Structure
RUP	Rational Unified Process
Scrum	An iterative and incremental agile software development framework for managing software projects and product or application development.
IDE	Integrated Development Environment
IFTTT	"If This Then That" is a free web-based service that allows users to create chains of simple conditional statements, called "recipes", which are triggered based on changes to other web services such as Gmail, Facebook, Instagram, and Pinterest.
FAQ	Frequently Asked Questions
HTML	Hyper Text Markup Language is a markup language for describing web documents (web pages).
DSL	Domain Specific Language
FSM	Finite-state machine
UPnP	Universal Plug and Play, a set of networking protocols
M2M	Machine to machine
POJO	Plain Old Java Object
BRMS	Business Rules Management System
SIM	Strategy Innovation Marketing meeting (Philips internal)
BRE	Business Rules Engine
MQTT	formerly known as MQ Telemetry Transport is an ISO standard publish-subscribe-based "lightweight" messaging protocol for use on top of the TCP/IP protocol
SoC	System On Chip

References

- [1] PDEng programs [Online]. Available: https://www.4tu.nl/sai/en/ [Last accessed: September 2016]
- [2] Software Technology [Online]. Available: https://www.tue.nl/en/university/departments/mathematics-and-computer-science/education/graduate-programs/pdeng-programs/software-technology/ [Last accessed: September 2016]
- [3] Philips, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Philips [Last accessed: June 2016]
- [4] Philips Annual Report 2015 [Online]. Available: http://www.philips.com/corporate/resources/annualresults/2015/PhilipsFullAnnualReport2015_English.pdf [Last accessed: May 2016]
- [5] Philips Intranet [Online]. Available: https://intranet.philips.com/ [Last accessed: April 2016]
- [6] Philips Lighting Intranet [Online]. Available: http://pww.lighting.philips.com/ [Last accessed: April 2016]
- [7] History of Light Bulbs, Google images [Online]. Available: https://www.google.nl [Last accessed: September 2016]
- [8] History of Home Automation [Online]. Available: http://betanews.com/2015/08/24/the-history-of-home-automation-from-the-beginning/ [Last accessed: April 2016]
- [9] Home Automation, Google images [Online]. Available: https://www.google.nl [Last accessed: September 2016]
- [10] SmartThings, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/SmartThings [Last accessed: September 2016]
- [11] HomeKit [Online]. Available: http://www.pocket-lint.com/news/129922-apple-homekit-and-home-app-what-are-they-and-how-do-they-work [Last accessed: September 2016]
- [12] Fibaro, Wikipedia [Online]. Available: https://pl.wikipedia.org/wiki/Fibar_Group (translated) [Last accessed: September 2016]
- [13] Wink, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Wink_(platform) [Last accessed: September 2016]
- [14] WeMo, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Belkin_Wemo [Last accessed: September 2016]
- [15] OpenHAB, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Draft:OpenHAB [Last accessed: September 2016]
- [16] OpenRemote, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/OpenRemote [Last accessed: September 2016]
- [17] Home Assistant [Online]. Available: https://home-assistant.io/ [Last accessed: September 2016]

- [18] Home Automation Protocols [Online]. Available: http://www.topten-reviews.com/home/articles/a-guide-to-home-automation-protocols/ [Last accessed: April 2016]

- [19] Tim M. Madsen, Home Automation Systems Integration, Integrating home automation systems to promote openness and adoption. Department of Computer Science
- Aalborg University, Software Engineering Master Thesis, spring 2010.

- [20] Philips Hue [Online]. Available: http://www2.meethue.com/en-us/ [Last accessed: May 2016]

- [21] Hue System Overview [Restricted]. Available: https://pww.trac-pl.philips.com/svn/intelligentlamps/Systems/Hue/trunk/ArchitectureDesign/System/System%20Overview-%20oneslider.pptx [Last accessed: September 2016]

- [22] Hue Specification [Restricted]. Available: https://pww.trac.philips.com/trac/intelligentlamps/browser/intelligent-lamps/Systems/Hue/trunk/ArchitectureDesign/Features/Bridge v2/bridge-v2.docx [Last accessed: May 2016]

- [23] Hue bridge platform [Restricted]. Available: https://pww.trac-pl.philips.com/svn/intelligentlamps/Systems/Hue/trunk/ArchitectureDesign/Features/Bridge%20platform/hue%20bridge%20platform.docx [Last accessed: May 2016]

- [24] Hue system and software architecture overview [Restricted]. Available: https://pww.trac-pl.philips.com/svn/intelligentlamps/Systems/Hue/trunk/ArchitectureDesign/System/Hue%20system%20and%20software%20architecture%20overview.docx [Last accessed: May 2016]

- [25] Public Hue API specification [Online]. Available: http://www.developers.meethue.com/ [Last Accessed: September 2016]

- [26] Philips Hue Wiki [Restricted]. Available: https://pww.trac-pl.philips.com/svn/intelligentlamps/ [Last accessed: September 2016]

- [27] Hue Rule Engine [Restricted]. Available: https://pww.trac-pl.philips.com/trac/intelligentlamps/browser/intelligentlamps/Systems/Hue/trunk/ArchitectureDesign/Bridge/rule%20engine [Last accessed: September 2016]

- [28] K. v. H. a. K. v. Overveld, "Criteria for assessing a technological design", March 2010

- [29] OpenHAB [Online]. Available: http://www.openhab.org/ [Last accessed: September 2016]

- [30] OpenRemote [Online]. Available: http://www.openremote.org/display/HOME/Home [Last accessed: September 2016]

- [31] Vera - OpenLuup [Online]. Available: https://github.com/akbooer/openLuup [Last accessed: September 2016]

- [32] Domoticz [Online]. Available: https://domoticz.com/ [Last accessed: September 2016]

- [33] Easy Rules [Online]. Available: http://www.easyrules.org/ [Last accessed: September 2016]

- [34] Drools [Online]. Available: http://www.drools.org/ [Last accessed: September 2016]

- [35] Z-Way Home Automation [Online]. Available: https://z-wave.me/ [Last accessed: September 2016]
- [36] logic4mqtt, GitHub project [Online]. Available: https://github.com/owagner/logic4mqtt [Last accessed: September 2016]

- [37] mqtt-scripts, npm project [Online]. Available: https://www.npmjs.com/package/mqtt-scripts [Last accessed: September 2016]

- [38] MQTT architecture, GitHub project [Online]. Available: https://github.com/mqtt-smarthome/mqtt-smarthome/blob/master/Architecture.md [Last accessed: September 2016]

- [39] Advantages of scripting languages [Online]. Available: http://www.sqa.org.uk/e-learning/ClientSide01CD/page_22.htm [Last accessed: September 2016]

- [40] OpenWrt [Online]. Available: https://openwrt.org/ [Last accessed: July 2016]

- [41] L. Bass, P. Clements and R. Kazman, "Software Architecture in Practice", Second Edition, Addison Wesley, 2003.

- [42] Event-Driven architecture, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Event-driven_architecture [Last accessed: September 2016]

- [43] Understanding JSON Schema [Online]. Available: https://spacetelescope.github.io/understanding-json-schema/about.html#about [Last accessed: September 2016]

- [44] JSON, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/JSON [Last accessed: September 2016]

- [45] JSON Schema basic info [Online]. Available: http://json-schema.org/example1.html [Last accessed: September 2016]

- [46] JSON Schema core definitions [Online]. Available: http://json-schema.org/latest/json-schema-core.html [Last accessed: September 2016]

- [47] Full JSON schema definitions [Online]. Available: http://json-schema.org/draft-04/schema#definitions [Last accessed: September 2016]

- [48] Why JSON Schema [Online]. Available: http://code.tutsplus.com/tutorials/validating-data-with-json-schema-part-1--cms-25343 [Last accessed: September 2016]

- [49] Software Verification & Validation, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Software_verification_and_validation [Last accessed: September 2016]

- [50] JSON Schema validation [Online]. Available: http://json-schema.org/latest/json-schema-validation.html [Last accessed: September 2016]

- [51] Web JSON Schema validator [Online]. Available: http://www.jsonschemavalidator.net/ [Last accessed: September 2016]

- [52] RUP, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Rational_Unified_Process [Last accessed: September 2016]

- [53] Trello, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Trello [Last accessed: September 2016]

- [54] WBS, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Work_breakdown_structure [Last accessed: September 2016]

- [55] OSGi, Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/OSGi [Last accessed: September 2016]

About the Author

Spyridon Skoumpakis received his diploma in Electrical and Computer Engineering from the Aristotle University of Thessaloniki, Greece in 2013. During his studies he has specialized in Software Engineering, Data Mining and Computer Vision, resulting in his Master's thesis "Workflow Extraction from UML Activity Diagrams." The main objective of the thesis was the conceptual decomposition of UML images in a form that can be processed by both humans and machines. Furthermore, his working experience spans from IT support in Hellenic Telecommunications Organization S.A (Greece) to Software development as a member of a professional team engaged mostly in designing and maintaining websites. Furthermore, he participated in an EU Lifelong Learning Program concerning Computer Science as a Tutor. From September 2014 until September 2016, he worked at the Eindhoven University of Technology, as a PDEng trainee in the Software Technology program from the 4TU.Stan Ackermans Institute. During his graduation project, he worked for the Hue department of Philips Lighting on a project focused on sophisticated home lighting automation use cases.

4TU.School for Technological Design,
Stan Akkermans Institute offers two-year
postgraduate technological designer
programmes. This institute is a joint initiative
of the four technological universities of the
Netherlands: Delft University of Technology,
Eindhoven University of Technology,
Universty of Twente and Wageningen University.
For more information please visit: www.4tu.nl/sai