5 OPENLICHT – A SELF-LEARNING LIGHTING SYSTEM BASED ON OPENHAB

K. Bierzynski [a, *], F. Kalleder [b], P. Lutskov [a], F. Rohde [a],
D. M. Rodríguez [a], J. Mena-Carrillo [a], R. Schöne [c], U. Aßmann [c]

[a] Infineon Technologies AG, Am Campeon 1-15, D-85579 Neubiberg (Germany),
(Kay.Bierzynski, Pavel.Lutskov, David.MoralesRodriguez, Juan.MenaCarrillo, Frank.Rohde)@infineon.com
[b] Bernitz Electronics GmbH, Ferdinand-Lassalle-Str. 9, D-82008 Unterhaching (Germany), f.kalleder@dled.eu
[c] Software Technology Group, Technische Universität Dresden, Nöthnitzer Str. 46, D-01069, Dresden
(Germany), (rene.schoene, uwe.assmann)@tu-dresden.de

KEY WORDS: openHAB, Lighting, Machine Learning, Microservice, MAPE-K Loop, Reference Attribute Grammar, Smart Home

ABSTRACT:
Today's smart devices often can only be remotely controlled and generate and deliver data. In the lighting domain the research project *OpenLicht* wants to improve this situation by developing an intelligent lighting system that integrates smart light bulbs and sensors to adapt lighting to user activities and preferences. As an additional goal the *OpenLicht* system should run at the network edge, because developments in the last years have shown that cloud-based infrastructures are not sufficient to fulfil all demands of the growing Internet of Things. As solutions to issues of cloud-based systems like reliability and privacy, edge and fog computing were suggested by the research community. These approaches are investigated and used in *OpenLicht* to realize a self-learning lighting system. As the base of this system the open Home Automation Bus (openHAB) was chosen, because of its state of development, size of community and range of devices that can be connected to it. In this paper the extensions and adaptations planned for openHAB to realize the self-learning lighting control are presented. Furthermore, implementation details are described for those parts that are already completed. Besides the software, a general overview of *OpenLicht* and its hardware details are discussed in this paper.

5.1 Introduction

Today's smart home technology market provides consumers a rich choice of devices that are connectable with each other and that are able to take advantage of cloud-based services. The product range includes security systems, connected thermostats and voice command devices. In Germany, most revenue is achieved by selling smart speakers, gateways and appliances (IoT Analytics, 2017). A survey conducted by *AudioAnalytic* concluded that most participants would be interested in intelligent smart home solutions.

Smart bulbs like Philips Hue and Osram Lightify provide convenient functionalities for manual light operation, but lack functionality that would enable intelligent behavior of the smart homes lighting installation. The basic relationship between user and lighting system is not changed. The user either operates the system manually, or has to program it, e.g., using rules and scripts. This lack of profound change severely limits the adoption of Smart Home technology, as technology experts have pointed out (Higginbotham, 2018).

The comprehensive smart home product range causes interoperability problems: smart home devices are not connectable, as they provide incompatible interfaces, e.g., ZigBee, Z-Wave or Bluetooth.

Due to its reliance on cloud-based services, smart home technology might suffer from latency-issues, as the data has to be sent to a remote cloud-provider. Moreover, the user's privacy is at risk, as cloud-providers can act independently from the user and might share and use the received data in an improper way. User surveys have shown that privacy issues are an important purchase barrier for potential smart home users (PwC, 2017).

The research project *OpenLicht* (OpenLicht Consortium, 2018) contributes to the solution of the aforementioned problems by exploring a lighting control approach that (1) is self-learning in that the user is the subject that technology is learning from, rather than the manual operator or the programmer of automatic light behavior and

* Corresponding author.

D. Uckelmann (Ed.): Proceedings of the 1st Conference on Smart Public Buildings, pp. 43–52,
19-20 October 2018, University of Applied Sciences Stuttgart, Germany.
ISBN: 978-3-940670-66-3

that (2) complements cloud-based services with locally provided services. The interoperability problem is addressed by incorporating openHAB as a middleware platform into the system. We Shave chosen openHAB, as it provides a rich set of device- and interface-bindings and is popular with a broad user community.

The goal of this paper is to provide an understanding for the *OpenLicht* hardware- and software-system and the planned openHAB extensions that will allow its users to apply a self-learning approach to achieve intelligent lighting control. In the next section an overview about the project is given. Goals and solution approaches are presented. Section 5.3 presents a hardware setup for the *OpenLicht* system and discusses potential alternatives. Section 5.4 describes the envisaged software architecture as well as individual architectural components. Section 5.5 presents the planned demonstrators, while section 5.6 concludes the paper.

5.2 Project Overview

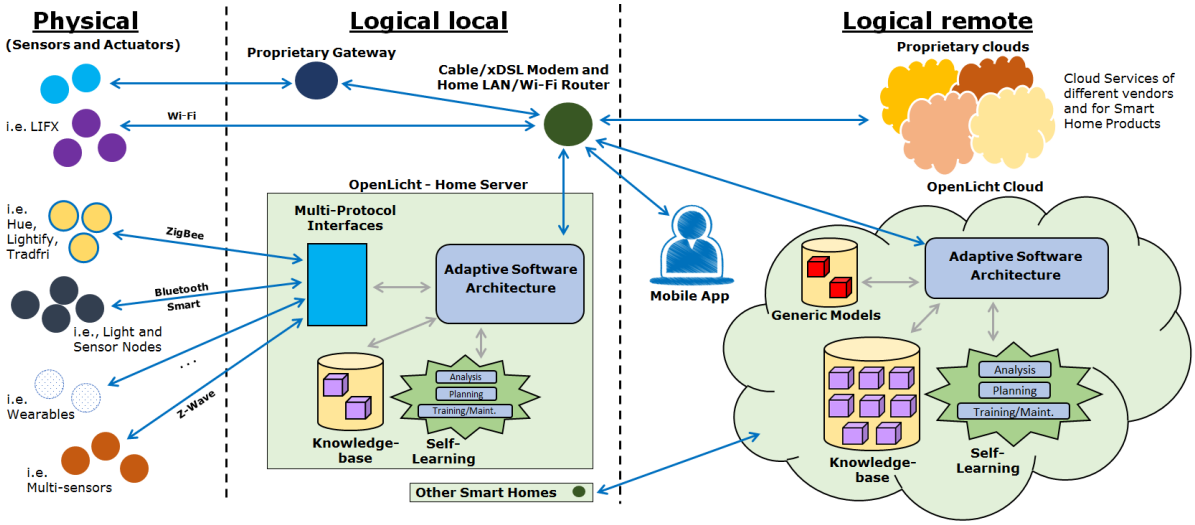

Figure 10. System architecture overview

Over the last years, many systems for smart lighting like Phillips Hue (Philips, 2018) or Ikea Tradfri (Ikea, 2018) appeared on the market. These systems provide a wide range of features such as daylight and occupancy simulation. However, apps or control devices are required for their use. This makes smart lights cumbersome and unintuitive to use. Furthermore, many of those systems need a connection to the cloud to work properly, which creates a privacy risk and threatens reliability, as already mentioned in the previous section.

OpenLicht, which is part of the Open Photonik program (VDI, 2018), explores how to alleviate these issues. State-of-the-art systems already employ automated light control; however, they are mostly rule-based. That means that users need to program those systems by themselves. Hence, the focus was set to a self-learning light system. Such a system is able to learn behavioral patterns and to correlate them with the preferences of the user. After a learn phase, it is ready to recognize what kind of activities users are doing and to adapt the light setting according to their preferences, current activities and the natural light. This idea is the central solution approach that is investigated in the scope of *OpenLicht*.

To get a better understanding on how a future smart home should look like, a survey was done. One question central for the learning part was where data about user activities and preferences should be stored and processed. The majority of the survey participants answered that they would prefer that the data remains in the home network for privacy reasons. To deal with this requirement, the self-learning system was combined with edge computing (Shi et al., 2016). Edge computing is an approach that describes the trend of moving data processing towards the network edge, leading to better latency and less traffic, both needed for the big amount of data created by the growing Internet of Things. Supporting this trend in the context of lighting applications, the automated machine learning framework *Ledge* was developed within the scope of *OpenLicht*. *Ledge* takes cleaned and labeled data of sensors or other data sources as well as configuration files that contain information about the execution environment as input and delivers C code of a trained machine learning model. The code can then be deployed on microcontrollers. It was designed to support makers and other developers in the development of intelligent lighting applications at the network edge. Besides *Ledge*, many small applications like openHAB bindings were written to process data and to connect sensors with XMC microcontrollers

(Infineon, 2018) and openHAB. The goal of these applications is to simplify prototyping, the usage of artificial intelligence and smart home hardware. At the end of the project, these small applications and the code of the complete system will be made available as open source software.

The overall envisioned architecture is depicted in Figure 10. It is divided into three parts, one containing all sensors and actuators, both of which are directly connected to gateways in the second part, the logical local processing part. In the logical parts, interfaces for remote accesses for example by apps are provided. Furthermore, it is the part where all processing takes place, either locally or remotely on cloud resources. In addition, the cloud resources are used to generate generic models from the data of multiple smart homes. These can be applied in a new smart home system to reduce the learning time. For evaluating the implementation of the architecture parts, a demo room was set up. Furthermore, a set of activities was defined that serve as test cases for the system in this room. Details of the implementation are described later in section 5.4.

Concerning hardware, central topics are design of a gateway for a self-learning system at the network edge and simplifications for development of light and sensor nodes. Two kinds of gateways are planned: The first type is a Raspberry Pi (Raspberry Pi Foundation, 2018) with an extension board developed in the project and assembled from open source components. This gateway serves as the central control unit of the *OpenLicht* system. The second type is a miniaturized gateway, which is used to scale the system. Furthermore, it is designed as a modular component that can serve as a control and processing unit for smaller applications. Besides these gateways, other hardware components are developed to simplify the prototyping and development of light solutions. To give makers and developers an overview and to support them in handling these components, a web platform is provided including tools that simplify searching for parts as well as for designing and simulating light solutions.

5.3 OpenLicht Hardware Concepts and Solutions

Core component of the *OpenLicht* hardware architecture is a central gateway, depicted in Figure 11, which is responsible for data processing and all control tasks. The gateway is connected to the internet via a Wi-Fi router which is considered to be available in almost all private households. For data transmission to the used sensors and actuators (exclusively lamps in the presented case) wireless communication interfaces are used in *OpenLicht*. The used radio interfaces are meant to be integrated directly into the gateway by usage of radio modules. In order to use proprietary radio interfaces or to include far-away areas, suitable proprietary gateways or bridges may also be integrated into the *OpenLicht* system.

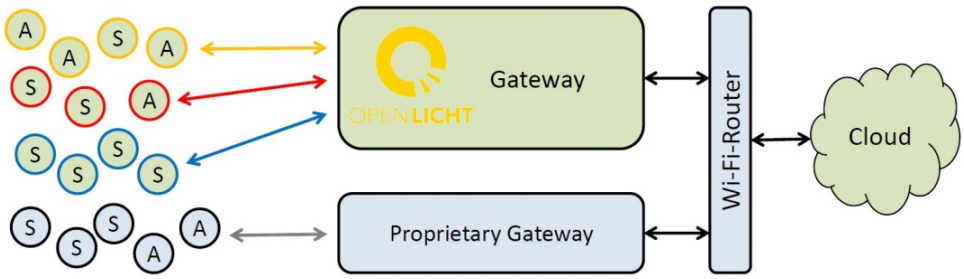

Figure 11. *OpenLicht* hardware architecture

To exchange data with sensors and actuators a wide range of wireless interfaces can be used in the system. Due to the high acceptance in the smart home sector and the great number of already existing products the following wireless interfaces are given priority in the *OpenLicht* project: ZigBee, Z-Wave, Bluetooth and Bluetooth Smart. Also, Wi-Fi is used because it can be assumed to be a standard component in most home networks and therefore no additional hardware is required.

In the smart home sector, a distinction has to be made between two different types of gateways. Some only work together with higher level control units or with the cloud, the others are largely independent and constitute such a higher level control unit by themselves. As part of the project, two different versions of the *OpenLicht* gateway are being developed.

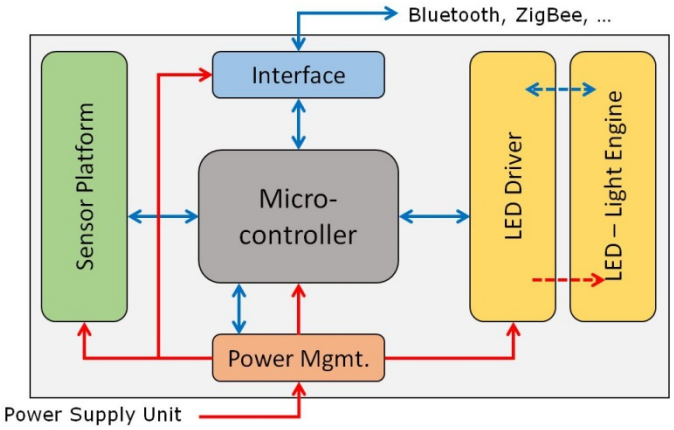

Figure 12. Architecture of the Light and Sensor Node

The open source version is based on a Raspberry Pi single-board computer with an additional expansion board. Due to additional components such as embedded interfaces, a security module as well as control elements for user interaction, the underlying hardware is extended. With its hardware capabilities, the Open Source Gateway is able to control the entire *OpenLicht* system. This includes not only the application of machine-learning algorithms, data processing and communication with the individual system components, but also the provisioning of suitable user interfaces like a web interface or app interfaces.

The miniaturized version of the gateway is based on a microcontroller architecture and is meant to be used for up- and down-scaling the *OpenLicht* system or for connecting smaller remote areas to it. To exchange data this gateway is equipped with pre-selected wired and wireless interfaces, e.g. Ethernet and Bluetooth, which are integrated directly into hardware. In stand-alone operation, the miniaturized version is able to control actuators, to capture and preprocess sensor data and to perform simple control algorithms. In the proposed *OpenLicht* system, this gateway requires a higher level control unit, such as the Open Source Gateway, which is able to control the entire system.

As part of the *OpenLicht* project, so called Light and Sensor Nodes will be developed. These may be sensors or actuators as well as a combination of both. A modular concept is applied to simplify the development of additional nodes. As shown in Figure 10Figure 12, functional modules are developed which can be combined according to the desired use case.

5.4 OpenLicht Software Concepts and Solutions

Based on the survey mentioned in section 5.2, requirements for the *OpenLicht* system were gathered. Among others, the following software requirements have been included: Integration of new components such as wearables, new sensors, as well as new software components during runtime; change between local and remote execution of computation; device-agnostic representation of sensors and actuators; learning of user activities and their preferences; maintaining service without internet connection.

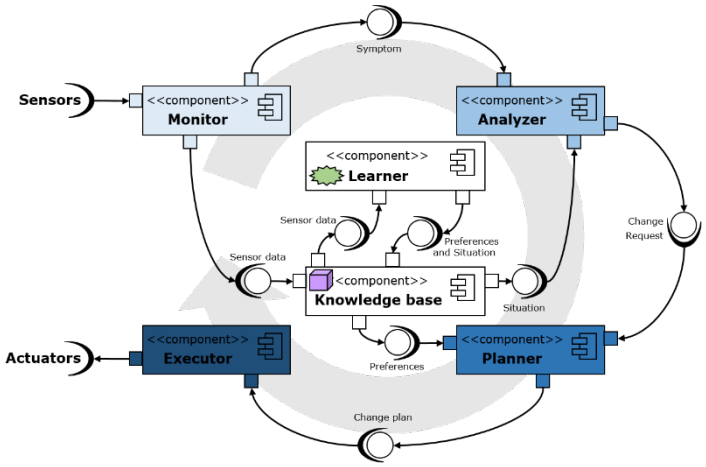

Figure 13. Extended MAPE-K feedback loop

Some of those requirements were met by employing openHAB as a middleware to connect to various sensors and actuators and thereby getting an abstract representation of them. Others were tackled in our project by developing extensions for openHAB described later in more detail.

The software architecture is organized as an enhanced MAPE-K (IBM, 2016) feedback loop, which was proposed by IBM and can since be found in many concepts and implementations of adaptive and self-adaptive systems. The name is an acronym of its four phases *monitor*, *analyze*, *plan* and *execute*, as well as their source of information, the *knowledge base*. In *OpenLicht*, this concept was extended with another component, the *learner*, which is responsible for learning activities and preferences of the user.

In the following, we will briefly describe the basic concept of the MAPE-K architectural approach and our extension. The *monitor* component continuously receives sensor data and writes this data into the *knowledge base*, possibly aggregating it beforehand. Once an update occurs, the *analyze* component checks, whether it was relevant, i.e., if a previously learnt activity can be recognized, or if other changes occurred that would require the system to adapt itself. In both of these cases, the *plan* component is invoked, otherwise the *monitor* component takes over. The *planner* is the central component, as it computes the reaction of the system to changes. Therefore, sensor data, learnt situations, user preferences are combined to derive the needed actions. Those will then be forwarded to the *execute* component which generates suitable commands to be sent to the actuators. The *knowledge base* is responsible for storing all information known to the system, e.g., sensors, their position and current state, learnt activities, preferences related to those activities, other relevant context information as well as historical decisions of the system.

Our extension introduces the *learner* component having three tasks: First, it takes all the data from sensors, lamps and users that are required for learning activities and user preferences. Secondly, it estimates the current user activity when triggered by *analyze.* Thirdly, it determines the user preference in context of the current activity and generates a fitting lamp configuration plan after the *analyze* phase and on request from the *plan* component. Besides these main tasks, the *learner* monitors the input and output of all trained machine learning models and retrains them when a change in the input data or model results is detected.

To relieve the home system from the possibly heavy computation needed to, e.g., train the model produced by the *learner*, a subset of the components can be moved to remote execution in a known cloud-based server or to other nodes in the home network. To achieve this, all components have incoming and outgoing ports which can be plugged together as needed as shown in the following figure.

The last part of the software architecture that needs to be discussed is the machine learning pipeline depicted in Figure 14. Activity recognition and preference planning are the central components that realize the learning features of the *OpenLicht* system.

The first is part of the *analyze* step of the MAPE-K loop and takes sensor data as input. However, data of more complex sensors like radar sensors cannot be directly used in many cases. Their data must be preprocessed to obtain suitable information for the recognition. After the pre-processing, the current activity is predicted with a voting classifier (Zhang et al., 2014) that uses multiple feedforward neural networks and decision trees. The exact number of the used models depends on the size of the smart home or apartment in which the system is deployed.

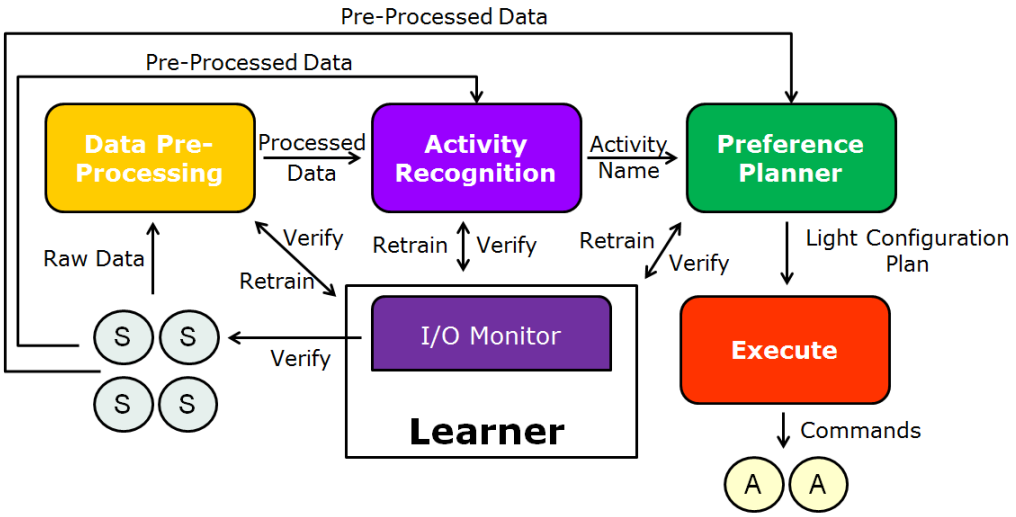

Figure 14. Learning process overview

The results of the prediction and data of some sensors are used as input for the preference planner component. This component is the central part of the *planner* and uses different sensor data as input depending on the trade-offs the system should make. The light intensity sensor is of fundamental importance, as it is necessary to configure the light in an area correctly. However, when the system needs to save energy, information from power meters will be utilized in the planning as well. Independent of used sensors, input data is processed by *DQN-Learning* (Mnih et al., 2015). This processing approach was chosen for the preference planner, as it is able to handle many different trade-offs due to its generality. After processing, the planner provides a list that contains the configuration of every lamp that is close to the current location of the acting user. This list is used to generate commands in the *execute* step.

The last component of the pipeline is the *I/O monitor*, which is part of the *learner*. This *monitor* is necessary, since the smart home is a dynamic environment. Weather, burglars, furniture rearrangement and new inhabitants are only some of the circumstances that can change the behavior of the sensor in and around the house and as a consequence the output of trained machine learning models. Hence, the *I/O monitor* records the input data after the data pre-processing and the results of the activity recognition and preference planner. They are then verified and tested for concept drift (Gama et al., 2014) by comparing them to historical data. In this context, *concept drift* means that the accuracy of the activity recognition or preference planner drops over time due to changing sensor behavior. When the *I/O monitor* detects a concept drift, it tries to find the cause in the input data and triggers a retraining of those models, whose results started to drift.

In the following subsections, we explain how we realized or plan to implement the MAPE-K loop parts with openHAB and extensions. However, the implementation of the cloud part of the system is not discussed as work on this part was not started yet. The *learner* is not covered within an additional subsection, since it is strongly interweaved with *analyze* and *plan*.

5.4.1 Monitor

The basic components of the system are the sensors and other data sources. The latter may include weather web sites, the state of the lamps, which are important for learning the preference of the user. To integrate them in the self-learning lighting system, the standard binding infrastructure of openHAB is used.

The next step in the system process is data pre-processing, whose kind and implementation depends on the sensors and their position within the data processing chain. On the lower levels, the data of one sensor is processed in most cases by filtering or normalizing the data. In some cases, such processing is provided by the sensor vendor and done directly on the sensor node. In case of no available pre-processing for sensors used in the *OpenLicht* demonstrators, we developed the necessary software and integrated it with the corresponding openHAB binding. We choose bindings as implementation points, because they are typically focused on a product or product family and may therefore preferably be extended with specialized data preparation and transformation services. An alternative implementation approach would be to let the bindings communicate with external sources that provide these services and delegate the processing to them. Delegation to external sources is one possible approach to scale the data pre-processing. Another strategy, we plan to utilize for the *OpenLicht* system, is to move the program with the services to nodes that are located logically between the sensor and the central gateway. This means that the binding would no longer communicate directly with the sensor, but with the program instead. Furthermore, it would be also possible to move the binding code at runtime from the central gateway to other nodes. In any case, the data pre-processing services would be monitored by the *I/O monitor* and be maintained by the *learner*, since they have a big impact on the data quality and often artificial intelligence approaches are their central components. After the data was transformed to some degree and checked by the *I/O monitor*, the higher level data pre-processing services aggregate the transformed data from multiple sensors into condensed information. These services can also be implemented in bindings to process data from product families. However, we tried to integrate them into the item infrastructure of openHAB to reuse as much of existing services as possible. In the approach we experimented with, the results from low level services are stored in items. These items are then used as input for the aggregation services. These services can be realized as custom rule triggers, when the result is not needed in other processes or as we did by using group items. They have a set of aggregation functions, but these are not sufficient for more complex processing. We tried to realize more functions and possibilities to use functions in conjunction with each other to solve this problem, which significantly enlarged the item definitions and made them more confusing. Furthermore, programming the services in this way was very restrictive. For these reasons, we decided to replace the representation of devices and the home by items with an approach that utilizes the *knowledge base* presented in section 5.4.5. Currently, the *knowledge base* runs as an extra system and is connected to openHAB over an MQTT broker. We are considering integrating the ideas behind the *knowledge base* into openHAB items.

The last component to be described in the *monitor* part is the *I/O monitor*. In the first version of the *OpenLicht* system, the *I/O monitor* is implemented as a set of openHAB rules with custom actions. For handling historical data, these rules access a SQL database that is connected over the *MariaDB JDBC* persistence service. However, the results of testing or verifying the data processing are archived by directly writing them into another SQL database that runs concurrently with the system and is separated from the database, which stores all item states. The system uses two databases to simplify testing and changing of the *I/O monitor*. In future iterations of the system, we plan to develop monitor functions that can be added to items or to *knowledge base* objects in a similar way as the aggregation functions are added to group items.

5.4.2 Analyze

In the *OpenLicht* system, the *analyze* phase of the MAPE-K loop consists mainly of the activity recognition. However, the transition from the data pre-processing to it is blurred, since the recognition is divided into many small parts and some of them, like determining the body posture of a person, could be categorized as pre-processing service or as an application of the *analyze* phase depending on the context and definition. Hence, the activity recognition and high level data pre-processing services are handled by the *I/O monitor*, *learner* and other parts of the system in the same way to provide flexibility for further developments.

The parts of the activity recognition are implemented in the first iteration of the *OpenLicht* system as openHAB custom actions. These actions use the optimized neural network and decision tree code of the *Weka* library (Hall, et al. 2009). Furthermore, they are integrated into openHAB rules, which communicate with each other. These rules, together with the custom action instances, form the mentioned voting classifier.

In the current system, all parts of the classifier run on a single Raspberry Pi together with the rest of the system. This will be changed in the future to enable scaling. It is planned to integrate a voter that is represented by one rule and an action instance into a *Docker* container (Rad et al., 2017). This allows the voter to be easily moved to another node of the home network. A special binding based on MQTT is intended for the communication with those other nodes. The container approach will not only be used to distribute the voting classifier, but for data processing of the system in general. Besides the implementation of the distribution, we are working at integrating the voters in the *knowledge base* for faster processing.

The voter actions do not include methods for learning, because separating the learning from the execution increases the system stability due to the possibility of testing and verifying the model before it is deployed to the running system or adapted at runtime. Hence, the *learner* component takes care of the training and is also implemented as a set of openHAB rules and custom actions. To select and train the algorithms, the *learner* uses *Weka* for activity recognition and the *Encog* library (Heaton, 2015) for the preference planner. It is planned for future system versions to unify both learning applications, to add more machine libraries to increase choice for developers and to only use the algorithm code from libraries. For this purpose, the *learner* will be reworked restructured to employ automated machine learning (Feurer et al., 2015). This means that some training steps are automated to reduce the amount of necessary support services and user input. The latter is especially important to realize training of the system at home. At the moment the system is not able to do this as it just reads the prepared data sets and learns from them. Afterwards, the learned parameters are sent to the corresponding custom actions, which save them in a file format of their library and load them when necessary. However, the actions only save the newest state of the algorithm parameters. Older states of the parameters are archived by the *learner* by writing them directly into the SQL database used by the *I/O monitor* for storing testing results. The adaptation of the models in response to changes is done in the *learner* as well. It has a copy of every model used in an action to do this and waits for the *I/O monitor* to trigger the relearning of them.

5.4.3 Plan

The preference planner makes up the *plan* step of the *OpenLicht* system. The planner is implemented like the activity recognition in the current system as a set of openHAB rules and custom actions. The actions use the algorithm code of the *Encog* library. Furthermore, the planner has two *DQN-Learner* models per room. One model is used for predicting the color of the lamps and the other for determining the light intensity and temperature. The advantage of splitting the prediction like this is that for lamps without color only one model needs to be used and maintenance becomes simpler. After both models have finished processing, their results are combined into one list and sent to the *execute* part. For future iterations of the system, we plan to implement the planner like the activity recognition using the *knowledge base*.

The *DQN-Learner* is a reinforcement learning algorithm and as such relies on user input to learn the correct configurations. In the *OpenLicht* system, the preference learning is designed as follows: First, the system only

gathers data about activities and light states. After the system has learned activities and corresponding light preferences, it starts to control the lights itself. The user can set the system into a learn state to adapt preferences any time after the initial training. Only the last part of this process is implemented in the current system. It was realized with a switch item that triggers some monitor rules, which are part of the *I/O monitor*. These rules gather information about the state changes of the lamps and send them to the *learner*. Then, the *learner* adapts the corresponding *DQN* models.

5.4.4 Execute

The implementation of the *execute* phase of the *OpenLicht* system is mainly provided by the openHAB binding infrastructure. Only an additional openHAB rule is needed that iterates through the list provided by the preference planner and triggers the bindings of the lamps whose state should be reconfigured.

5.4.5 Knowledge Base

To store all relevant data in a smart home, an extensible mechanism is needed to describe the structure and possible computation for this data. We use the rather unusual framework of Reference Attribute Grammars (RAGs) (Hedin, 2000) to cope with these challenges. RAGs are most commonly used in the compiler construction domain to describe the structure of parsed source code and to transform this representation into code of another language or into machine code. We use RAGs to describe the structure of the *knowledge base*, i.e., the structure of possible information to store. This allows us to additionally specify computation to infer new or to change existing pieces of information.

Specifically, we can mirror the information stored in openHAB and extend it with the learnt models like a decision tree or a neural network. Those models are produced by the *learner* component and pushed into the *knowledge base* accordingly. Using RAGs, we can specify an interpretation function for those models and, thus, evaluate them for the current situation. Hence, we can evaluate the decision tree and get the correct preference.

To implement our *knowledge base*, we employ JastAdd (Ekman & Hedin, 2007), an extensible RAG system extended by bidirectional relations (Mey et al., 2018). The structure of the data is specified using an EBNF-like syntax for production rules, i.e., to describe possible types in the *knowledge base*. With this, a tree-like structure results, which can be extended to a graph using reference edges. To specify computation, attributes can be defined for certain types. As JastAdd produces plain Java code, attributes are just Java methods with a special signature syntax.

An intriguing feature of JastAdd is its ability to incrementally evaluate the specified attributes. This means, if an attribute was previously computed and the *knowledge base* has not changed in the meantime, the value is cached, and will be returned immediately. If there were some changes, only those attributes will be recomputed that depend on changed data.

Mirroring the current state of all openHAB items in the *knowledge base* has also another advantage. We are able to write and process complex ECA-rules, which can use previous states of sensors, recognized activities and other context information not available in openHAB. Furthermore, the actions to execute upon recognizing the event of a rule can be more complex, such as updating the learnt models, or any other computation expressible with Java.

5.5 Demonstrator

As part of the research project, a demo room equipped with the *OpenLicht* system is set up. This room is used to evaluate the developed hardware and software, as well as to collect sensor data needed for machine learning and modelling. Later, it will also be the final demonstrator of the overall system. As central control component the open source version of the *OpenLicht* gateway is used. In addition to the developed Light and Sensor Nodes, lamps and sensors from various vendors are also integrated. A floor plan of the currently used demo room is shown in Figure 15. There are two areas, a work and a relax area, where the user can do various activities such as working on a computer, reading newspapers or sleeping. Based on the activity and on the user preferences the lighting in the room is adjusted.

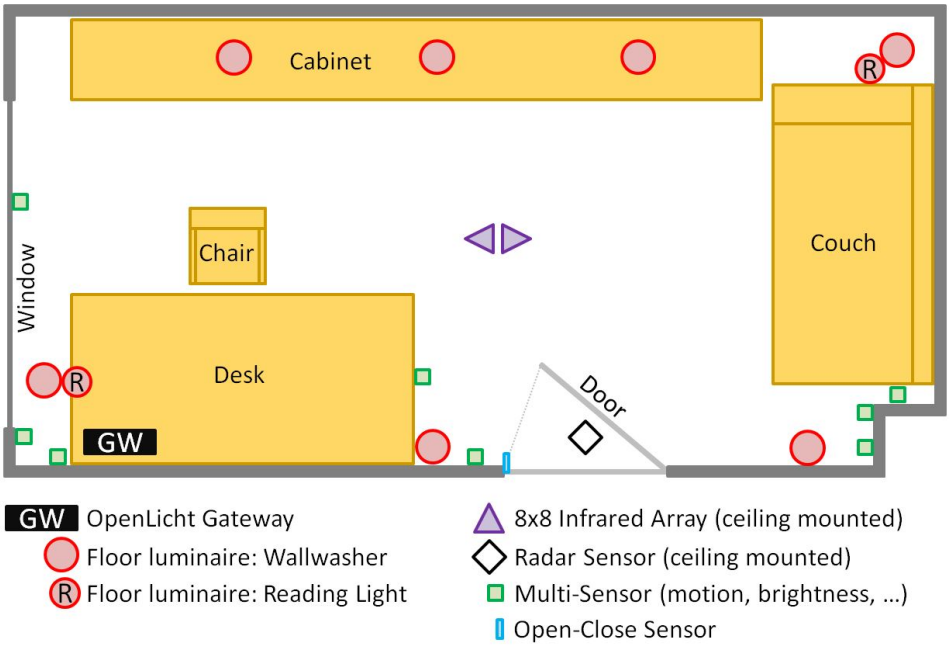

Figure 15. *OpenLicht* demo room

5.6 Conclusion

In this paper, we gave an overview about the research project *OpenLicht* and presented its main part an intelligent, self-learning lighting system.

The central requirements of the system are reliability, scalability, usability, privacy, support of various devices and communication protocols, as well as an intuitive user interface. Those goals are partly achieved by using the framework openHAB as system base and by employing the edge computing paradigm, which ensures that computation is done within the local network, to increase reliability and decrease latency. Moreover, machine learning algorithms are used to improve the usability by using them to automate the light control. For this automation, the applied algorithms learn user activities and preferences. Afterwards, the system can recognize the learned activities of the user and apply the fitting user preference pattern. The patterns are used to generate a light configuration plan. The infrastructure for the artificial intelligence of the OpenLicht system was integrated into openHAB as an extension module. Another component added, is the extensible *knowledge base*, based on Reference Attribute Grammars serving as a performant base for the whole system.

To support various communication protocols such as ZigBee, Z-Wave and Bluetooth Smart, radio modules, a Raspberry Pi single board computer and a specially developed extension board were combined to create a gateway.

The self-learning lighting system developed within OpenLicht will be evaluated in a demo room. Furthermore, small kits for users will be put together to evaluate a limited version of our OpenLicht system in their homes. With the results gathered during the evaluation, the software components will be finalized and possible limitations of the system will be determined. The hardware components and the *OpenLicht* platform will be assessed in the scope of fairs, competitions, and workshops in schools.

5.7 References

Ekman, T., & Görel H., 2007. The JastAdd System - Modular Extensible Compiler Construction. In: *Science of Computer Programming* 69, no. 1.

Feurer, M., Klein A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F., 2015. Efficient and Robust Automated Machine Learning. In: *Advances in Neural Information Processing Systems,* pp. 2962-2970.

Gama, J., Zliobaité, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A., 2014. A Survey on Concept Drift Adaptation. In: *ACM Computing Surveys (CSUR)* 46, no. 4.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H., 2009. The WEKA Data Mining Software: An Update. In: *SIGKDD Explorations Newsletter* 11, no. 1, pp. 10-18.

Heaton, J., 2015. Encog: Library of Interchangeable Machine Learning Models for Java and C#. In: *Journal of Machine Learning Research* 16, pp. 1243-1247.

Hedin, G., 2000. Reference Attributed Grammars. In: *Informatica (Slovenia)* 24, no. 3, pp. 301-317.

Higginbotham, S., 2018. *Rethinking the smart home in 2018.* https://staceyoniot.com/rethinking-the-smart-home-in-2018/ (accessed September 12, 2018)

IBM, 2016. An Architectural Blueprint for Autonomic Computing. *IBM White Paper*, 2016.

Ikea, 2018. *Ikea Tradfri Website.* https://www.ikea.com/gb/en/products/lighting/smart-lighting/ (accessed October 10, 2018)

Infineon, 2018. *XMC Microcontroller Website.* https://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-cortex-m/32-bit-xmc4000-industrial-microcontroller-arm-cortex-m4/?redirId=41425 (accessed October 10, 2018)

IoT Analytics, 2017. *State of the Smart Home Market 2017.* https://iot-analytics.com/wp/wp-content/uploads/2017/12/StateofSmartHomeMarket2017-vf.pdf (accessed September 12, 2018)

Mey, J., et al., 2018. Continuous Model Validation Using Reference Attribute Grammars. In: *International Conference on Software Language Engineering.* ACM, 2018, Boston, USA.

Mnih, V., et al., 2015. Human-level control through deep reinforcement learning. In: *Nature* 518, p. 529.

OpenLicht Consortium, 2018. *OpenLicht Website.* http://openlicht.de/ (accessed October 15, 2018)

Philips, 2018. *Philips Hue Website.* https://www2.meethue.com/ (accessed October 10, 2018)

PwC, 2017. *Smart home, seamless life: Unlocking a culture of convenience.* https://www.pwc.com/CISconnectedhome (accessed September 14, 2018).

Rad, B. B., Bhatti, H. J., & Ahmadi, M., 2017. An Introduction to Docker and Analysis of its Performance. In: *International Journal of Computer Science and Network Security* 17, no. 3.

Raspberry Pi Foundation, 2018. *Raspberry Pi Website.* https://www.raspberrypi.org/ (accessed October 10, 2018)

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L., 2016. Edge computing: Vision and challenges. In: *IEEE Internet of Things Journal* 3, no. 5.

VDI, 2018. *Open Photonik Website.* https://www.photonikforschung.de/projekte/open-innovation/foerdermassnahme/open-photonik-innovationsprozess.html (accessed October 10, 2018)

Zhang, Y., Zhang, H., Cai, J., & Yang, B., 2014. A Weighted Voting Classifier Based on Differential Evolution. In: *Abstract and Applied Analysis, vol. 2014*.

5.8 Acknowledgements

This work is part of the research project *OpenLicht* (project number 13N14047) and is funded by the German Federal Ministry of Education and Research (BMBF). We thank our partner Deggendorf Institute of Technology for their cooperation in *OpenLicht*.